Congestion Analysis and Management

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de rector magnificus prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op maandag 24 augustus 2009 om 16.00 uur

doors

Hylke Jurjen Lijsbert Westra

geboren te Voorschoten
Preface

This thesis marks an important event in my life. It puts an official end to the time I could consider myself a student. Looking back, I can only be grateful for opportunities I have been given and the people I have met.

First of all, I thank professor Patrick Groeneveld for our cooperation during my PhD research. I am especially grateful he has given me the freedom to find my own way and work on the topics I was interested in. I have enjoyed our conversations not only about work, but also about science, economics, politics and life in general.

I am grateful to professor Ralph Otten for introducing me to design automation and luring me into doing my Master’s work in his group at Delft University. Without his confidence in me and his enthusiasm my life would certainly have taken an entirely different course.

During my PhD research at Eindhoven I met many bright and inspiring people. This thesis is my work, but it would not have been possible to write it without them. Especially the students I have coached have left a mark in this thesis. I also have fond memories of the coffee table discussions, talking about nothing and everything. I am especially indebted to Marja de Mol. Without her help and persistence this thesis would not have been published.

There have been many more people that have contributed to my research in one way or another. Although many people in my direct environment do not really understand what I have been doing all these years, I would not have been able to write this thesis without them. Thanks to my friends for great times and good advice at times. I should also thank my family for their mental support and practical help when I needed it. And of course special thanks go to my girlfriend Dorien for her love and pleasant distraction all these years.
Contents

Preface i

1 Introduction 1

2 Design flows for integrated circuits 3
  2.1 Methodology and design flow 3
  2.2 Main design flow 4
  2.3 Physical design
    2.3.1 Classical physical design flow 9
    2.3.2 Magma physical design flow 10
  2.4 Congestion during the flow 12
    2.4.1 Congestion as a supply and demand problem 13
    2.4.2 Congestion as a constraint 14
    2.4.3 Congestion in different design steps 14

3 Congestion analysis and management 15
  3.1 Basic definitions and notions 16
    3.1.1 Tile model for congestion analysis 17
    3.1.2 Edge model for congestion analysis 19
    3.1.3 True and LZ freedom 21
    3.1.4 Vias and freedom 23
    3.1.5 Modeling standard cell wires, pre-routes and blockages 24
    3.1.6 Congestion maps 25
  3.2 The congestion estimation problem 25
    3.2.1 Comparison against global or detailed routing 25
    3.2.2 Wire length and congestion 26
  3.3 Congestion during placement 27
    3.3.1 Global placement 28
    3.3.2 Congestion management during global placement 31
    3.3.3 Detailed placement 32
    3.3.4 Congestion management during detailed placement 34
  3.4 Congestion during routing 35
    3.4.1 Global routing 35
    3.4.2 Congestion management during global routing 38
    3.4.3 Detailed routing 39
3.5 Congestion during logic synthesis .................................................. 40
  3.5.1 Impact of logic synthesis on congestion ................................. 41
  3.5.2 Congestion metrics .............................................................. 41

4 Steiner Tree Decomposition ......................................................... 43
  4.1 Problem formulation ............................................................... 43
  4.2 Previous work ........................................................................... 45
    4.2.1 Theoretical results and exact algorithms .............................. 45
    4.2.2 Heuristic approaches .......................................................... 46
    4.2.3 Requirements for steiner tree algorithms in a VLSI context .... 46
  4.3 Secondary criteria for MST and RSMT ......................................... 47
    4.3.1 Freedom, bends and vias ....................................................... 48
  4.4 The BOI algorithm ..................................................................... 49
  4.5 BOI with precalculation .............................................................. 51
  4.6 MST decomposition .................................................................... 52
    4.6.1 Maximum and minimum freedom MSTs .............................. 54
    4.6.2 Prim's MST algorithm .......................................................... 55
    4.6.3 Prim's algorithm for MaxFMST and MinFMST ..................... 55
  4.7 Extension to spanning graph ....................................................... 58
    4.7.1 Octal partitions .................................................................. 58
    4.7.2 Sweep line algorithm ............................................................ 58
    4.7.3 Spanning graph for MaxFMST and MinFMST ..................... 59
  4.8 Experimental results ................................................................. 60
    4.8.1 BOI with precalculation ....................................................... 61
    4.8.2 Effect of freedom on BOI .................................................... 62
    4.8.3 Effect of freedom on routing result ..................................... 63
  4.9 Summary and conclusions .......................................................... 64

5 Probabilistic congestion estimation ................................................. 65
  5.1 Objectives for congestion estimation .......................................... 65
  5.2 Previous work ............................................................................ 66
  5.3 Preliminaries .............................................................................. 67
  5.4 Lou's method .............................................................................. 69
  5.5 Improved congestion model ....................................................... 73
    5.5.1 Foundation of the model ....................................................... 73
    5.5.2 Usage of short wires ............................................................. 74
    5.5.3 Usage of flat wires ............................................................... 75
    5.5.4 Usage of L-shapes ............................................................... 76
    5.5.5 Usage of Z-shapes ............................................................... 78
    5.5.6 Combination of usages ........................................................ 80
    5.5.7 Properties of usages ............................................................. 82
    5.5.8 Blockages ............................................................................ 84
  5.6 Implementation .......................................................................... 85
    5.6.1 M-TCL implementation ........................................................ 85
    5.6.2 C++ implementation ............................................................ 86
  5.7 Experimental results .................................................................. 88
    5.7.1 Routing probabilities ........................................................... 89
## 6 Congestion estimation by fast degenerate global routing

### 6.1 Objectives for fast degenerate global routing

#### 6.1.1 Suitability of global routing for congestion estimation

#### 6.1.2 Weaknesses of probabilistic methods

#### 6.1.3 Metrics

### 6.2 Previous work

### 6.3 Fast degenerate global routing

#### 6.3.1 Degenerate routing graph

#### 6.3.2 Shortest-path algorithms and the choice for A*

#### 6.3.3 The choice for A*

#### 6.3.4 Rip-up and reroute

#### 6.3.5 Two-phase strategy

#### 6.3.6 Wire order

#### 6.3.7 Cost function

#### 6.3.8 Detour bounding

### 6.4 Experimental results

#### 6.4.1 Varying capacity

#### 6.4.2 Visual inspection

#### 6.4.3 Error maps

#### 6.4.4 Wrongly congested and wrongly uncongested edges

#### 6.4.5 Run time

### 6.5 Discussion

#### 6.5.1 Estimation refinement

#### 6.5.2 Blockages

### 6.6 Summary and conclusions

## 7 Global routing

### 7.1 Purposes of global routing

#### 7.1.1 Complexity reduction

#### 7.1.2 Wire delay estimation

#### 7.1.3 Congestion control and routability

### 7.2 Objectives for global routing algorithms

#### 7.2.1 Overflow and congestion distribution

#### 7.2.2 Bends and vias

#### 7.2.3 Detour and detour distribution

#### 7.2.4 Run time

### 7.3 Implementation and experimental setup

#### 7.3.1 Overview of the router

#### 7.3.2 Implementation details

#### 7.3.3 Benchmarks

### 7.4 Wire ordering

#### 7.4.1 Wire ordering with single wire optimal router

#### 7.4.2 Wire ordering based on freedom

#### 7.4.3 Experimental results
7.5 Cost function and routing tiebreakers .................................................. 131
  7.5.1 Tiebreakers versus cost scaling ......................................................... 132
  7.5.2 Information theoretic interpretation .................................................. 132
  7.5.3 A* with tiebreakers ................................................................. 133
  7.5.4 Cost function ................................................................. 133
  7.5.5 Tiebreaker true freedom ............................................................ 135
  7.5.6 Tiebreaker bends ................................................................. 141
  7.5.7 Tiebreaker random ................................................................. 143
  7.5.8 Distance to destination .............................................................. 143
  7.5.9 Experimental results ................................................................. 144

7.6 Wavefront expansion ........................................................................... 148
  7.6.1 Examples ....................................................................................... 150
  7.6.2 Using pseudo-edges to model potential steiner points ......................... 150
  7.6.3 Experimental results ....................................................................... 151

7.7 Improving the detour distribution .......................................................... 153
  7.7.1 Implementation .............................................................................. 154
  7.7.2 Time slack and detour bounds ......................................................... 154
  7.7.3 Experimental results ....................................................................... 155

7.8 Comparison against other tools ........................................................... 156
  7.9 Conclusions and discussion ................................................................ 158
  7.9.1 Main conclusions .......................................................................... 158
  7.9.2 Design-specific tuning .................................................................... 159
  7.9.3 Extension to 3D model with vias ...................................................... 159
  7.9.4 Routing after layer assignment ....................................................... 160
  7.9.5 Detour bounding of individual wires ............................................... 160

8 Concluding remarks ................................................................................. 161
  8.1 Outlook to the future .......................................................................... 162
    8.1.1 Incorporating congestion in the design flow ..................................... 162

References ................................................................................................. 165

Summary ..................................................................................................... 175

Samenvatting ............................................................................................. 177

About the author ........................................................................................ 179
Chapter 1

Introduction

The design of integrated circuits (chips) is a complicated process involving many design steps. During the design flow, a very abstract description of a chip is translated into a specification suitable for production on manufacturing equipment, using as much automation as possible. This automation has been enabled through the use of abstraction. Necessarily, some of the aspects that are important at a certain level of abstraction are ignored at higher levels of abstraction.

One such issue that is ignored during the larger part of the design flow is routing congestion. The congestion problem essentially represents the supply and demand problem for the metal wires that are used to connect the functional base units of the chip. The ever increasing design sizes in the semiconductor industry and the shrinking of feature sizes due to improved manufacturing technology have made this problem increasingly difficult to deal with. In recent technology nodes the amount of chip real estate that is necessary is no longer determined by functional units but by the demand for routing resources. Even with as much as 25% white space routability is not guaranteed. Effectively, congestion has become a decisive factor for the cost of integrated circuits.

In this thesis, early congestion estimation is proposed as a tool to help with congestion management. In current design flows global routers are often used as a congestion estimation tool and based on the results design steps such as placement, floorplanning and logic restructuring are evaluated and guided. The congestion estimators proposed in this thesis are much faster than global routing. They enable designers to more often evaluate the impact of design decisions on congestion. Additionally, due to their speed they can also be used in inner loops of other algorithms such as the ones used in the design steps mentioned above.

Another important topic of this thesis is bends and vias. Vias are used to connect different interconnect layers and are not only relatively likely to fail, but also may use space otherwise available to wires and hence impact congestion. In this thesis ways to reduce the number of vias during wire topology generation and global routing are proposed, a topic that seems to be largely ignored in the literature on (global) routing.

A recurring theme in this thesis is the use of tiebreakers. The guiding principle is explained as follows. For many of the problems dealt with in this thesis, the primary objec-

1 Usually, transistors, standard cells and analog elements are considered as such.
tives such as metrics for congestion are relatively well-understood\textsuperscript{2}. Many optimization
directions may be equally attractive with regard to the primary objective. In such cases
secondary objectives can be considered. Such an approach is used consistently to reduce
the number of bends during wire topology generation and routing, \textit{without impacting con-
gestion negatively}.

Another theme in this thesis is \textit{freedom preservation}. \textit{Routing freedom} is related to the
number of acceptable realizations a wire has. Routing is such a complicated design step
that no methods guaranteed to be optimal exist. Freedom analysis is used to implement
highly effective and efficient algorithms for global routing. Experimental results indicate
that the methods can be used to efficiently account for criteria such as congestion, over-
flow, bends and run time.

The research described in this thesis has resulted in a number of software tools and
scientific publications.

\textsuperscript{2}Although also on those primary objectives relatively large improvements compared to academic tools are
made.
Chapter 2

Design flows for integrated circuits

Integrated Circuits (ICs) are among the most complex systems designed by man. In the beginning of the electronic revolution, ICs consisted of a handful of transistors, the basic building blocks of ICs. Nowadays, Systems-on-Chip (SoCs) may consist of millions of transistors. This advance has been enabled by Electronic Design Automation (EDA) software. New levels of abstraction have been introduced by these tools. Nowadays, a chip designer does not have to design a circuit in terms of transistors anymore. Standard cell libraries have been introduced such that the basic building block of Application Specific Integrated Circuits (ASICs) is no longer the transistor, but a group of transistors called a standard cell. Such a standard cell implements a specific function in which chip functionality is to be expressed. As a next abstraction, high-level Hardware Description Languages (HDLs) were introduced such that a designer could describe the chip in an abstract language that is automatically transformed into standard cells, with the possibility to optimize different objectives. Historically, tool chains have been built up roughly in bottom-up fashion: the last steps in the design process were automated first.

In this chapter the design flow of ASICs is described. First, the main steps of the flow are sketched to show the big picture. This will give an idea of where the requirements for the algorithms and tools described in this thesis come from. Typically, there is little interaction between the main design steps since they are the domain of separate specialist design teams (often in different companies), and each of these steps encompasses a flow of its own. Next, the design step called physical design is discussed in more detail since this is the context of the central topic of this thesis: congestion.

2.1 Methodology and design flow

The methodology of the design process is the philosophy behind it. An example is a top-down design methodology. In such a methodology, the design is first specified in an abstract fashion. The details are filled in later. Such a methodology is characterized by refinement, i.e. details are added. A bottom-up methodology on the contrary focuses on

---

1 Especially in the early phases of process development, several teams may work on several issues simultaneously, and more interaction is necessary. This is achieved e.g. through a regular update of libraries.
the lowest level of abstraction first. First, the basic building blocks are designed, and from these blocks, the design as a whole is assembled.

A design flow can be considered as an instantiation of a methodology. Usually however, a flow is described as a tool chain or as a sequence of commands within a tool (a script). Therefore, a design flow is considered to be more practical and less philosophical about many issues.

The design flow of ASICs is usually a combination of a top-down and bottom-up flow, i.e. a meet in the middle methodology. Designers first specify functionality in terms of decoders, implementations of communication protocols and so on. Later, such blocks are refined in terms of basic operators such as multipliers and IDCTs. Eventually, the refinement process leads to a description in terms of standard cells. Such standard cells have been assembled bottom-up, typically by a company specialized in standard cell design in cooperation with a foundry (the place where the chip is actually manufactured). In some large Integrated Device Manufacturers (IDMs), standard cell design and foundry services can actually be part of the same company.

2.2 Main design flow

The main steps in a classical ASIC design flow are conceptual design, behavioral synthesis, logic synthesis, physical design, and mask preparation as outlined in Fig. 2.1. These steps are briefly discussed below.

- During conceptual design[118] the rough functionality of the IC is determined. This task is a challenging one because “soft” real world constraints and objectives have to be translated into a technical specification. On the one hand, economic issues such as time-to-market and competing products demand a short design cycle, low manufacturing costs, high yield, high speed, low power consumption etc., while on the other hand technology, tools and designers set constraints. At this level of abstraction, it is very difficult to set realistic targets. During the further design of the IC, constraints may need to be relaxed or iterated upon.

Manufacturing and design technology has advanced to the point where whole systems are integrated on a single ASIC. This is called a System on Chip (SoC). Such systems are very large and designing them is typically distributed over multiple design teams. This division is made during conceptual design. Other considerations are the possible use of Intellectual Property (IP) blocks and making sub-systems reconfigurable to be able to deal with changing standards and protocols. Essentially, the design evolves from a rough idea to a set of constraints and concepts of implementation that can be used downstream in the design flow. The issues involved are very complicated and cannot be effectively automated. In practice, much depends on the creativity and experience in the design team.

- Behavioral synthesis consists of the precise development of algorithms and architectures. This very challenging and difficult to automate, although progress is being

---

2The generation of cells on the fly is possible but seems to have lost its attraction. This is primarily the case because well-instrumented human designers can optimize cells better than generators. Given the fact that cell libraries are used many times it is worth it in practice to spend a considerable amount of effort in optimizing a library for a given manufacturing process.
made. The freedom at this stage of the flow is enormous and typically a designer has little more than simulators and common sense to make tradeoffs.

First, the conceptual design needs to be converted into a technical implementation. Nowadays often a C-based language is used, previously HDLs such as Verilog or VHDL were used to this end. The way an algorithm is described/implemented during this design step can have a huge impact on the quality of the final implementation. Typically, there is a library with previously synthesized parameterized components with power and performance numbers and algorithms that try to describe the originally specified functionality in terms of these components. **High-level synthesis** is a behavioral synthesis step that is successfully applied in the area of Digital Signal Processors (DSPs). Behavioral synthesis for the more general case of ASICs is an active research area.

The main focus is on implementing the correct functionality. Using simulators, the impact of design decisions on issues such as timing, power and area is estimated. It is also possible to perform hardware-software co-simulation.

Behavioral synthesis tools typically translate the C code into Register Transfer Level (RTL) descriptions using languages such as VHDL and Verilog. These languages are at a level of abstraction that contains all the functionality, but do not describe how the functionality is realized e.g. in terms of standard cells\(^3\). This RTL can then be used in an ASIC or FPGA\(^4\) implementation flow.

- **Logic synthesis** transforms an RTL description into a **netlist**: a list of nets that con-

---

\[^3\]It is possible to describe netlists in RTL, but this is not the output of behavioral synthesis tools.

\[^4\]FPGA: Field Programmable Gate Array. Essentially programmable hardware. In terms of efficiency and flexibility it is between an ASIC and a general purpose CPU such as a Pentium.
nect standard cells from a library. Logic synthesis largely determines the final chip performance, power numbers, and area. The final netlist may consist of millions of standard cells and the optimization problems associated with logic synthesis are very complex. Initially, logic synthesis focuses on mathematical criteria such as the number of literals or number of sub-expressions. Standard cells are characterized in terms of delay and power. This information is used to optimize during technology mapping, when boolean expressions are mapped onto a library. Additionally, logic synthesis must prepare the final netlist for physical implementation. Techniques such as standard cell sizing and buffering are therefore necessary.

- **Physical design** is sometimes referred to as *layout synthesis or place and route* because the main steps are giving locations to the standard cells (placement), and the realization of nets in terms of metal wires (routing). Essentially, it is the task of physical design to come up with the geometries of all polygons of the chip. The input of physical design—netlists—routinely consists of millions of standard cells and nets, and in practice most of the problems in physical design are solved using heuristic approaches.

ASIC and SoC design flows use *cell libraries*. This means that physical design does not need to design each individual transistor. Instead of designing individual polygons, it assigns locations to *groups* of polygons. Since the most complicated polygon patterns are typically found in standard cell libraries, this alleviates the task of physical design to some extend.

Since physical design is so close to manufacturing, it is subject to a large number of constraints on top of the constraints and objectives formulated by the designer. Contrary to previous stages, it cannot be assumed that things *average out*. If a single transistor switches too late as a result of a detoured wire, or a single wire has such dimensions that due to current densities it evaporates, the chip will fail. Constraints related to the manufacturing process and the laws of physics are specified in so-called *design rules*. In practice, many checkers and simulators are used to validate the design before entering the manufacturing flow.

- **During Mask Data Preparation (MDP)**, the design is preprocessed to compensate for non-ideal manufacturing equipment. Examples are *Optical Proximity Correction (OPC)*, *dummy fill insertion* and *assist feature insertion*. OPC compensates for limitations due to the properties of light and the lens system that is used during the manufacturing of chips. The purpose of dummy fill insertion is to ensure the average “hardness” level across the chip is roughly constant. Differences in hardness occur as a result of different densities of the different materials that make up the chip. Assist features such as *scatter bars* are polygons that do not print on the chip. 

---

5The resolution or *resolving power* of an optical system is determined by the Numerical Aperture (NA) of the lens and the wave length of the light that is used. In current state-of-the-art technologies (65nm and below) the patterns that are printed are well below the wavelength of the light that is used to print it. In those technologies, masks are 4 times larger than the printed patterns and light with a wavelength of 193nm is used. It is not easy to switch to lenses with higher magnification because of e.g. loss of depth-of-focus (DOF). It is also currently not feasible to switch to light with shorter wavelengths because of lack of good light sources and resist. More on this topic can be found in the literature on *optical lithography*.[61,86].

6After some processing steps excess material needs to be removed using *Chemical Mechanical Planarization*.[93]. The term “hardness” as used above must therefore be understood in this context.
Their presence on the mask impacts the diffraction of light in such a way that the intended image is enhanced. Besides the above, decisions regarding the properties of the mask need to be made. The output of mask data preparation is a specification of the masks that can be send to mask manufacturers.

Optimal circuit topology and standard cell sizes depend on the capacitive load that needs to be driven. In modern technology, wires represent a large part of this load. The length of the wires is determined by placement and routing. The traditional approach of selecting the standard cells entirely before place and route is therefore clearly sub-optimal. This has led to the integration of logic synthesis and physical design in physical synthesis. Essentially, steps from both design phases are interlaced and iterated upon in this approach.

When chip performance was primarily limited by the speed and number of standard cells, physical design merely implemented the netlist produced by logic synthesis. Now that wires have become performance-limiting factors, physical design has become a crucial step for timing closure and optimization. This development has made congestion—the topic of this thesis—a crucial factor for chip performance.

**Design freedom and refinement during the flow**

Design flows are based on refinement and some iteration. In the beginning of the design flow, there are typically few hard constraints and objectives. On the other hand, there is a lot of freedom regarding choices about the algorithms and architectures that are used. During the flow, constraints and objectives are constantly added, and choices are made. This results in reduced design freedom. During physical design, there is only limited freedom left (mainly the locations of the standard cells and the routes of the wires), but there are many constraints such as power and delay budgets and design rules. Essentially, the problem of physical design is that it must realize constraints and objectives that were made based on an abstract model that necessarily did not capture the full reality.

**2.3 Physical design**

The main tasks of physical design are the placement of standard cells and macros and the routing of nets (including power and clock nets). Typically, the optimization objective of physical design is power or area, subject to performance constraints. Other constraints are specified in the design rules, and include maximum current densities, minimum wire width and spacing, maximum wire length, and so on. State-of-the-art process nodes (65nm and 45nm) have very complicated design rules that require a two-dimensional analysis of wire patterns. Many ASIC designs are very complicated and push the performance envelope. Then, there is little room left to make tradeoffs, and physical design can be considered as the task of finding a feasible solution, i.e. a fully placed and routed design that does not violate any of the constraints set by the designer or the design rules.

---

7 Macros result from an hierarchical approach in which the design task of a chip as a whole is split up into parts that are design more or less independently from each other. These parts are called macros and are instantiated at the right locations to assemble the full chip.
A simplified view on physical design is illustrated by Fig. 2.2. A chip is implemented in a number of electrically insulated layers. Standard cells need to be given locations and consist of polygons primarily in ACTIVE and the lower METAL layer(s). The technology provides a number of routing layers (METAL1 \ldots METAL4) that can be used for the realization of the nets. A connection between two routing layers is made with a via. In modern processes, many more routing layers are available to designers (up to 9 currently), but since a larger number of layers corresponds to a larger number of processing steps and larger mask set, there is a clear incentive to use as little layers as possible. Also, addition of layers is not as attractive as it may look at first sight. Devices reside in the bottom active layers, and in order to access the higher layers, holes need to be made in intermediate layers. Typically, routing layers contain alternating horizontal and vertical wires. The lowest routing layers are used for short wires, while the highest routing layers are used for longer wires. Typically, the larger part of METAL1 can not be used for connecting standard cells because it is used for the internal wiring of the standard cells. The standard cells can be accessed through pins on the two lowest routing layers.

Routing power and clock networks requires special care. Modern chips dissipate lots of power. This translates in a voltage drop over the power distribution network that can cause the IC to malfunction. Similar things may happen in the clock network: current through the network impacts the delays on the clock network, and the clock may not arrive on time, causing functional errors.

Designers are assisted by many simulators throughout the design process. Functional correctness can be verified using formal verification, and timing engines calculate the delays at the different levels of abstraction. The power and clock networks can be checked for voltage drop, undesired delays and transient effects. Eventually, parasitic effects such as crosstalk can be analyzed by building physical models of the wires and standard cells.
Finally, the design is checked against the design rules by Design Rule Checking (DRC) engines.

### 2.3.1 Classical physical design flow

In a fast evolving industry with many companies using different approaches, it is difficult to sketch a “typical” physical design flow. However, there are a number of concepts that are commonly understood and used. In practice, designers have used scripts to iterate between design steps, and for most designs, custom scripts and flows are created. However, we mention the main steps in the order they are mostly used to give an impression.

**Floorplanning**

In many cases, tool capacity is not sufficient to handle a design as a whole, and essentially such a design is partitioned into several blocks that need to be implemented largely separately and at the end assembled together again. Top-down design methodologies or simply the use of different design teams may lead to the same situation. Based on estimates, chip area is assigned to each of the blocks, and resources for common infrastructure such as clock and power routing are created.

**Global placement**

Approximate positions are assigned to the basic building blocks of the chip (standard cells) during global placement. This gives an accurate idea on how the cells are placed relative to each other. The main objective of this design step has until recently been to minimize the amount of wiring needed. More recently, timing also became a consideration. Global placement will be discussed in more detail in Chapter 3.

**Global routing**

The task for global routing is to find locations for the wires connecting the devices, but only at a coarse level. Initially, this gives the designer an idea whether enough routing resources are available, and also how timing is impacted. Later, *detailed* routers use the global routing result as a start point. Global routing is one of the main topics of this thesis, and will be discussed in depth in Chapter 7.

**Detailed placement**

Detailed placement starts off where global placements stops. Essentially, cells are moved by small bits in order to assure there is no overlap. Detailed placement is discussed in Chapter 3.

**Detailed routing**

Detailed routing has the task of starting from a global routing result, push the wires such that they do not overlap. Also, connections to the access pins of the devices need to be made. A discussion on detailed routing can be found in Chapter 3 of this thesis.
Timing analysis

Timing analysis is run many times during the design flow. After each of the above mentioned design steps, it is run to check if the timing constraints are feasible. If not, the designer can use back-annotation to guide one of the earlier design steps. Also, designers can manually change things in order to solve the problems. Timing analysis is also used as a final sign-off tool to make sure no problems surface after manufacturing.

2.3.2 Magma physical design flow

As an example of a physical design flow, Fig. 2.3 shows the main steps in the Magma physical design flow[87]. Physical design flows from different vendors may differ, but all contain the same basic functionality. Sometimes, the organization of the tasks is different, but this flow gives a good idea of the methods and algorithms used in physical design.

We shortly discuss the steps of the flow.

- During fix time, the timing of the chip is estimated, based on simple models of the standard cells. After this step, timing becomes a constraint. The initial time budgets are based on the netlist only. No physical data such as placement or routing is needed. Physical design has a large impact on timing, but other optimization techniques such as sizing, buffering and logic restructuring will be used to meet the time budgets. At this stage, also optimizations such as buffering and logic restructuring are performed to improve timing or area.
Floorplanning is a highly interactive task during which important decisions such as the size of the chip are made. For the first time in the flow, physical data from the standard cell library is used. Basic infrastructure for I/O and power is created, macro cells are placed, and pin positions are assigned.

The fix cell step produces a placement for the cells, based on floorplanning information. First, global placement is used to get a rough idea on the positions of the cells. Based on this information, techniques such as buffering, logical restructuring, and cloning are used to optimize timing and area. The global router finds approximate paths for the wires, and creates more accurate load models for the cells. Again, techniques such as standard cell sizing and buffering are used to optimize the design based on the more accurate load information. Finally, detailed placement assigns exact positions to the standard cells.

The clock network is synthesized during fix clock. This involves creation and buffering of the clock tree, and routing the clock wires. The resulting clock network is not ideal. Therefore, the standard cells are sized again, based on arrival times of the clock signal. Then, the global router is run again to update load, timing, and congestion information.

During fix wire, the exact locations and widths of all wires is determined. First, short wires on the lowest routing layer are routed. Then, the global router is run on the remaining (long) wires, taking the already routed small wires into account. The next step is track routing, which orders the wires after global routing, and prepares them for detailed routing. Finally, detailed routing algorithms assign exact positions to the wires without violating spacing rules (if possible). Wires may be routed multiple times in order to refine the result, and create a violation-free design. The final result is a file in the GDSII format that can be sent to a chip manufacturer.

The flow as sketched above is based on refinement. Its main task is timing closure, i.e. meeting a given performance. The performance is limited most by the capacitive load that both standard cells and wires represent. Because of the inherent order in the different design steps, wire load has the greatest uncertainty associated with it. During physical design, the load models are constantly refined. At first, there is only some statistical wire load based on the fan-out\(^8\). After placement, distance-based load models can be used, and the netlist can be optimized for these distances. After global routing, there is information on which wires detour, and about cross-talk. Finally, after detailed routing, the exact loads can be extracted. During the flow, optimizations originally from logic synthesis such as buffering, cloning, sizing, and logic restructuring are used to adapt the netlist to the more accurate load model.

**Iteration**

The physical design flow as sketched above appears to be iteration-free. The idea is that based on available knowledge at the time, decisions are taken and used as constraints further down the flow. In practice, there is no push-button flow. Full automation has turned out to be difficult due to the many hard and soft constraints and the sheer size of designs.

---

\(^8\)The fan-out of a standard cell is the number of cells it drives.
Full automation may be possible, but will yield sub-optimal results. Typical chips sell in millions making more elaborate flows economically viable.

Decisions that are made early during the flow can have great impact on the remainder of the flow. They need to be taken in uncertainty about what happens downstream, and assumptions are made. It may happen that such decisions need to be revised after (partial) execution of the remaining part of the flow. During fix time for example, time constraints can turn out to be too ambitious. Another example is floorplanning decisions such as macro and pin positions. These decisions can have a large influence on timing, power and area numbers, and can turn out to be less than ideal. The exact implications are typically found only after fix cell, and a designer may have to change his or her floorplan. Congestion can also be the cause of unforeseen problems. Because placement fixes standard cell positions, the success of routing largely depends on it. Detailed congestion information is only available after some routing steps, and if the congestion problems are too severe, the designer needs to re-run the placer or adjust the floorplan.

If two design steps depend on each other, and co-optimization is not feasible, iteration is employed. Fig. 2.4 shows an example of iteration between cell sizing and global placement. The size of a standard cell is assigned based on the capacitive load it is driving. This load consists partially of the standard cells that are driven, and partially of the wires. When cell sizes are chosen for the first time, the load that must be driven is unknown, and statistical methods based on the fan-out are used to estimate it. However, after placement more refined loads can be calculated and it may be necessary to change cell sizes. Although there is generally no guarantee of convergence, usually good results are obtained. Sometimes manual interventions are necessary, and iterative flows are inherently slow.

### 2.4 Congestion during the flow

Congestion, the topic of this thesis, is the supply and demand problem associated with routing. It is not an optimization objective but a fundamental constraint: if there are too little routing resources, or the routing resources are not used well, some of the nets cannot

---

9 Iteration is often employed on chicken and egg problems. For a truly optimal result, steps involved in the iteration should be optimized simultaneously. Such a combined problem may be much harder than the sub-problems. In that case, a few iterations can actually be a relatively cheap method of obtaining good results. Additionally, between the iterations human intervention is possible to guide the algorithms.
be routed, and there is no correctly functioning chip. Although it is directly associated with routing, design steps earlier in the flow (most notably standard cell and macro placement) have a direct effect on it, and should take congestion into account.

### 2.4.1 Congestion as a supply and demand problem

The amount of routing resources is expressed as the number of *routing tracks*. The number of tracks maximally available to a router is determined by design rules that specify minimum wire width, spacing and pitch as illustrated by Fig. 2.5. Often, the minimum pitch is used to calculate the number of tracks for a given piece of routing area. Note that this is an overly simplistic view. In reality, wires of non-minimal width exist, and special *line-end design rules* apply when wires end. Thus, although a best-case resource estimation is possible, a more practical estimate will take issues such as mentioned above into account.

The demand for routing resources obviously depends on the locations of the standard cells. Thus, the demand depends on the result of the placement steps in the flow. Fig. 2.6 is an illustration of how standard cell locations can impact the routing demand. Note that horizontal and vertical wire pieces are typically routed on different routing layers, and that they therefore cannot short-circuit. The congestion problem during placement is well-known and has been targeted by making *wire length* the main objective for placement algorithms.

In (most) supply and demand problems, some resources are in higher demand than others. In the case of congestion, it happens that multiple wires are naturally routed through the same region of the chip. These regions represent the scarce resources. As illustrated by Fig. 2.7, an analysis is important because of the sequential nature of many routers.
2.4.2 Congestion as a constraint

In a sense, the design of an ASIC can be seen as an optimization problem yielding a specification for a chip manufacturer. Considerations such as performance, chip area, and power consumption can either be objectives or constraints. There are a number of hard constraints: the chip must be functional correct, and it must be possible to manufacture it. Congestion is clearly related to the latter, but it also has impact on other criteria.

2.4.3 Congestion in different design steps

The design flow as a whole is broken up in numerous steps and employs numerous algorithms. It is not practical to formulate the chip design process as an optimization problem with an object function and constraints and use these rigorously in all algorithms. In the early phases of the process there is hardly any awareness of congestion. Although congestion is considered to be a constraint during chip design, it makes sense to treat it as a minimization objective during design steps such as logic synthesis and (global) placement. The models that are used to model congestion during such steps are relatively inaccurate, and by using minimization successful routing is more likely.

In this thesis, congestion is modeled and minimized during several stages of a physical design flow such as steiner tree decomposition and global routing. In some cases, congestion is considered an optimization criterion while in other cases it is considered a constraint. It is important to have a general understanding of the flow an algorithm is used in since this is one of the main motivations for the models and metrics that are used.
Chapter 3

Congestion analysis and management

The design process of ASICs can be seen as a series of transformations. The final goal is a design that is “optimal” in some sense, e.g. performance, power or area. Some of the requirements on the chip are somewhat soft. If a power budget for instance cannot be realized, this may lead to reduced battery-life, but may still be acceptable.

There are two truly hard constraints: firstly, the design must have the desired functionality. The second constraint is that it must be possible to realize the design in silicon. At higher levels of the flow, guaranteeing correct functionality is relatively easy since transformations are correct by constructions. The second constraint is more difficult to deal with because many potential problems at the higher levels of abstraction.\(^1\)

The question whether it is possible to route the chip is related to the term routability. Early in the flow, it is not a concern, but as the flow progresses, it becomes increasingly important. Routability is binary in nature: a design can be automatically routed (with a given tool), or it cannot be automatically routed. Given the difficulty of the routing problem, it is impossible to predict routability accurately for all but the simplest cases. Because of the binary nature, improvements to the design for routing are not captured by a routability-based metric. A more gradual metric is congestion, which is the ratio between routing demand and routing resources, typically defined on small areas of the chip. Because the purpose of congestion and routability analysis is to assess physical feasibility, congestion analysis is typically based on estimates of routing demand and routing resources.

In the remainder of this chapter, some basic definitions and notions will be introduced. Then, the congestion estimation problem is introduced. Congestion is largely affected by placement, and therefore placement algorithms and ways to improve them for congestion are discussed. Finally, since congestion is obviously associated with routing, some common routing approaches, and how they deal with congestion, are discussed.

\(^{1}\)In fact, not having to deal with all kinds of cumbersome details is the whole point of using multiple levels of abstraction.
3.1 Basic definitions and notions

No matter what the optimization objectives at different stages of the flow are, a chip design eventually has to be realized in silicon.\(^2\) Given a tool set, it must be possible to refine a higher-level description eventually into a set of masks, with as much automation as possible. In this thesis we focus on routability which is defined as follows.

**Definition 3.1 (Routability).** A placed design is called *routable* for a given technology if a solution exists such that all nets are realized and no design rules are violated. A design is called *unroutable* if this is not the case.

Given a placement, routability cannot be easily verified. Numerous formulations exist for routing problems, but all practical formulations are NP-hard (see e.g. [115] for details). The term routability therefore in practice means something like *the probability of the design being routable, or the amount of effort that is needed to route the design.* The above notions are somewhat vague and are not easily quantified, although it is possible to use tools such as complexity analysis and cpu time measurements. When we discuss algorithms that improve routability in this thesis, typically arguments are given why routability is discussed, and experiments are conducted in order to quantify the effect on criteria such as run time, overflow and wire length.

Because of increasing design sizes, it has become increasingly difficult for designers to guide the routing process directly. They have to resort to indirect measures such as tool settings and routing blockages. Routability can only be tested by actually trying to route the design, which is a very time consuming design step. It is not practical to route it after every adjustment, but a designer is interested in the effect of design changes on routability, since there may be tradeoffs between optimization criteria such as power or area, and routability.

An indirect metric of routability is *congestion*, which is defined as follows.

**Definition 3.2 (Congestion).** The congestion \(C(A)\) of an area \(A\) with usage \(U(A)\) and capacity \(\mathcal{C}(A)\) expressed in the number of routing tracks is:

\[
C(A) = \frac{U(A)}{\mathcal{C}(A)}.
\]

Low congestion corresponds to high routability. Exact usages are not known until completion of the routing because of e.g. possible detours. For that reason, congestion analysis is often based on *estimates*. For the capacity typically the maximum number of available routing tracks multiplied by a factor is used. The number of tracks depends on the technology and the presence of blockages and pre-routes, and the factor accounts for the fact that not all wires are minimum width, and that additional spacing may be necessary due to design rules.

Sometimes, the routing resources are not sufficient. An area with \(C(A) > 1\) is called *over-congested*. Over-congested areas are generally unroutable, although routing capacity may have been modeled overly conservative. Note that the presence of over-congested regions on a chip does not necessarily mean the chip as a whole is unroutable since routing demand may be moved to other regions. Congestion analysis enables algorithms that spread congestion, and thus improve routability.

\(^2\)Technically, other materials are possible as well.
Definition 3.3 (Overflow). The overflow of an area $A$ is defined as

$$O(A) = \max(0, U(A) - C(A)).$$ (3.2)

Evidently, overflow corresponds to over-congestion, but is an absolute measure instead of a relative one.

The size of the regions on which congestion is estimated determines how useful a congestion estimate is. In practice these regions have roughly the size of a big standard cell. A chip is divided into an array of such regions called tiles. A large chip can be divided into an array of several hundreds by several hundreds, yielding tens of thousands of tiles.

When designers try to improve routability they try to improve the design for routing. Not only do they try to make sure the design as it currently is is routable, but they also try to decrease the sensitivity of routability to design changes. Congestion estimation can be of help here: routability improvement typically boils down to moving routing demand from high-demand regions to low-demand regions. If estimation is sufficiently fast, it is practical to run it often or use it within other algorithms.

3.1.1 Tile model for congestion analysis

Congestion is analyzed on areas of the chip. Thereto the chip is divided in tiles by a mesh as illustrated by Fig. 3.1. A tile is identified by its row and column coordinates $(r, c)$. A chip has multiple routing layers, and so do the tiles. Smaller tiles correspond to higher accuracy, but also to higher run times for congestion estimation algorithms. This is essentially the same tradeoff as the tradeoff that is made during global routing. In this thesis, the same tile size is used for congestion estimation and global routing, and this size is an input of our algorithms. Typically tiles are one cell row high, and (almost) square. In current technologies, this means roughly 10 parallel wires can be routed in a layer of a tile, also depending on the layer. The maximum horizontal and vertical capacities are calculated as follows:

$$C_{tile, max}^{hor}(r, c) = \sum_{l \in L_{hor}} \frac{H}{p_l},$$ (3.3)

and

$$C_{tile, max}^{ver}(r, c) = \sum_{l \in L_{ver}} \frac{W}{p_l},$$ (3.4)

where $L_{hor}$ and $L_{ver}$ represent the sets of horizontal and vertical routing layers, respectively, and $H$ and $W$ the tile height and width. Typically, the grid is uniform, i.e. all tiles have the same height and width, but if this is not the case, $H$ and $W$ can be replaced by $H(r, c)$ and $W(r, c)$ to account for the heights and widths of the different tiles. $p_l$ represents the routing pitch on layer $l$, i.e. the minimum distance at which wires can be routed in that layer. This pitch is a limitation of the manufacturing technology and is a given for our algorithms. Maximum capacities are usually scaled to find the capacity as used in congestion models:

$$\gamma_{hor} \cdot C_{tile, max}^{hor}(r, c),$$ (3.5)

$^3$Designs that cannot be routed do not necessarily have high average congestion. If tile size is too coarse the problematic areas may disappear in the congestion view as a result of averaging.

$^4$Other names for tiles include bins, buckets and GCells.
Figure 3.1: A chip is divided into tiles by a mesh. This mesh usually coincides with the power stripes that separate the cell rows.

and

\[ C_{\text{tile}}(r,c) = \gamma_{\text{ver}} \cdot C_{\text{tile, max}}(r,c), \]  

(3.6)

where \( \gamma_{\text{hor}} \) and \( \gamma_{\text{ver}} \) are typically between 0.8 and 1.

The horizontal and vertical usages of a tile are calculated as

\[ U_{\text{tile}}(r,c) = \sum_{w \in W_{\text{hor}}} \frac{\text{length}(w,r,c)}{W}, \]  

(3.7)

and

\[ U_{\text{tile}}(r,c) = \sum_{w \in W_{\text{ver}}} \frac{\text{length}(w,r,c)}{H}, \]  

(3.8)

where \( W_{\text{hor}} \) and \( W_{\text{ver}} \) represent the sets of wires in a horizontal and vertical layer, respectively, and \( \text{length}(w,r,c) \) represents the length of the wire \( w \) in tile \( (r,c) \). Note that this length is normalized for the tile width \( W \). This definition allows for using exact pin locations and exact wire lengths, resulting in fractional usages.

With the above, tile congestion is defined as

**Definition 3.4** (Tile congestion). The horizontal tile congestion of a tile with coordinates \( (r,c) \) is defined as the ratio between the number of used routing tracks in horizontal layers and the number of available routing tracks in horizontal layers. It can be calculated as

\[ C_{\text{tile}}(r,c) = \frac{U_{\text{tile}}(r,c)}{U_{\text{tile}}(r,c)} = \sum_{w \in W_{\text{hor}}} \frac{\text{length}(w,r,c)}{W} \gamma_{\text{hor}} \cdot \sum_{l \in L_{\text{hor}}} \frac{H}{P_l}. \]  

(3.9)

Similarly, the vertical tile congestion of a tile with coordinates \( (r,c) \) is defined as the ratio between the number of used routing tracks in vertical layers and the number of available routing tracks in vertical layers. It can be calculated as

\[ C_{\text{tile}}(r,c) = \frac{U_{\text{tile}}(r,c)}{U_{\text{tile}}(r,c)} = \sum_{w \in W_{\text{ver}}} \frac{\text{length}(w,r,c)}{H} \gamma_{\text{ver}} \cdot \sum_{l \in L_{\text{ver}}} \frac{W}{P_l}. \]  

(3.10)
Typically, the superscript \textit{tile} is dropped because it should be clear when we are dealing with tile congestion from the context (instead of edge congestion, as discussed in the next section).

![Figure 3.2: The two left-most tiles have the same congestion value, but the middle tile has two free routing tracks whereas the left-most tile essentially is blocked. On the right, the pre-route blocks pin access from the left which may cause congestion.](image)

\textbf{Accuracy of tile model}

In the tile model, the total usage and congestion of a tile are lumped in a single congestion value. The \textit{location} of the wires is not taken into account. Fig. 3.2 illustrates how important the locations of pins and pre-routes and blockages can be.

\subsection{3.1.2 Edge model for congestion analysis}

Congestion is obviously associated with routing, and in current design flows global routing is usually the preferred means of congestion analysis. The global routing problem as posed in this thesis is defined on a grid graph $G(V,E)$ such as shown in Fig. 3.3. There is a relation with the tile model discussed in the previous paragraph: the nodes correspond to the tiles in that model, and are consequently identified by two coordinates $(r,c)$, representing the row and column the node is in. The edges correspond to \textit{boundaries} between two neighboring areas, and are separated in a set of \textit{horizontal} edges $E_{\text{hor}}$, and a set of \textit{vertical} edges $E_{\text{ver}}$ such that $E = E_{\text{hor}} \cup E_{\text{ver}}$. A horizontal (vertical) edge is identified by its leftmost (bottommost) node: a horizontal edge $(r,c)$ connects the nodes $(r,c)$ and $(r,c+1)$. Both $E_{\text{hor}}$ and $E_{\text{ver}}$ can be represented by a matrix using those indices (see Fig. 3.3).

![Figure 3.3: The global routing graph for global routing and congestion estimation.](image)

Equivalently to tiles, each tile boundary also has a limited capacity for wires. The number of routing tracks that can pass the boundary is usually the same as the number of available routing tracks in both corresponding tiles\(^5\) and can hence be calculated the same way.

\(^5\)When not all routing resources are available due to restrictions imposed by the designer, this may be modeled as different capacities of tiles and edges.
Thus, the horizontal edge capacity is
\[ C_{\text{edge}}^{\text{hor}}(r, c) = \gamma_{\text{hor}} \sum_{l \in L_{\text{hor}}} \frac{H}{p_l}, \]  
(3.11)
and equivalently, each vertical edge has a capacity of
\[ C_{\text{edge}}^{\text{ver}}(r, c) = \gamma_{\text{ver}} \sum_{l \in L_{\text{hor}}} \frac{W}{p_l}. \]  
(3.12)
Note that although the formulas are the same as in the case for tile congestion, the interpretation is different. Routing tracks can be split into parts that are used by different wires, while tile boundaries cannot be shared.

The horizontal and vertical usages of an edge are simply the number of wires that cross the corresponding tile boundary:
\[ U_{\text{hor}}(r, c) = |B_{\text{hor}}(r, c)|, \]  
(3.13)
and
\[ U_{\text{ver}}(r, c) = |B_{\text{ver}}(r, c)|, \]  
(3.14)
where \( B_{\text{hor}}(r, c) \) and \( B_{\text{ver}}(r, c) \) are the sets of wires that cross the boundary between tile \((r, c)\) and \((r, c+1)\), and the boundary between tile \((r, c)\) and \((r+1, c)\), respectively. Note that this definition does not allow for fractional usages. In the tile model, routing tracks can be partially used contrary to the boundary crossing that is used here. In usage estimates however, it is possible to use fractions in both cases.

Using the above, edge congestion is defined as

**Definition 3.5 (Edge congestion).** The horizontal edge congestion of an edge with coordinates \((r, c)\) is defined as the ratio between the number of wires crossing the associated tile boundary, and the number of available routing tracks on that boundary. It is calculated as
\[ C_{\text{edge}}^{\text{hor}}(r, c) = \frac{U_{\text{edge}}^{\text{hor}}(r, c)}{C_{\text{edge}}^{\text{hor}}(r, c)} = \frac{|B_{\text{hor}}(r, c)|}{\gamma_{\text{hor}} \sum_{l \in L_{\text{hor}}} \frac{H}{p_l}}. \]  
(3.15)
Equivalently, the vertical edge congestion of an edge with coordinates \((r, c)\) is defined as the ratio between the number of wires crossing the associated tile boundary, and the number of available routing tracks on that boundary. It is calculated as
\[ C_{\text{edge}}^{\text{ver}}(r, c) = \frac{U_{\text{edge}}^{\text{ver}}(r, c)}{C_{\text{edge}}^{\text{ver}}(r, c)} = \frac{|B_{\text{ver}}(r, c)|}{\gamma_{\text{ver}} \sum_{l \in L_{\text{hor}}} \frac{W}{p_l}}. \]  
(3.16)

Typically, we will drop the superscript \textit{edge} because it should be clear when we are dealing with edge congestion from the context.

**Accuracy of edge model**

Edge congestion essentially indicates whether it is possible to connect the tiles as desired. Within tiles, exact pin positions are not taken into account. Nets that reside fully inside a tile are not taken into account at all. In practice, such inaccuracies have not been too problematic. Short wires are typically routed before global routing, and their usage is deducted from the routing resources. Additionally, tile sizes are sufficiently small.
3.1 Basic definitions and notions

3.1.3 True and LZ freedom

Global routing algorithms are very effective: the large majority of wires are routed without detours. Detours only become necessary when congestion starts to play a role. It obviously helps if a wire has more than one detour-free realization. This is captured in the concept freedom, as illustrated by Fig. 3.4.

We define two types of freedom.

Definition 3.6 (True freedom). The true freedom $f_{true}(w)$ of a wire $w$ is the number of detour-free realizations that exist for that wire.

Definition 3.7 (LZ freedom). The LZ freedom $f_{LZ}(w)$ of a wire $w$ is the number of detour-free realizations that exist for that wire with at most two bends.

The latter definition is motivated by the observation that routers in practice route the large majority of wires with at most two bends (see [137] and Chapter 5). Also, bend minimization is a specific goal of many algorithms in this thesis.

For a wire $w$ with pins at the row and column coordinates $(r_0, c_0)$ and $(r_1, c_1)$ in a global routing graph, and $(r_0, c_0) \neq (r_1, c_1)$ we find

Theorem 3.1 (True freedom).

$$f_{true}(w) = \frac{|r_0 - r_1| + |c_0 - c_1|!}{|r_0 - r_1|! \cdot |c_0 - c_1|!}$$ (3.17)

Proof. A route is represented by a path in the routing graph from $(r_0, c_0)$ to $(r_1, c_1)$. Since the route is detour-free, the path must consist of exactly $|r_0 - r_1|$ vertical and $|c_0 - c_1|$ horizontal edges, and the row and column coordinates of the nodes on the path must be monotonic along the path. With a step we denote moving from one node to one of its neighbors. Starting from $(r_0, c_0)$ we can construct a path to $(r_1, c_1)$ by taking steps. Since the paths we are interested in are monotonic and the routing graph is a two-dimensional grid graph, only two symbols (we use zeros and ones) are needed to code the path in a bit string (e.g. a 0 for increasing row coordinates, and a 1 for decreasing column coordinates).
Evidently, each possible bit string consisting of exactly $|r_0 - r_1|$ ones and $|c_0 - c_1|$ zeros encodes a unique detour-free path, and vice versa. The number of permutations of the bit string is
\[
\frac{(\#1s + \#0s)!}{\#1s! \cdot \#0s!} = \frac{(|r_0 - r_1| + |c_0 - c_1|)!}{|r_0 - r_1|! \cdot |c_0 - c_1|!}.
\] (3.18)

It is also easy to find that

**Theorem 3.2** (LZ freedom).

\[
f_{LZ}(w) = \begin{cases} 1 & r_0 = r_1 \text{ or } c_0 = c_1 \\ |r_0 - r_1| + |c_0 - c_1| & \text{otherwise,} \end{cases}
\] (3.19)

where $L_1(w)$ is the Manhattan distance between the pins of the wire.

**Proof.** The first case is easily verified.

For the second case we observe that there are exactly two one-bend routes. There are two kinds of two-bend routes: one with a horizontal, and one with a vertical “middle bar”. There are $|r_0 - r_1| - 1$ rows where the horizontal bar may reside, namely those rows $r$ where $r_0 < r < r_1$ or $r_1 < r < r_0$ (depending on the relative locations of the pins). By the same argument we find that there are $|c_0 - c_1| - 1$ columns where the vertical bar may reside. Together this yields

\[
f_{LZ} = 2 + |r_0 - r_1| - 1 + |c_0 - c_1| - 1 = |r_0 - r_1| + |c_0 - c_1|.
\] (3.20)

As expected we find that

**Observation 3.3.** $f_{LZ}(w) \leq f_{true}(w),$

since the LZ freedom represents a subset of the detour-free routes represented by the true freedom. Both $f_{true}$ and $f_{LZ}$ are symmetric with respect to row and column coordinates. Also, only the relative distances between the pins are of interest. Therefore, we will in some cases use $\Delta r = |r_0 - r_1|$ and $\Delta c = |c_0 - c_1|$.

**Relation between true freedom and LZ freedom**

In several algorithms in this thesis, freedoms are compared. Wires can for instance be sorted based on their true or LZ freedom. Usually, we will focus on true freedom. This is motivated by the fact that if a wire $v$ has a larger true freedom than a wire $w$, it will typically also have a larger LZ freedom. The converse is less likely true since all wires with equal length have the same LZ freedom, except for those wires with pins in either the same row or column. This intuition is captured in the following theorem.

**Theorem 3.4.** Let $N_{TF}(f) = |\{(\Delta r, \Delta c) : f_{true}(\Delta r, \Delta c) = f\}|$ and $N_{LZ}(f) = |\{(\Delta r, \Delta c) : f_{LZ}(\Delta r, \Delta c) = f\}|$ be the number of “different” wire topologies having the same true or LZ freedom $f$, respectively. Then there exists an integer $f_0$ such that $N_{TF}(f) < N_{LZ}(f)$ for all $f > f_0$. 
Proof. We use the fact that the true freedom is equivalent to the binomial coefficient\[6\]. For two natural numbers \(n\) and \(k\), the binomial coefficient is defined as

\[
\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}.
\] (3.21)

Using the transformation \(n = \Delta r + \Delta c\) and \(k = \Delta r\), the binomial coefficient transforms to the true freedom. The problem of finding the number of different wire topologies yielding the same true freedom now reduces to finding multiplicities of entries in Pascal’s triangle\[6\] since this triangle contains all binomial coefficients exactly once. The following bound has been established\[6\[1\].

\[
N_{TF}(f) = O\left(\frac{\log(f)}{\log\log(f)}\right).
\] (3.22)

Now consider the LZ freedom \(f_{LZ}(\Delta r, \Delta c) = \Delta r + \Delta c = f\). Given \(f\), this is true for any \(0 < \Delta r < f\) with \(\Delta c = f - \Delta r\), and therefore \(N_{LZ}(f) = f - 1 = O(f)\). Evidently, \(N_{TF}(f)\) is a slower growing function than \(N_{LZ}(f)\), which proves the theorem. \(\square\)

As an example, consider a 1000 \(\times\) 1000 grid. Ignoring the multiplicity of \(1\), the largest multiplicity for true freedoms is 6. The pin configurations are enlisted in Table 3.1. Evidently, even for the smallest freedom in the table (120) there is a much larger number of LZ freedom multiplicities.

<table>
<thead>
<tr>
<th>(f_{\text{true}})</th>
<th>((\Delta r, \Delta c))</th>
</tr>
</thead>
<tbody>
<tr>
<td>3003</td>
<td>(2, 76) (5, 10) (6, 8) (8, 6) (10, 5) (76, 2)</td>
</tr>
<tr>
<td>210</td>
<td>(1, 209) (2, 19) (4, 6) (6, 4) (19, 2) (209, 1)</td>
</tr>
<tr>
<td>120</td>
<td>b(1, 119) (2, 14) (3, 7) (7, 3) (14, 2) (119, 1)</td>
</tr>
</tbody>
</table>

Even though comparison by true freedom can be considered a “stronger” operation than comparison by LZ freedom, it does not follow that the LZ freedom is fully “covered” by the true freedom.

Observation 3.5. Given two wires \(v\) and \(w\), if \(f_{true}(v) < f_{true}(w)\), it does not necessarily follow that \(f_{LZ}(v) < f_{LZ}(w)\). An example is given in Fig. 3.5.

The true freedom and LZ freedom of the design as a whole is the sum of the true and LZ freedoms of all wires. Designs with higher freedom are expected to have higher routability, so we consider the amount of freedom in a design as a metric for routability.

### 3.1.4 Vias and freedom

For routability purposes, lots of freedom in a design is desirable. Unfortunately, higher freedom designs also tend to have more bends after global routing: wires with a freedom

\[6\] There is even a conjecture by Singmaster that states that \(N_{TF} = O(1)[117]\).

\[7\] One is the only number that occurs infinitely many times in Pascal’s triangle.
of 1 are typically routed \textit{without} bends, while wires with freedom exceeding 1 need \textit{at least} one bend.

Let us imagine a technology with two routing layers. All pins are on the first routing layer, and the first routing layer is used strictly for horizontal routing, and the second routing layer is used solely for vertical routing. A via is used to change layer. We divide the vias in two categories. The first set of \textit{pin vias} consists of the vias that are used to connect a pin to a vertical piece of wire, and the second set of \textit{bend vias} encompasses the vias due to bends in the global routing. Obviously, the number of bends in a global routing solution is a lower bound for the final number of vias. The number of pins forms an upper bound on the number of pin bends, resulting in the following bound.

\[
\#\text{bends} \leq \#\text{vias} \leq \#\text{bends} + \#\text{pins}. \tag{3.23}
\]

In Chapter 4 more formal bounds are deduced.

### 3.1.5 Modeling standard cell wires, pre-routes and blockages

Since the tile grid is (near-)uniform and the design rules are the same for each tile, horizontal (vertical) capacity should be the same in each tile or edge row (column). The reason that capacities are defined \textit{per tile} is that we want to be able to adjust the capacities of each individual tile or edge. Up to here it has been implicitly assumed that all routing resources are available for routing. In practice, this is not always the case. Typically, the larger part of the bottom two metal layers is needed for internal wires of standard cells. Additionally, power and clock nets and buses are often given special treatment and are typically routed \textit{before} ordinary signal nets. The resulting wires are called \textit{pre-routes} and must be taken into account by congestion analysis.

\textit{Blockages} are polygons in the design database indicating that routers cannot use certain resources. They are used to guide the routing process or to reserve resources for subsequent routing steps or potential adjustments later on.

Both pre-routes and blockages are accounted for in the capacity calculation. They occupy a number of tracks, and these are simply subtracted from the tile or edge capacities. Note that although they are somewhat similar for the router because they both affect capacity, they are entirely different with respect to design rules and electrical effects.
3.2 The congestion estimation problem

3.1.6 Congestion maps

Congestion is associated with areas on a chip. **Congestion maps** are visualizations of the congestion values. In this thesis, **bright colors** correspond to high congestion values and **dark colors** to low congestion values. Typically, congestion values are stored in a two-dimensional array or matrix. If a chip is divided in $m \times n$ tiles, tile congestion can be stored in two $m \times n$ arrays or matrices: one for horizontal congestion, and one for vertical congestion. Horizontal and vertical **edge** congestion maps are stored in $m \times (n-1)$ and $(m-1) \times n$ matrices, respectively. If a single congestion map is shown, it is usually the **maximum** of horizontal and vertical tile congestion.

If congestion maps are shown in a **Graphical User Interface (GUI)**, designers can select windows, and apply optimizations to that window in order to resolve congestion problems interactively. If the congestion maps are based on congestion **estimates**, designers can judge routability based on the patterns in the maps. If e.g. a few over-congested tiles are surrounded by relatively empty tiles, the design may be routable. If on the other hand, a large group of tiles is near capacity, routability improvement may be necessary. The patterns in the congestion maps are important, and an experienced designer is capable of making the right decisions based on the congestion maps. Automation based on e.g. pattern recognition has not been found to work (yet).

3.2 The congestion estimation problem

In current design flows congestion is typically assessed with global routers. This is known to be a slow process and a bottle-neck in the design flow, although routers can typically be accelerated with special options. Congestion estimation methods that are **much faster** than global routers are therefore useful. The congestion estimation problem is described as follows.

**Problem 3.1 (Congestion estimation).** Given a netlist, a placement area with dissection in tiles, a standard cell placement and a set of pre-routes, the **congestion estimation problem** is to find congestion estimates for the tiles (or edges) due to the remaining signal wires as accurately as possible, but at least one order of magnitude faster than global routing.

There are a number of issues associated with the evaluation of congestion estimation results. Some authors\[138, 110\] compare against **global routing results**, while others\[137, 69\] compare against **detailed routing results**. There is also little agreement on **how** results are to be evaluated. Most authors compare on a tile-by-tile basis\[69,137,110\], but in \[138\] it was argued that the areas that are potentially over-congested, i.e. the areas with a congestion value around 1, are of most interest since they determine routability.

3.2.1 Comparison against global or detailed routing

The most important task of global routing is to manage congestion\[139\]. In practice, global routing results in terms of congestion are very close to actual values after detailed routing.\[8\] Therefore, it is reasonable to compare congestion estimates with global routing results.

---

\[8\]This merely means that for routable designs detailed routing does not need to deviate much from global routing.
This also reflects the idea of refinement: global routing congestion maps are a refinement upon congestion estimation maps. Correlation with global routing results is important: designers will trust a method that correlates well with global routing results more than a method that does not. Obviously, if an estimation method does correlate well with detailed routing results and not with global routing results, something strange is happening: this implies that global and detailed routing do not correlate well.

The congestion as measured by global routing is edge congestion. The inherent inaccuracy of edge congestion is in the fact that it counts boundary crossings only. If exact pin positions are available, congestion estimation methods can potentially estimate track usage more accurately. Edge congestion results lack this level of detail and therefore such methods can be compared against detailed routing.

In this thesis, two different congestion estimation methods are developed. One method is developed to estimate tile congestion using exact pin positions, and is therefore compared against detailed routing results. The other method estimates edge congestion and is compared against global routing results. Also, an extension of the first method is used to estimate edge congestion and is consequently compared against global routing results.

### 3.2.2 Wire length and congestion

Wire length is often used as a first order metric for congestion. Especially in the pre-placement stages of the design, all kinds of methods exist to predict the wire length of the design. An overview of methods is given in [122]. Evidently, the total wire length in horizontal or vertical direction is equal to the summed tile usages in that direction:

\[
L_{\text{total}}^{\text{hor}} = \sum_{0 \leq r < m} \sum_{0 \leq c < n} U_{\text{tile}}^{\text{hor}}(r, c) \cdot W,
\]

and

\[
L_{\text{total}}^{\text{ver}} = \sum_{0 \leq r < m} \sum_{0 \leq c < n} U_{\text{tile}}^{\text{ver}}(r, c) \cdot H,
\]

where an \( m \times n \) tile grid is assumed. In the absence of blockages and pre-routes, the average horizontal and vertical congestion are linear with the total horizontal and vertical wire length.

**Theorem 3.6** (Average horizontal congestion and wire length). Let horizontal wire length be normalized for tile width, and each tile have the same horizontal capacity. Then, the average horizontal congestion is linear with the total horizontal wire length.

**Proof.** The average horizontal congestion on a \( m \times n \) grid is

\[
C_{\text{hor}}^{\text{avg}} = \frac{1}{mn} \sum_{0 \leq r < m} \sum_{0 \leq c < n} C_{\text{hor}}(r, c) = \frac{1}{mn} \sum_{0 \leq r < m} \sum_{0 \leq c < n} U_{\text{hor}}(r, c).
\]

Since all edges have the same capacities, we can write

\[
C_{\text{hor}}^{\text{avg}} = \frac{1}{mn \cdot C_{\text{hor}}} \sum_{0 \leq r < m} \sum_{0 \leq c < n} U_{\text{hor}}(r, c) = \frac{1}{mn \cdot C_{\text{hor}}} \cdot L_{\text{hor}}^{\text{total}},
\]

which is linear in the total horizontal wire length. \( \square \)
3.3 Congestion during placement

Evidently a similar theorem exists for vertical tile congestion. For edge congestion, the situation is slightly different. Edge congestion only accounts for tile boundary crossings, and ignores the amount of wiring within tiles.

In practice, even almost unroutable designs can have low average congestion (< 0.7). The problem with congestion is that typically only a limited number of tiles causes problems. Usually, a group of highly connected standard cells is placed together too closely, and the routers cannot find legal routes for the connections. Such places on the congestion maps are called congestion hotspots or simply hotspots. Hotspots cannot be predicted with wire length models only since wire length is associated with average congestion, whereas hotspots are associated with worst-case congestion. Congestion estimates can be seen as a refinement on wire length estimates since they add positional information to the wire length estimate.

3.3 Congestion during placement

Placement is the design step where standard cells are given locations. This stage is largely responsible for the congestion problems faced by the routers, although the structure of the netlist is obviously also important here. The main objective for placement is low wire length. The primary reason is that low wire length roughly corresponds to low wire delay. Obviously, minimum total wire length does not necessarily correspond to minimum total wire delay or minimum path wire delay. Nonetheless, in practice variations of minimum wire length using e.g. net weights or non-linear functions of net/wire length are used successfully. Algorithms more directly targeting objectives such as critical path delay optimization have not been very successful. Placement deals with very large problem sizes and the main objective function cannot capture the amount of detail needed for optimization of metrics more directly related to final chip performance without exceedingly long run times. The second reason is that low wire length directly corresponds to low average congestion as discussed previously and is hence good for routability. Routability has become such a problem that (close to) minimum wire length has effectively become necessary for many designs to be feasible. Timing-driven placement can only happen under relatively tight wire length budgets. A simplified version of the placement problem can be described as follows.

Problem 3.2 (Placement). Given a netlist and a footprint, the placement problem is to find non-overlapping positions for the standard cells on the cell rows such that the total wire length is minimized and the result is routable.

Placement routinely involves hundreds of thousands to millions of standard cells. All reasonable formulations of the placement problem are NP-hard and therefore heuristics are used. Traditionally, placement is performed in two stages: global placement and detailed placement. Global placement has a global view over the problem, i.e. it con-

9 Wire length estimates can be made for regions on a chip as well. Such methods however lack sufficient resolution due to wires between regions.

10 A footprint is the shape and size of the placement area, the positions of the cell rows, and the locations of the primary inputs and outputs, i.e. the pins.
siders all cells at once\textsuperscript{[11]}. All cells are given approximate positions that are allowed to have some overlap. The process of removing overlap is called \textit{legalization} and is performed by detailed placement. Additionally, detailed placement typically tries to improve timing, wire length and congestion by \textit{locally} moving and swapping a few cells.

In order to see how congestion problems are addressed during placement, the most common approaches to global and detailed placement are briefly discussed.

### 3.3.1 Global placement

Global placement is a relaxation of the placement problem that allows (some) overlap. Overlap should not be too great since it needs to be removed by the subsequent detailed placement step while maintaining the structure of the global placement. Note that from a pure wire length perspective a global placement with all cells on top of each other is usually (close to) optimal so some overlap management needs to be present. The existence of a legal placement not too different from the global placement can only be guaranteed if there is sufficient white space available to move the cells to legal positions. The placement area is typically divided in \textit{bins} (similar to tiles\textsuperscript{[12]}) and white space is guaranteed by setting \textit{density} constraints on the bins. The global placement problem is therefore formulated as follows.

**Problem 3.3** (Global placement). Given a netlist and footprint, the \textit{global placement} problem is to find positions for all standard cells such that bin densities do not exceed target densities, and total wire length is minimized.

Note that many papers do not formally present the problem that they are trying to solve. It has been accepted that heuristics are necessary and that many possible metrics exist for evaluating global placement quality. In practice many variations of the above formulation are used.

In [131] it is shown that reducing wire length during placement yields better results in terms of routability than using more direct objectives such as overflow. Typically, \textit{Half-Perimeter Wire Length} (HPWL)\textsuperscript{[13]} is used as the measure for wire length.

As illustrated by Fig. 3.6, a netlist can be represented as a \textit{hypergraph}. Each node represents a standard cell and has a width associated with it\textsuperscript{[14]}. Nets are a set of cells (pins) that are to be connected and are consequently modeled using a hyperedge. In some approaches, the hyperedge is modeled as a \textit{clique}\textsuperscript{[37]} or \textit{star}\textsuperscript{[128]}.

A recent overview on large-scale placement is given in [31]. Global placement algorithms can roughly be classified in approaches based on partitioning\textsuperscript{[13, 36, 15, 134, 145]}, simulated annealing\textsuperscript{[109, 108, 20]}, and analytical placement\textsuperscript{[37, 56, 68, 17]}.

\textsuperscript{11}This is not entirely accurate. In modern placement engines, tightly connected groups of cells are clustered into single \textit{placeable objects}. Under the assumption that such clusters will be placed in single tiles, between-tile wiring is still entirely present in the \textit{coarsened} netlist. This is the wiring responsible for the congestion global routers such as the one presented in this thesis target.

\textsuperscript{12}We distinguish between bins and tiles because of the different context and because bins may change in size during placement algorithms.

\textsuperscript{13}HPWL is half the length of the perimeter of the bounding box of the pins of a net. For two and three pin nets this is the length of the minimum length steiner tree.

\textsuperscript{14}Standard cells have the height of cell rows. In some approaches the nodes represent \textit{clusters} of cells, in which case the exact dimensions need to be known.
Partitioning-based placement

Partitioning-based placement is based on recursive bi- or quadrisection of both the netlist and the placement area. As illustrated by Fig. 3.6, each partition of the graph is assigned to a partition of the placement area. Usually, the objective is to minimize the number of nets that cross the cut, subject to constraints on the sizes of both area and graph partitions. The idea is that cells within a partition have higher connectivity, and should thus be placed close to each other. In theory the procedure can be applied until each standard cell has its own region but in practice some constraints are relaxed and an additional legalization step is necessary. Well-known academic placers based on partitioning are Capo[15] and Fengshui[145]. In recent years the approach has become more scalable by adaptation of the multilevel paradigm[70].

Compared to other approaches partitioning-based placers appear to be relatively effective when it comes to congestion mitigation, also due to advances in white space management[15][55, 16, 2]. One explanation is that partitioning minimizes cut sizes and these may capture local routing demands better than a metric such as total wire length. A weakness of the approach is that due to its recursive nature it cannot always recover from bad initial choices.

Graph and hypergraph partitioning has widespread application. The Fiduccia-Mattheyses algorithm[40] is a well-known algorithm in this field. In recent years great advances have been made using the multilevel paradigm. The hMetis tool by Karypis[70] is state-of-the art and used within placement algorithms. The outcome of a global placement algorithm can be viewed as relative positions of the placeable objects. Slicing algorithms such as described in [127] maintain these relative positions and remove overlap. These algorithms have been used mainly in the context of floorplanning but recently similar concepts are used for placement[80].

Simulated annealing-based placement

Simulated annealing[74] is a general optimization technique that resembles the annealing process of a crystal and is based on randomization. Starting from a feasible solution, random moves such as swapping two cells are considered. A move may or may not be

\[15\]White space is the complement of utilization. Uniform white space allocation is considered to be good for congestion and physical synthesis but has been shown to be bad for wire length[16]
accepted based on its effect on the objective and a temperature parameter. Early on during the algorithm the temperature is high and even moves that degrade the solution are accepted. As the temperature cools down degrading moves are less likely to be accepted. Simulated annealing can yield very good results but usually requires very long run time.

Recently, simulated annealing has lost much of its initial popularity because it is considered either too slow or too unstable in terms of quality of solution in comparison to analytic and partitioning-based approaches. Apparently, the method does not scale very well. Methods based on simulated annealing are still used to improve the output of other placers. Recently, simulated annealing is used more in the context of detailed placement than in the context of global placement.

In the Timberwolf placer\[109] the object function of the annealing process is a combination of wire length and overlap. The moves include cell swapping, cell moving and changing the cell orientation\[16]. NRG\[108] models the placement area as a set of coarse bins and the global placement problem consists of assigning standard cells to these bins. The object function consists of total wire length and a penalty for uneven densities between the bins. The moves are cell swaps. The Dragon placer\[134] combines simulated annealing with partitioning, effectively limiting the problem size for simulated annealing. In [17] a basic simulated annealing placement procedure is combined with the multilevel paradigm. This enables the possibility to combine standard cell placement with macro placement.

Analytic placement

Analytic methods are typically based on an analytic expression for wire length. Multi-pin nets are decomposed into a set of two-pin connection using the complete graph spanning all the pins. Quadratic wire length, i.e. \( WL = \sum_w \text{length}(w)^2 \), is expressed in a matrix equation as

\[
WL = \frac{1}{2} x^T Q x + d^T x + \frac{1}{2} y^T Q y + d^T y + \text{const},
\]

which is minimized by solving

\[
Qx = 0 \text{ and } Qy = 0,
\]

where \( x \) and \( y \) represent the vectors containing the \( x \)- and \( y \)-positions of the standard cells respectively and \( Q \) and \( d \) are a matrix and a vector representing the connectivity between movable cells and between movable cells and fixed pins, respectively. The problem of minimizing wire length is equivalent to finding the equilibrium of a system of connected springs, where each wire is modeled as a spring according to Hooke's law. This approach is therefore often referred to as force-directed placement.

Recently, log-sum-exp expressions that more closely approximate half-perimeter wire length have been found\[68,17]:

\[
WL = \sum_{n \in N} \alpha \left( \ln \sum_{(x,y) \in n} e^{x/a} + \ln \sum_{(x,y) \in n} e^{-x/a} + \ln \sum_{(x,y) \in n} e^{y/a} + \ln \sum_{(x,y) \in n} e^{-y/a} \right),
\]

\[16\]Pins are not necessarily at the center of the cell. Therefore, rotating or mirroring the cell may reduce wire length.
where \( N \) represents the set of all nets and \( x \) and \( y \) the coordinates of the pins of a net. The smoothing parameter \( \alpha \) is used to adjust the numerical properties of the expression favorably.

Low wire length solutions typically have too much overlap. Bin structures are used to measure how well cells are spread. Common methods to improve the spreading use density-related forces\[37\] or the addition of pseudo pins that pull cells away from high-density areas to low-density areas\[56, 128\]. Other methods include adding a density-related penalty function to the objective and formulating density as a constraint. Such formulations are solved using constrained minimization techniques\[17\].

Analytical methods using quadratic formulations were pioneered in \[23, 105\], resulting in Proud. Gordian\[63\] improved overlap minimization by combining the method with partitioning. The Kraftwerk\[37\] placer uses density-related forces to force a more equal spreading of the cells over the placement area. A recent state-of-the-art academic placer is APlace\[68\] which uses the log-sum-exp expressions discussed above. It solves very large problems with low wire lengths. FastPlace\[128\] on the other hand employs a number of greedy optimization techniques such as cell-based moves in combination with pseudo-pins to obtain reasonable wire lengths in very low run time.

Typically, iterative methods are used to balance the wire length and spreading objectives. After each iteration it is possible to adjust the formulation such that less overlap results in the next iteration. Analytic placement has also been integrated in a multi-level framework\[17\].

### Other methods

Hybrids and combinations between some of the methods discussed above have been tried. Most notable is the combination of partitioning with analytical placement\[129\]. In this approach the partitioning of the netlist does not depend on the cut size only. First, all cells are assigned positions with an analytic formulation. Then, based on these positions, the netlist is cut.

In \[58\], randomization is used to select a small group of mobile cells. Wire length of nets incident to these cells is minimized by a linear programming formulation, ignoring density. Too high density is resolved by “rippling” cells from high-density areas to low-density areas. A user parameter determines how many times this optimization is carried out.

### 3.3.2 Congestion management during global placement

The placement problem is formulated as wire length minimization subject to routability. The latter constraint is often interpreted as density targets or constraints. Several approaches to target congestion during global placement exist although some argue that post-processing methods are sufficient\[131\].

Congestion is treated in much the same way as overlap during global placement. Most methods rely on the availability of intermediate placements that are analyzed for congestion using congestion estimation methods as described in this thesis. The constraints

\[17\] In many constrained optimization methods, active constraints appear as penalty functions in the objective function so some methods are more related than it may seem at first sight.
and/or object function of the placer are altered based on this analysis. Intermediate placements are typically available because most global placement algorithms are based on refinement. Partitioning-based algorithms refine by further partitioning and analytical placers refine by spreading more.

**Cell bloating**

Placement algorithms use the size of standard cells to avoid too much overlap or too high densities. Congestion can often be mitigated by lowering cell densities in congested areas. Cell bloating techniques such as used in [54,12] achieve this by assigning a *virtual* area to cells. Bloating values can be based on properties of the cell such as pin count and area, but also on the congestion level at the location the cell has in an intermediate placement. Bloating algorithms require tuning to the placement algorithm and the congestion estimation method.

**Density targets**

Density is the ratio between cell area and placement area. Bin densities can be changed by moving cells or bin boundaries. Bin densities can be part of an object function or constraints. Similar to cell bloating, the fact that lower density usually corresponds to lower congestion can be exploited by setting density targets based on congestion estimates.

*White space allocation* is used in the context of partitioning, and has received much attention in recent years[5,141,16,2]. It is easy to change sizes of area partitions during partitioning based on expected congestion levels. In analytic placement, congestion levels are successfully used to adapt the penalty function for expected density [68]. In [37], density-driven forces are used to remove overlap. These forces are easily adjusted for congestion levels. Density is really used as a constraint in [17] and it is easy to see how congestion-based target densities can be used here.

In recent years most placers are evaluated using real routers. Consequently they need congestion management. The placer presented in [22] is based on multi-level analytic placement. Partitioning is used to evaluate congestion and the resulting slicing tree is used to redistribute white space. Another analytic placer[120] uses a simple bounding box-based method for congestion estimation and uses the resulting congestion map in a generic supply and demand framework to optimize routability.

### 3.3.3 Detailed placement

The detailed placement problem is illustrated by Fig. 3.7. Standard cells are laid out in *cell rows* such that they can easily share power and ground wires. In order to enable this all cells must have *the same height*, but may differ in width depending both on functionality and drive strength[18]. In some models cell rows are divided in discrete *sites* to which standard cells are assigned. These sites are typically the *size* of the smallest standard cell in the cell library and cells may occupy multiple sites. In the general case standard cells may have overlap and are not placed on rows or sites.

---

18 Modules with larger height may exist, but in the context of detailed placement these modules are considered to have fixed positions, i.e. they have been placed before.
Detailed placement can have different meanings and different constraints and objectives in different flows. Ignoring issues such as timing and signal quality the detailed placement problem is described as follows.

**Problem 3.4 (Detailed placement).** Given a netlist, a footprint with cell rows and a global placement, the *detailed placement* problem is to find non-overlapping positions for the standard cells on the cell rows such that the total wire length is minimized and the result is routable.

Note that the detailed placement problem is formulated almost the same way as the placement problem (Problem 3.2). The purpose is that detailed placers are allowed to make changes to the placement structure. They have an additional requirement (non-overlapping placement on cell rows) but they also get an additional input (the global placement). Detailed placement focuses on small parts of the layout only. Thus, the global placement structure is maintained and detailed placers are often able to further reduce wire length because they can find the global optimum for the small problem they deal with. Although an important step, detailed placement has received much less attention than global placement.\(^{19}\)

Many approaches to detailed placement exist, also depending on what the global placer produces. In some flows cells are first *row assigned*. This step is then followed by *final legalization*.

\(^{19}\)This may be the case because of lack of benchmarks, clear definition of the input and well-established objectives.
Row or region legalization

Typically, cells are first assigned to a cell row or a sub-row called a region. Given target densities, the row/region legalization problem can be stated as a supply and demand problem. Min-cost flow techniques are commonly used to solve the problem [130]. A greedy approach is presented in [58], where a minimum-cost path is constructed from a supply to a demand region. Cells “ripple” from supply to demand according to this path. In [65], cells are greedily moved from supply rows to demand rows, according to the increase in wire length.

Partitioning-based placers typically try to align cut lines with the cell rows and cells are automatically assigned to a row [14]. Assignment to regions is based on the white space allocation techniques discussed earlier.

Final legalization

For partitioning-based approaches it is possible to calculate optimal placements for the smallest sub-problems using branch-and-bound techniques [14]. Typically, cells have only partially overlapping positions within a row or region, i.e. they have an order. Given this order linear programming techniques are used to assign optimal positions [130][20]. In [58] such a sequence is split in two sub-sequences for a region. Then, these sub-sequences are optimally interleaved using dynamic programming. In [65], the legalization problem is formulated as a shortest-path problem on a large graph containing all cells and sites as nodes. Different weights on the edges make minimization of either perturbation or wire length possible.

3.3.4 Congestion management during detailed placement

The congestion management techniques that are used during detailed placement rely on the same technique as the techniques used during global placement: lowering cell density where congestion is high. The difference is that during detailed placement potentially more accurate congestion information is available since the approximate positions of the cells are fixed. Note that this is not very different from the final stages of many global placement algorithms.

An important class of algorithms that targets congestion is used between traditional global and detailed placement. These algorithms are typically window-based, so we classify them under detailed placement.

Row balancing

Although many global placement algorithms have target densities, detailed placement algorithms typically start with similar objectives when assigning cells to cell rows. Obviously, target densities can be altered based on the observed congestion levels. By explicitly formulating supply and demand rows densities can be reached using greedy [65] or flow-based exact methods [130]. At this stage, simple target densities are not sufficient to deal with the remaining congestion problems. Methods such as [131, 133, 143] only move

---

20 Essentially, white space allocation is performed here.
specially selected cells such that some congestion related cost is reduced, thus more directly targeting congestion. Randomized methods including simulated annealing seem to be favorite in this area.

Final legalization

There are only few methods that remove congestion in the final stages of detailed placement. Given relative positions for the cells, methods such as [65] remove overlap while minimizing wire length of perturbation. An approach worth mentioning is [62], where congestion is reduced in the final detailed placement stage by combining it with global routing.

Detailed placement improvement

Several detailed placers focus on improving results obtained by other tools. In [102] the primary objective is wire length while [103] uses a sophisticated model to improve timing as well. Both methods can be characterized as \textit{try all possibilities} on a small area. This is feasible for [102] since only a limited number of moves on a pre-existing detailed placement are proposed while [102] uses a Brand-and-Price approach.

3.4 Congestion during routing

Congestion is the main problem of routing. The decisions of routers impact congestion in two ways. Obviously, by choosing the routes over the chip they define the demand for routing resources. Secondly, wires and vias may \textit{block} other wires. While many considerations such as timing and yield may be considered, the main objective of routing is simply to find ways to connect all pins as desired without violating design rules.

\textbf{Problem 3.5} (Routing). Given a netlist, placement with locations for all pins and a technology with a given number of routing layers and design rules, the \textit{routing problem} is to connect all nets such that no design rules are violated. Among all solutions, the one with least total wire length is best.

The number of nets is in the same order of magnitude as the number of standard cells: up to millions. Reasonable formulations of the routing problem are NP-hard\cite{107,115}, and heuristic approaches are used in practice. Typically, the same approach as with placement is used: routing is split up in \textit{global routing} and \textit{detailed routing}. Global routing finds coarse routes and has a \textit{global} view over the problem. Its main task is congestion management\cite{139}. The detailed routing task is to generate the polygons that connect the pins, given the coarse routes defined by global routing. We briefly study the most common approaches to global and detailed routing.

3.4.1 Global routing

The main task of global routing is to find approximate routes over the chip area and to provide detailed routers with a starting point. Global routing is an abstraction that necessarily ignores many details of final wire implementation including wire sizing, spacing
and tapering\textsuperscript{21}. As argued in [139], the most important goal for global routing is to spread congestion. Essentially, designs with less congestion hotspots provide more robustness and freedom for the tools downstream in the design flow.

**Problem 3.6** (Global routing). Given a netlist and global routing graph with capacities on the edges, the *global routing* problem is to find paths through the graph for all nets such that the congestion is spread best.

In many cases it is easy to decide which of two routing solutions is best. In some cases however, one solution may have *lower average congestion* but *higher maximum congestion*. In such cases it cannot always be predicted which solution is best for the detailed routers.

Routing resources are often modeled as a three dimensional grid graph as shown in Fig. 3.8 although the grid graph is not necessarily regular. In this thesis a separate *layer assignment* step is assumed, enabling us to use the two dimensional model illustrated earlier by Fig. 3.3. Layer assignment is often based on the length of wires or on speed considerations. The upper layers are often manufactured in a different process with different design rules and are usually faster. By assigning primarily long wires to the upper layers, both via counts and longest-path delay are minimized. Layer assignment is outside the scope of this thesis. Details can be found in [18, 73, 3, 104, 89, 26]. The assumption of a two-dimensional global routing graph is used often in the recent literature ([71, 4, 51, 104, 89, 26]).

Although vias are often ignored in the routing literature, global routing essentially determines the amount and locations of vias in a design\textsuperscript{22}. Fortunately, routing solutions with low wire length and evenly spread congestion will usually have low via counts as well. This is largely explained by the following theorem.

**Theorem 3.7** (Minimum bends - minimum length). A minimum-bend path in a global routing graph is also a minimum-length path.

*Proof*. If the begin and end nodes of the path are in the same row or column, there exists a single minimum-bend path with zero bends. This is also the only minimum-length path.

\textsuperscript{21}In many cases considerations such as delay, cross-talk and vias are also effectively ignored by global routing algorithms.

\textsuperscript{22}One can also argue that (global) placement decides the relative positions of pins to each other and the (minimum) length of wires. Thereby it puts a lower bound on the number of vias that are necessary.
Otherwise, the minimum-bend paths have one bend. There exist two such paths which are also minimum-length.

Bend minimization is an explicit goal of many algorithms presented in this thesis. Although typical routing algorithms may yield minimum-bend paths in many cases this is not always the case in the presence of congestion.

Until recently global routing not been such a vibrant research topic as for instance (global) placement. An excellent survey can be found in [57]. Only the last few years global routing has seen an increase in interest. Research has especially been bolstered as a result of the ISPD global routing contests. In this chapter the most common approaches to global routing are shortly reviewed. In Chapter 7 our own global router is presented.

Maze routing

Most popular methods for global routing appear to be based on maze routing (e.g. [79, 115, 51, 104, 95, 89, 26]). The basic approach is summarized as follows.

1. sort nets (usually based on length estimates)
2. for each net
   (a) break up net in two-pin wires
   (b) find a minimum-cost path
   (c) update the costs in the routing graph with the path
3. As long as there is overflow (rip-up and reroute)
   (a) rip (a number of) net(s)
   (b) route net(s) as in 2

In some algorithms nets are not broken up, but multi-pin algorithms are not always feasible for large nets. Wires are typically broken up using Minimum Spanning Tree (MST), Rectilinear Steiner Minimum Tree (RSMT), or Rectilinear Steiner Arborescence Tree (A-tree) algorithms. Step 2b is in maze routing based on single-source shortest path algorithms[32].

The success of maze routing depends on many implementation details including the way congestion is modeled in cost functions. The weakness of the approach is due to its sequential nature. Optimal paths for a single net or wire are (relatively) easily found but the order in which the nets and wires are routed is important (this is known as the net ordering problem). This problem is countered by using rip-up and reroute (R&R), but there is no guarantee that bad initial choices can be recovered from.

Multi-commodity flow based approaches

Global routing problems can be modeled as multi-commodity flow problems [32]. Such problems are defined on flow networks with source and sink nodes for each commodity and capacities on the edges. Several flavors of the problem exist, including minimum cost multi-commodity flow and maximum multi-commodity flow. In the case of routing, each

---

23 All paths between a source node and all other nodes are minimal in an A-tree.
24 The name maze routing stems from the observation that finding a path through a routing graph is similar to finding a path through a maze.
wire represents a commodity that needs to be transported from one node to another. Fractional flows are obviously not allowed. The approach has the advantage that there is no net-ordering problem. Unfortunately multi-commodity flow problems are NP-complete for integer flows\cite{39}.

The approach of \cite{4} is an advanced router based on multi-commodity flow techniques. In the algorithm, nets are represented by steiner trees that are generated on the fly. Since a net may have only one such implementation each steiner tree gets a 0-1 variable associated with it. Initially, the 0-1 constraint is relaxed such that an algorithm based on the maximum concurrent multi-commodity flow algorithm of \cite{49} can be used. Randomized rounding techniques are used to ensure a single steiner tree is used for each net. The approach does not necessarily yield very good solutions, so R&R techniques are used to further improve the results.

It appears that global routers based on algorithms for solving multi-commodity flow problems are used in the industry for difficult instances. They are slow in comparison with maze routers but may have higher quality of results, although no consensus exists on this.

### 3.4.2 Congestion management during global routing

The main purpose of global routing is congestion management\cite{139}. Typically, the global routing task consists of two steps. First, nets are broken up in two-pin wires. This is also called topology generation. Next, the individual wires are routed. For each of these steps methods to take congestion into account exist. In \cite{57}, methods to improve the way the two steps interact are discussed.

**Congestion-aware topology generation**

Topology generation is usually based on minimum-length trees since low length corresponds to low average congestion. In \cite{11} flexibility is created in steiner trees. An edge is flexible if it has more than one minimum length route. In this thesis the concept is generalized in the concept of freedom.

In some cases congestion is more severe on horizontal edges than on vertical edges or vice versa. This is often an artifact of e.g. the placement algorithm that is used or asymmetric routing resources. In \cite{3} an approach to layer balancing based on the transformation shown in Fig. 3.9 is proposed.

Recently a method to generate the topology based on congestion maps has been proposed in \cite{95}. Essentially any congestion map can be used. The authors demonstrate that improved topology helps to reduce congestion and run time.
3.4 Congestion during routing

Congestion management during routing

In global routing approaches based on sequential routing of wires, congestion management relies mainly on the interplay between wire order, cost function and rip-up and reroute strategies. All popular approaches contain highly heuristic parts and typically a lot of tuning is necessary to find an “optimal” scheme. Although shortest wires or nets are typically routed first, wire (net) order has been shown to be of only little influence in [30]. In [30] a number of different cost functions is evaluated and a large difference in effectiveness is observed. R&R has been researched extensively, but much research is placed outside the current context of technologies with many routing layers.

Global routers based on flow approximation algorithms may have congestion either as a constraint, e.g. minimize total wire length subject to capacity constraints, or as an objective, e.g. minimize the maximum congestion on any routing edge. Sometimes also cost functions similar to the ones in sequential routing are used and the problem is formulated as a cost minimization problem. The approach in [4] shows that R&R can typically be used to improve the results further. For the R&R engine the same approaches as in sequential routing can be used although in practice R&R strategies may be tuned to observed behavior of a given router.

An interesting and successful approach to overflow minimization during global routing is [51]. In this approach costs are not only based on current congestion levels but also on congestion estimates. Congestion estimates are amplified and a cost component is added. A more or less traditional maze router is used to route the wires. During R&R the amplification factor is decreased and wires will move more and more into the regions that are hard to route. Effectively, this approach works because the wires that are routed first do not see an “empty cost landscape” but see costs based on the scarceness of a routing resource. Other routers including [47] use historic congestion numbers to guide R&R, which is somewhat similar.

Recent directions

The recent Maize router[89] introduces extreme edge shifting and edge retraction. These are greedy techniques that move segments of a routed wire in order to improve congestion. BoxRouter[26] is based on ILP formulations and maze routing. Its most interesting novelty is that it routes congested areas first. It also incorporates historic congestion costs in its cost function. The FGR router[104] is based on Discrete Lagrange Multipliers. Interestingly, A* algorithms are used to improve net topologies and avoid congestion. A router called NTHU-Route introduces wire ordering based on analysis of congestion maps[47].

3.4.3 Detailed routing

Detailed routing is the last implementation step in a physical design flow. It is somewhat beyond the scope of this thesis since it does not have a global view over the congestion. Like in the case for detailed placement many considerations including timing are important. Let us formulate the basic problem.

Problem 3.7 (Detailed routing). Given a set of nets, a global routing result and a set of design rules, the detailed routing problem is to connect all nets abiding the global routing result and the design rules.
The detailed routing problem is not formulated as an optimization problem. Its task is simply to connect all nets without violating design rules25 and the global routing result should be respected, i.e. the route should remain in the tiles of the global route. Due to very difficult to interpret design rules and the fact that new processes have increasingly more design rules, detailed routing is becoming increasingly hard. The problem is difficult to abstract and there is relatively little published research in this area.

**Track routing**

The first step of detailed routing is typically *track routing*. Each tile has a number of tracks (based on design rules) and a number of global routing segments associated with it. Track routing assigns global routing segments to tracks, i.e. it assigns an order to the global routing segments. A segment may span several tiles and should therefore (ideally) be assigned to the same track in all these tiles. Track routing therefore typically works on a single row or column of tiles at a time. It is closely related to the *channel routing*107 and *graph coloring* problems. Track routing determines which (long) routes are neighboring and can therefore be used to prevent signal integrity problems. Connecting the end-points of the segments to the pins and short nets is left to downstream routers.

**Gridded area routing**

Gridded approaches to detailed routing use a mesh similar to the one used for global routing to model routing resources. If two points are to be connected, a routing graph is created for the local area. Typically, the preferred direction constraint is somewhat relaxed: non-preferred direction edges exist but have a (much) higher cost than preferred direction edges. Then, algorithms similar to maze routing can be used to connect the nets. Optimizations such as wire sizing and -spreading and making room for vias are typically performed in a post-processing step. Typically, R&R methods are needed to connect all wires in dense areas.

**Gridless approaches**

Contrary to gridded approaches, gridless approaches build a graph model of the routing resources *on the fly* taking obstacles including previously routed wires into account. Gridless approaches have the advantage that it is easier to incorporate e.g. non-uniform wire widths than in gridded approaches. Much effort is spend in building the graph model of the routing resources. Once the model is constructed similar algorithms as in gridded approaches can be employed. Lots of R&R is typically needed to complete all wires. An excellent overview is given in [28] and the references therein.

### 3.5 Congestion during logic synthesis

This thesis deals primarily with post-placement congestion analysis. Logic synthesis is not considered. However, in modern design flows more and more integration (or often

25In advanced technologies preferred or recommended design rules may also exist. Violating such rules may reduce yield and in such a case the detailed routing problem may be formulated as a yield maximization problem.
iteration) between different stages is used to mitigate problems such as congestion. In this section a brief overview of congestion metrics for logic synthesis is given. Most of these metrics can be optimized within logic synthesis algorithms.

### 3.5.1 Impact of logic synthesis on congestion

Logic synthesis is traditionally targeted towards area, delay and power minimization or a combination thereof. Recent papers such as [121, 96, 112, 101] show that ignoring congestion is not a good idea and being congestion-aware during synthesis is essential for later stages. The following observations are made.

- Poor choices during logic synthesis may lead to unroutable designs.
- Traditional logic synthesis metrics such as literal count or gate size are not sufficient for routability assessment.

### 3.5.2 Congestion metrics

Congestion is associated with locations on the chip. At the logic synthesis stage little or no positional information is available. Most congestion metrics analyze a graph model of the boolean network or netlist. Let us refer to such a graph as an LS graph (Logic Synthesis graph).

- **A priori wire length estimates.** Even without a placement it is possible to estimate wire length. These methods are usually statistical and based on empirical observations such as Rent’s rule [78]. Since wire length correlates to average congestion such estimates are used to optimize designs. Stroobandt [122] is an excellent reference in this field.

- **Literal count.** The number of literals corresponds to the number of nets in a circuit. Lower number of literals therefore usually corresponds to better routability [94]. Many nets are absorbed in the standard cells the circuit is mapped to and therefore this metric is very coarse.

- **Intrinsic shortest path length.** Consider a net in an LS graph. The intrinsic shortest path length is the distance (a function of the number of hyperedges) between the pins after removal of the net. This metric is used for wire length estimation in [66].

- **Mutual contraction.** During logic synthesis an edge in an LS graph gets a mutual contraction value associated with it based on the edge degree of its nodes. The lower those numbers, the higher the mutual contraction. High mutual contraction therefore is associated with low wire length. This metric has been used to improve placement and technology mapping. Details can be found in [55, 82, 83].

- **Adhesion.** The adhesion value of a circuit has been shown to match peak congestion well [94]. This value is calculated as the sum over the min-cuts of all node-pairs in an LS graph. It is very expensive to calculate but appears to be relatively accurate.

- **Edge-separability.** Edge-separability is the minimum of all min-cuts as mentioned above and is used in [29].
• **Topological depth.** An acyclic graph representing a netlist can be traversed in topological order. The topological distances between nodes in the graph can be viewed as an estimate of the expected wire length after placement and routing. Wire length improvements and fan-out optimization due to such analysis have been reported [126, 83].

• **Neighborhood population.** When many cells are near a given cell in the netlist or LS graph congestion is likely. Methods such as described in [98, 99] calculate the number of cells at a given distance for each cell. These neighborhood populations are used to estimate wire length and layout area. More recently, the metric was used to guide logic synthesis [76].

In practice most of the above methods are useful but correlation to actual congestion is relatively low. Especially when the above metrics are used while performing technology-independent logical restructuring of the boolean network very high accuracy is not expected. During technology mapping (also known as library binding, see Chapter 2.2) many “nets” are absorbed in standard cells. Therefore it is difficult to predict which literals will actually contribute to congestion. It is important to realize that methods that directly or indirectly use wire length as a metric for congestion are not able to capture the localness of congestion problems. Methods based on analysis of the structure of an LS graph are usually better able to achieve this.

**Companion placement**

Recently, methods employing a companion placement have been successfully applied in the context of logic synthesis [112, 113]. The graph representation of the boolean network is treated as a netlist and placed using a coarse placement algorithm. The resulting placement is evaluated using congestion estimation methods as presented in this thesis. Logic synthesis operations are evaluated by the impact that they have on the congestion estimate.

---

26In the context of technology mapping the boolean network usually consists of two-input NAND gates and is known as the subject graph.
Chapter 4

Steiner Tree Decomposition

Both the congestion estimation techniques and the global router as discussed in this thesis are based on the analysis and processing of two-pin nets. Therefore it is necessary to break up multi-pin nets into two-pin wires. Methods based on Minimum Spanning Trees\[32\] are used in several congestion estimation methods[137, 110] and in numerous global routers. MST algorithms however do not guarantee a net is broken up in a set of wires with minimum total length as illustrated by Fig. 4.1. This is a serious drawback since total wire length is associated with average congestion.

The problem of connecting a set of pins with a tree of minimum length is known as the steiner tree problem\[60\]. If all edges of the tree need to be on a rectilinear grid\[1\] it is known as the Rectilinear Steiner Minimum Tree problem (RSMT)\[48\]. Compared to MST algorithms RSMT algorithms have the additional freedom of adding new pins (steiner points) to the tree. Although MST algorithms of complexity $O(n \log n)$ (where $n$ denotes the number of pins in the net) are known (e.g. \[50\] or \[147\]) RSMT is more complex and known to be NP-hard\[48\].

RSMT is a well-established problem in computer science and many theoretical results are known. The application in this thesis (congestion estimation and eventually routing) has the special property of small problem size\[43, 137\]. Additionally, computer science generally focuses on generating individual steiner trees while in our application the quality of result depends on the combination of multiple created trees. As first observed in \[11\] this results in a demand for flexibility in the generated trees. In that paper an incremental algorithm is developed that can be used on any previously generated steiner tree. In this chapter, a method to address the problem more rigorously during the core steiner tree algorithm is proposed. Additionally, the algorithm can be used to minimize bends.

4.1 Problem formulation

Let us more formally consider RSMT. Consider a grid graph $G_g(V_g, E_g)$. Essentially, this is a global routing graph as shown in Fig. 3.3. Each node $n$ has row and column coordinates

1A rectilinear grid consists exclusively of horizontal and vertical edges. The $\ell_1$ distance between two nodes on such a grid is known as the manhattan distance.
(r_n, c_n) associated with it, and each edge e ∈ E_g is between two neighboring nodes in either the same row or the same column and has a length of exactly 1 (note that this assumption is not strictly necessary, but it will be implicitly assumed throughout this thesis). A net N is a subset of nodes of the grid graph: N ⊆ V_g. The cardinality |N| of the net is the size of the set and is also known as the number of pins in the net.

A complete graph C_N(V_N, E_c,N) associated with a net N consists of the set V_N of nodes of the net and a set of edges E_c,N = V_N × V_N containing all combinations of nodes of the net. It is easily seen that |E_c,N| = \(|V_N|(|V_N|−1)\)/2. Each edge e = (a, b) has a rectilinear length \(L_1(e) = |r_a - r_b| + |c_a - c_b|\), where \((r_a, c_a)\) and \((r_b, c_b)\) denote the row and column coordinates of a and b in the global routing graph, respectively. Each edge also has a Euclidean length \(L_2(e) = \sqrt{(r_a - r_b)^2 + (c_a - c_b)^2}\) associated with it. Although the nodes of the net are present in the grid graph, this is not necessarily the case for the edges. This model is also known as the clique model (although a clique is only a complete sub-graph).

Let us define the Minimum Spanning Tree problem (MST) [32].

**Problem 4.1 (MST).** Consider a grid graph G_g(V_g, E_g), a net N with associated complete graph C_N(V_n, E_c,N) and a length function \(L(e)\) for each edge e ∈ E_c,N. The Minimum Spanning Tree problem is to find a set of edges E_MST ⊆ E_c,N such that \(G_{MST}(V_N, E_{MST})\) is a spanning tree and \(\sum_{e \in E_{MST}} L(e)\) is minimized.

A length function \(L(e)\) maps an edge to its non-negative length. If the function \(L(e) = L_1(e)\) is used, we speak of a rectilinear MST problem. If \(L(e) = L_2(e)\) is used we speak of a Euclidean MST problem. In this thesis the MST problems are associated with routing and a rectilinear routing style is assumed. Therefore, when an MST problem or algorithm is mentioned it is implicitly assumed we are dealing with the rectilinear case. MST is one of the best-studied problems in combinatorial optimization and more background information can be found in [32]. Efficient \(|n|\log|n|\) algorithms for rectilinear MST exist, e.g. the divide-and-conquer method of Guibas and Stolfi[50] and the sweep line algorithm of Zhou et al.[147].

Let us now define the Rectilinear Steiner Minimum Tree (RSMT) problem analogously to MST.

**Problem 4.2 (RSMT).** Consider a grid graph G_g(V_g, E_g) and a net N with associated complete graph C_N(V_N, E_c,N). The Rectilinear Steiner Minimum Tree (RSMT) problem is to find a set of nodes V_RSMT ⊆ V_g, and a set of edges E_RSMT such that \(V_N \subseteq V_RSMT\), \(G_{RSMT}(V_{RSMT}, E_{RSMT})\) is a tree, and \(\sum_{e \in E_{RSMT}} L_1(e)\) is minimized.

The solutions to MST and RSMT are topologies rather than routes: exact routes (paths) through the grid graph are not required. When \(|V_{RSMT}| > |V_N|\) the additional points (V_st =...
V_{RSMT} \backslash V_N) are called steiner points. Although not explicitly stated in RSMT it is usually preferred to have as few steiner points as possible since this gives the downstream routers more freedom to distribute congestion. Note that typically many solutions exist for RSMT since each edge with a length longer than 1 can be replaced by a minimum-length path between its two nodes through the graph.

The MST of a net provides both an upper and a lower bound on the length of the optimal steiner tree[60]:

$$\frac{2}{3} \sum_{e \in E_{MST}} e \leq \sum_{e \in E_{RSMT}} e \leq \sum_{e \in E_{MST}} e. \quad (4.1)$$

In practice, the improvement of RSMT over MST algorithms is relatively small. In the context of VLSI design where most nets consist of only few nodes the average improvement is in the order of only a few percent. Note that an RSMT is an MST over an extended set of nodes.

The above definitions of MST and RSMT imply that for each net its associated complete graph is available. Generally this is not the case and a net is defined in terms of pins that are associated with nodes in the grid graph. Generating the complete graph has a time and storage complexity of $O(|V_N|^2)$ and may be the bottle neck in MST and RSMT generation. In practice the complete graph is implicitly specified by the node set $V_N$.

## 4.2 Previous work

The rectilinear steiner tree problem is one of the best-studied problems in computer science and there is a wealth of literature, see e.g. [60, 44] and the references therein. An extensive overview of the field is beyond the scope of this thesis. Instead, a brief overview of mostly recent work on rectilinear steiner tree generation is given.

### 4.2.1 Theoretical results and exact algorithms

Hanan proved that only a limited set of nodes in the grid need to be considered for RSMT construction[53]: the nodes on the so-called hanan grid. Informally, this grid is defined by drawing horizontal and vertical lines through the nodes to be connected. A number of methods to reduce the number of potential steiner points even further are discussed in e.g. [60, 144]. An important theorem by Hwang[59] limits the number of topologies for full sets of nodes to two. A full set of nodes is a set for which all RSMTs only use the nodes as leaf nodes.

A simple algorithm for generating optimal steiner trees is based on enumeration of possible steiner points and calculating the MST[52]. Dynamic programming approaches such as [35, 75, 33, 125, 124, 46, 45] are based on a decomposition theorem stating that the problem can be split in two smaller steiner tree problems. Methods to exploit planarity are discussed in [9]. This is of interest since in the rectilinear case the resulting steiner trees are planar. Branch-and-bound algorithms for RSMT such as [144, 116, 8, 60] branch on either the inclusion of edges or the inclusion of steiner points in an optimal steiner tree.

In the area of exact steiner tree algorithms the work on GeoSteiner[136, 135] is worth mentioning. This tool contains many algorithms including LP solvers and is the fastest available tool for general steiner tree generation with guarantee of optimality. Another
algorithm worth mentioning is Flut3me\cite{27}. It is based on lookup tables and very fast. Because of size limitations of the lookup table large nets need to be broken up without guarantee of optimality. Recently a congestion-aware steiner tree decomposition based on Flut3me has been proposed\cite{95}.

### 4.2.2 Heuristic approaches

In the Iterated 1-Steiner heuristic of Kahng and Robins\cite{67} at each iteration optimal steiner points are greedily added to an MST. These points are found by exploiting the fact that an optimal steiner tree is also an MST. Although the generated steiner points are individually optimal, there is no guarantee on global optimality. The heuristic produces good quality of results in practice but is slow compared to later approaches.

The BOI algorithm of Borah, Owens and Irwin\cite{10} iteratively performs edge-based updates on a node-edge combination of an MST. The edge is “bended” towards the node using a steiner point, and a new edge between the steiner point and the node is added. The largest edge on the thus created cycle is removed. The method is popular in the VLSI community and has high-quality of results.

The algorithm presented by Zhou\cite{146} is an improvement over the previous method by combining it with the concept of spanning graphs\cite{147}. (Strong) spanning graphs are sub-graphs of the complete graph associated with a net containing at least one MST, but possibly more. They are constructed using a sweep line algorithm. Next, an MST is found by applying Kruskal’s algorithm\cite{32} on the spanning graph. The order in which edges are added to the MST in Kruskal’s algorithm is used to find the largest edge on a cycle created by a BOI update. The resulting algorithm is a bit faster than BOI, and also appears to have slightly better solution quality, although this could be coincidental.

The batched-greedy approach (BGA) of Kahng, Mandoiu and Zelikovsky\cite{64} is somewhat comparable to BOI. Updates on an MST are in this case based on triples consisting of three nodes rather than on node-edge combinations. By connecting such a triple with two new edges, two cycles are created. The largest edge on each cycle is removed. Sophisticated algorithms and data structures are used to generate the triples and find the largest edges. BGA produces slightly better results than a basic BOI implementation and is superior in terms of run time, although it is difficult to decide which part of the performance gain is to be attributed to its fast MST algorithm and which part to the RSMT algorithm.

### 4.2.3 Requirements for steiner tree algorithms in a VLSI context

Most research on steiner trees has focused on large sets of points. The reason is that those problems cannot be solved optimally in reasonable run time and are therefore more interesting for developers of heuristics. In VLSI design, the requirements are somewhat different as stated in the following observation.

**Observation 4.1.** The large majority of nets in VLSI designs have low pin counts. Run time of steiner tree algorithms therefore depends more on implementation details and constants than on theoretical worst-case bounds.

By far the most signal nets contain only two pins\cite{137}, and because of limited drive strength of standard cells only very few nets have more than 10 pins. Examples of such
nets are power, clock and special signal nets such as a global reset. These nets have entirely
different constraints (e.g. voltage drop and skew) and need to be treated differently than
the signal nets targeted by the algorithms in this thesis.

VLSI designs consist of many nets. Most of these nets are broken up using steiner tree
algorithms as discussed in this chapter, and the resulting wires are routed subsequently.
As stated in the next observation, this is why steiner tree decomposition is linked to con-
gestion.

Observation 4.2. Steiner tree decompositions are used for (global) routing. Thus, steiner
tree decomposition gives a degree of freedom for congestion optimization.

The above consideration is not well-understood and not considered at all in most re-
search on RSMT. There is another link between steiner tree topology and VLSI manufac-
turing that is not usually considered in steiner tree research.

Observation 4.3. If an edge in a steiner tree is not rectilinear, i.e. the two nodes are not in
the same row or column, this will result in at least one via in conventional technologies.
Thus, steiner tree decomposition can be used for via optimization.

4.3 Secondary criteria for MST and RSMT

The objective of the classical formulations of MST and RSMT is to minimize wire length. In
many cases more than one minimum wire length tree exists. Average congestion can not
be sacrificed in early stages of the design, but this observation provides us with a degree
of freedom that will be exploited in this chapter.

The potential for optimizing the number of bends during steiner tree decomposition
is illustrated by Fig. 4.2,left. On the right it is shown how tree decomposition may solve or
cause congestion problems. Usually, there is little or no congestion information available
when the topologies are generated and congestion is usually the result of routing many
nets and their decomposition. Although iterative methods are possible, the optimization
of topologies based on available congestion maps is not considered in this chapter.

![Figure 4.2](image)

**Figure 4.2:** Different steiner decompositions may require a different
number of bends (left) or solve congestion problems (right).
4.3.1 Freedom, bends and vias

The true (LZ) freedom of a tree $T(V, E)$ associated with a net $n$ is the sum of the true (LZ) freedoms of the edges in the tree.

$$f_{true}(T) = \sum_{e \in E} f_{true}(e).$$ (4.2)

$$f_{LZ}(T) = \sum_{e \in E} f_{LZ}(e).$$ (4.3)

Similarly, the number of bends associated with a tree $T(V, E)$ is the sum of bends of the edges in the tree.

$$bends(T) = \sum_{w \in E} bends(w).$$ (4.4)

Here, $bends(w)$ is the minimum number of bends required by a router to route the net $w = (a, b)^2$, i.e.

$$bends(w) = \begin{cases} 
1 & \text{if } r_a = r_b \text{ or } c_a = c_b \\
0 & \text{otherwise},
\end{cases}$$ (4.5)

where $r_a$ and $c_a$ denote the row and column coordinates of a pin (node) $a$. Let us define $F \subseteq E = \{e : f_{true}(e) > 0\}$ as the set of edges requiring a bend. Now the following bound on the number of bends that will be routed ($bends(n_{routed})$) is found.

**Theorem 4.4 (Bends bound).**

$$|F| = bends(T) \leq bends(n_{routed}).$$ (4.6)

**Proof.** Trivial. 

In order to achieve a reduced number of vias in the final layout the number of bends is reduced in this thesis. Clearly, there is a relation between via count ($vias(n)$) and bend count but the bend count is related to a part of the flow that is largely technology-independent whereas the final via count obviously depends on the implementing technology e.g. through the number of routing layers. The number of excess bends is the difference between the number of bends in the routing result and in the tree. Although a router has much freedom, under some relatively mild conditions the following bound is derived.

**Theorem 4.5 (Via bound).** Consider a technology with a set $L$ of uni-directional routing layers, alternating by direction. Let a net $n$ be broken up in a tree $T(V, E)$ where $E$ represents the set of wires. If layer assignment restricts each net to two neighboring layers$^3$, and $bends$ is an upper bound on the number of excess bends in a routed wire, then the following bound is valid.

$$bends(T) \leq vias(T) \leq |V| \cdot (|L| - 1) + \sum_{e \in E} bends(e) + \overline{bends}$$ (4.7)

$^2$Blockages are ignored here, but an extension is straightforward.

$^3$Such a pair with one layer for horizontal and one layer for vertical wiring is often referred to as a layer tier.
4.4 The BOI algorithm

The heuristic BOI algorithm is the basis of the steiner tree algorithm developed in this chapter. BOI appears to be one of the most widely used Steiner tree algorithms throughout the EDA industry. It combines the desirable properties of good quality of results and reasonable run times.

The BOI algorithm consists of greedy basic updates and is summarized in Fig. 4.3. This basic update is based on a node-edge combination \( (n, e) \) as illustrated by Fig. 4.4. Conceptually, the edge is bendet towards the node. A new node \( sp \)—a steiner point—is added to fix this bend. This does not add length to the tree. A new edge \( e_{new} \) connecting \( n \) and the steiner point is added to the tree, adding a length of \( L_1(e_{new}) \) to the tree. Adding an edge to a tree automatically creates a cycle. In the next step, the largest edge on this cycle, \( e_{l,new} \), is removed. The total gain of the update is the difference in length before and after the update. This is the difference in length between the removed and added edge since the bending did not add any length to the tree:

\[
\text{gain}(n, e) = (L_1(e_{l,new}) + L_e) - (L_1(e_{new}) + L_{e_1} + L_{e_2}) = L_1(e_{l,new}) - L_1(e_{new}).
\]  

4As a result of certain processing steps during manufacturing, metal wires become electrically charged. Discharge of these wires can destroy fragile gates. Since the amount of charge on the wires is proportional to its area, the area (length) of a wire must be limited. This is known as the antenna effect or plasma induced gate oxide damage.
BOI($T$)

1 $S \leftarrow \text{COLLECT-GAINS}(T)$
2 \textbf{while} $|S| > 0$
3 \quad \textbf{do for each gain } g$
4 \quad \quad e \leftarrow \text{BENDING-EDGE}(g)$
5 \quad \quad le \leftarrow \text{LARGEST-EDGE}(g)$
6 \quad \quad \textbf{if EXISTS}(e, T) \text{ and EXISTS}(le, T)$
7 \quad \quad \quad \textbf{then UPDATE}(T, g)$
8 \textbf{return } T$

\text{COLLECT-GAINS}(T)$

1 $S \leftarrow \emptyset$
2 \textbf{for each edge } $(u, v) \text{ in } E$
3 \quad \textbf{for each node } $n \text{ in } V \\setminus \{u, v\}$
4 \quad \quad \textbf{do } sp \leftarrow \text{STEINER-POINT}((u, v), n)$
5 \quad \quad \quad cost \leftarrow \text{MANHATTAN-DISTANCE}(sp, n)$
6 \quad \quad \quad le \leftarrow \text{LARGEST-EDGE}(e, n)$
7 \quad \quad \quad gain \leftarrow \text{MANHATTAN-LENGTH}(le) - cost$
8 \quad \quad \quad \textbf{if gain} > 0$
9 \quad \quad \quad \quad \textbf{then } S \leftarrow S \cup \text{MAKE-GAIN}(gain, sp, e, n, le)$
10 \textbf{return } \text{SORT}(S)$

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{boi_collect_gains.png}
\caption{The BOI and COLLECT-GAINS algorithms.}
\end{figure}

An edge or node may be part of more than one potential update with positive gain. Therefore, first all node-edge combinations are collected with their gains. These gains are sorted and then processed in \textit{batch mode}. The tree is greedily improved with the update with highest gain. There is a check ensuring only updates that have not been invalidated by previous updates are considered. The original algorithm contained a flaw, as also observed in [146]. In our implementation this is handled in a similar way as in that paper. The procedure is repeated until no positive gains are found anymore.

\section*{Time complexity of BOI}

The time complexity of our implementation of BOI is $O(|V|^3)$ because this is the time complexity of COLLECT-GAINS. LARGEST-EDGE is called for each of the $O(|E| \cdot |V|)$ edge-node combinations. It is basically a depth-first search of $O(|E|)$. Together, this yields a complexity of $O(|E| \cdot |V| \cdot |E|) = O(|V|^3)$.

The original paper [10] quotes a time complexity of $O(|V|^2)$ but it is not clear how this is achieved. According to [64] a widely used implementation “appears to have cubic rather than quadratic time complexity”.
4.5 BOI with precalculation

In this section a new way to speed up the BOI algorithm is presented. The calculations are organized such that the time complexity reduces to $O(|V|^2)$. This is achieved by using precalculation for the largest edge calculation.

The precalculation algorithm is based on depth-first search. During the search a set of nodes that has been reached and a set of nodes that has not been reached yet is maintained. Let us assume that for each combination of visited nodes the largest edge on the path between them is known. If the search now expands from a node $m$ to a node $n$ as illustrated by Fig. 4.5, the largest edge between $n$ and $a$ must either be $(m, n)$ or the previously found largest edge between $m$ and $a$, as proven in the following theorem.

**Theorem 4.7** (Largest edge property). Consider a tree $T(V, E)$, the edge $(m, n) \in E$, and a node $a$. Let node $m$ be on the path from $n$ to $a$. Then, the largest edge on the path between $n$ and $a$ is either the edge $(n, m)$, or the largest edge on the path between $m$ and $a$.

**Proof.** By contradiction. Let us assume that another edge is the largest edge between $n$ and $a$. Since this is not $(n, m)$ and not on the path between $m$ and $a$, it is on another path between $n$ and $a$. This implies that $T$ is not a tree, which is a contradiction.

Using the above theorem it is possible to precalculate the largest edges between any pair of nodes in the tree with a single depth-first search. During the search the largest edge between any combination of nodes is stored in a matrix $M$. Then, the largest edge between

![Figure 4.4: A BOI update.](image)

![Figure 4.5: The largest edge between $n$ and $a$ is either the largest edge between $m$ and $a$ or $(n, m)$.](image)
LARGEST-EDGE-PRECOMPUTE($T$)

▷ Initialize the DFS
1. $S ← \text{EMPTY-SET}()$ ▷ The set of visited nodes
2. Find a leaf node $n$ and its edge $e$
3. $S ← S \cup n$
4. return LARGEST-EDGE-DFS($n, e, S, \text{EMPTY-MATRIX}()$)

LARGEST-EDGE-DFS($n, e, S, M$)

1. $m$ is the node connected to $n$ through $e$
2. $S ← S \cup m$
3. for each node $o$ in $S$
4.    do $M[m, o] ← M[o, m] ← \text{MAX}(e, M[o, m])$
5. for each edge $f \neq e$ adjacent to $m$
6.    do $M ← \text{LARGEST-EDGE-DFS}(m, f, S, M)$
7. return $M$

Figure 4.6: The LARGEST-EDGE-PRECOMPUTE algorithm.

an edge $(m, n)$ and any node $a$ is easily found as the larger of the largest edge between $m$ and $a$ and the largest edge between $n$ and $a$. This procedure takes only constant time.

The precalculate algorithm is shown in Fig. 4.6. During the search a set of already visited nodes is maintained. When the search progresses from a node $m$ to $n$, the largest edge between these two nodes is automatically found. Next, the largest edge between any node from the visited set and $n$ is found by comparing the largest edge between $m$ and the node from the visited set with the new edge. The algorithm progresses until all nodes are visited.

Run time and memory complexity

The memory requirement for LARGEST-EDGE-PRECOMPUTE is $O(|V|^2)$, the size of $M$. The set of visited nodes has size $O(|V|)$, so $O(|V|)$ MAX operations are performed during each visit. Since each node is visited only once, this yields an overall time complexity of $O(|V|^2)$.

4.6 MST decomposition

Most steiner tree heuristics including BOI are based on MSTs. Many efficient MST algorithms exist. (See a good textbook such as [32] for an overview. More recent work can be found in [64] and [146], and the references therein.) Run time is obviously an important consideration, especially when the MST/RSMST algorithm is used in a congestion estimator. All MST algorithms yield trees with the same minimum length, but when multiple minimum length trees exist other considerations are also of importance. Firstly, some MSTs may be better for the BOI heuristic. Regardless of the MST it has as an input, BOI almost always yields an optimal steiner tree. The expected gain due to better MST selec-
Effect of freedom on BOI

The amount of freedom in the MST BOI works on will not have a large impact on the results of BOI in terms of tree length since BOI is known to be optimal for most cases. Nonetheless, it is possible to improve results based on the following observation.

Observation 4.8. MSTs with more freedom contain more edges with the opportunity to “bend” and are thus more friendly to the BOI algorithm.

Let us now introduce the terms minimum freedom MST and maximum freedom MST.

Definition 4.1 (Minimum and Maximum freedom MST). A tree \( T(V_N, E_{MST}) \) is called a minimum (maximum) freedom MST associated with a net \( N \) if it is a minimum spanning tree of the complete graph \( G_N(V_N, E_{CN}) \) associated with \( N \) and no other MST of \( G_N(V_N, E_{CN}) \) with less (more) true freedom exists.

Fig. 4.7 is an example taken from one of the benchmarks. On the left and right are a minimum and maximum freedom MST shown, respectively. The steiner tree produced with the maximum freedom MST is shorter than the steiner tree produced with the minimum freedom MST.

Freedom and global routing

One of the purposes of the steiner tree algorithm is to provide a global router with a set of wires that is to be routed. Steiner tree algorithms minimize tree length since wire length corresponds to average congestion. Wires generated with a maximum freedom MST based steiner tree algorithm have more detour-free realizations. Thus, the router has greater possibility to find paths through a congested region. On the other hand, it has been shown that trees with lower freedom are more likely to yield less bends.
In short, for designs that are on the edge of routability, maximum freedom MSTs are attractive because the larger freedom may increase the probability of routability. For easily routable designs, minimum freedom MSTs are attractive because less bends and vias are required.

### 4.6.1 Maximum and minimum freedom MSTs

The problems of finding maximum and minimum freedom MSTs are defined as follows.

**Problem 4.3** (MaxFMST). Consider a grid graph $G_g(V_g, E_g)$, a net $N$ with associated complete graph $C_N(V_N, E_{c,N})$ and a length $L(e)$ for each edge $e \in E_{c,N}$. Of all MSTs, find an MST $T(V_N, E_{MST})$ with maximum true freedom, i.e. $\sum_{e \in E_{MST}} f_{true}(e)$ is maximized.

**Problem 4.4** (MinFMST). Consider a grid graph $G_g(V_g, E_g)$, a net $N$ with associated complete graph $C_N(V_N, E_{c,N})$ and a length $L(e)$ for each edge $e \in E_{c,N}$. Of all MSTs, find an MST $T(V_N, E_{MST})$ with minimum true freedom, i.e. $\sum_{e \in E_{MST}} f_{true}(e)$ is minimized.

### Freedom, bends and vias in steiner trees

Since the steiner trees generated with BOI are based on MSTs the freedom in the steiner tree largely depends on the freedom in the MST it is based upon. In practice RSMT algorithms reduce the length compared to MST relatively little and only few wires are affected. In our experience this relatively small improvement sometimes makes a design routable. The reduction of wire length comes at the cost of reduction in freedom as stated in the following theorem.

**Theorem 4.9** (Freedom destruction). Consider a steiner tree $T'(V', E')$ that is generated from an MST $T(V, E)$ by the BOI algorithm. The total freedom in the tree and the average true freedom per wire is never increased by BOI, i.e.

$$f_{true}(T') \leq f_{true}(T),$$

and

$$\frac{f_{true}(T')}{|E'|} \leq \frac{f_{true}(T)}{|E|}.$$  \hspace{1cm} (4.10)

**Proof.** If $T'(V', E') = T(V, E)$ the theorem is evidently correct. Otherwise consider the basic update of BOI as illustrated by Fig. 4.4. Each of the newly created edges ($e_1$, $e_2$, and $e_{new}$) has a true freedom that is strict less than that of the bending edge ($e$). Each update increments the edge count in the tree with one and both the total and average freedom per edge decrease. Since BOI only consists of basic updates, BOI can never increase the average freedom.

As a result of freedom destruction the number of bends in a steiner tree generated with BOI may only decrease, as stated in the next theorem.

**Theorem 4.10** (Bends decrease). Consider a steiner tree $T'(V', E')$ that is generated from an MST $T(V, E)$ by the BOI algorithm. The number of bends associated with $T'$ is never increased by BOI, i.e.

$$bends(T') \leq bends(T).$$  \hspace{1cm} (4.11)
Proof. Similar to the proof of 4.9. If $T'(V', E') = T(V, E)$ the theorem is evidently correct. Otherwise consider the basic update of BOI. The bending edge is replaced by a set of three edges of which at most one has a bend. Accordingly, the number of bends in the tree can only decrease.

Although the number of bends typically decreases due to BOI, the same thing is not necessarily true for the number of vias. An important observation is that BOI updates always involve an edge that already has a bend. This bend is fixed at some location and becomes a steiner point. Ideally, the new connection to the steiner point does not require a new via, as is the case when the net is implemented in layer tier. In many cases the long disappearing edge required a via and the total number of vias that is expected to be necessary for the implementation of the tree decreases. In practice a reduction in vias is not always achieved due to a number of reasons. Pin access from the different routing layers requires a different number of vias, routers do not always stay within a layer tier and there may be a large number of congestion related bends and vias. Nonetheless, on average bend reduction means a reduction in via count.

### 4.6.2 Prim’s MST algorithm

There is a large number of efficient MST algorithms. The GSRC Bookshelf has compared Prim’s algorithm, the algorithm by Guibas and Stolfi[50] and a modification to Prim’s algorithm by Lou Scheffer. The conclusion is that although Prim’s algorithm has a higher time complexity $O(|E| + |V| \log |V|) = O(|V|^2)$ than Guibas-Stolfi ($O(|V| \log |V|)$), for nets with up to 100 terminals Prim’s algorithm is on average faster than the other two algorithms. For relatively small nets the overhead associated with these algorithms and their advanced data structures apparently outweighs the benefits. Since VLSI nets have far less pins than 100 on average, Prim’s algorithm has been chosen as the basis for Minimum Spanning Tree generation.

The implementation of Prim’s algorithm as used in this thesis is outlined in Fig. 4.8. The algorithm is well-known and well-explained in [32] so it will be discussed only briefly. The tree is grown starting from a random node. The algorithm greedily selects edges that are cheapest according to the COMPAR function. For the rectilinear case this function would compare $L_1$ edge lengths. These edges must not connect two nodes that are both already in the tree. An efficient heap data structure is used to find the shortest edge in the set of candidate edges. When a node is added to the tree its edges that connect to nodes not already in the tree are added to the heap.

### 4.6.3 Prim’s algorithm for MaxFMST and MinFMST

This thesis proposes a version of Prim’s algorithm that yields a MaxFMST using the concept of tiebreakers. Freedom is optimized as a secondary criterion. First, the algorithm looks at length. Only if the lengths of two edges are equal the algorithm looks at freedom. This results in exactly the same pseudo code for the algorithm. The difference is that the

---

5 An implementation of Prim’s algorithm using Fibonacci heaps yields a time complexity of $O(|E| + |V| \log |V|) = O(|V|^2)$ since we deal with complete graphs. A more traditional implementation using less advanced data structures yields a time complexity of $O(|E| \log |V|) = O(|V|^2 \log |V|)$. Details can be found in [32].

6 These results can be found at [http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST](http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST).
**PRIM-MST***(N, COMPARE)\n
1. \[E_{\text{MST}} \leftarrow \text{MAKE-EMPTY-SET}()\] The set of edges to be returned.
2. \[G \leftarrow \text{MAKE-COMPLETE-GRAPH}(N)\]
3. for each node \(n \in V[G]\) \n4. \[\text{do mark}[n] \leftarrow \text{FALSE} \] Marks whether the node is in the tree.
5. \[H \leftarrow \text{EMPTY-HEAP}(\text{COMPARE}) \] Compares edges based on the COMPARE metric.
6. Select a random node \(n\) \n7. \[\text{mark}[n] \leftarrow \text{TRUE}\]
8. for each edge \((m, n) \in \text{edges}[n]\) \n9. \[\text{do PUSH-HEAP}(H,(m, n))\]
10. while \(|E_{\text{MST}}| < |V[G]| - 1\) \n11. \[\text{do } (u, v) \leftarrow \text{POP-HEAP}(H)\]
12. if not \(\text{mark}[u]\) \n13. then \(\text{mark}[u] \leftarrow \text{TRUE}\)
14. \[E_{\text{MST}} \leftarrow E_{\text{MST}} \cup (u, v)\]
15. for each \((w, u) \in \text{edges}[u]\) \n16. \[\text{do if } \text{mark}[w] = \text{FALSE}\]
17. then \(\text{PUSH-HEAP}(H,(m, n))\)
18. return \(T(G[V], E_{\text{MST}})\)

**Figure 4.8:** The PRIM-MST algorithm.

The heap uses a different COMPARE function. An example is shown in Fig. 4.9. Rectilinear length is called the primary objective and true freedom the first tiebreaker or secondary objective. It is possible to use other criteria such as LZ freedom or minimum freedom as a first or additional tiebreaker.

The heap requires a total order on the set of edges. In COMPARE-TIEBREAKERS the relation is split in two separate total orders: \(\leq_{L1}\) for increasing \(L1\) edge lengths and \(\leq_{\text{true}}\) for increasing true freedom. Operator \(\geq_{\text{true}}\) can also be used as a tiebreaker and yields a sequence of decreasing true freedoms. Tiebreakers \(\leq_{\text{LZ}}\) and \(\geq_{\text{LZ}}\) are defined similarly.

Prim’s algorithm using a COMPARE function with \(\geq_{\text{true}}\) as the first tiebreaker yields a MaxFMST as proven in the following theorem.

**Theorem 4.11** (MaxFMST). Consider a connected graph \(G(V, E)\). Algorithm PRIM-MST using a COMPARE function with operator \(\geq_{\text{true}}\) as the first tiebreaker yields a Minimum Spanning Tree with the largest amount of true freedom (maxFMST), i.e. no MST with lower true freedom exists.

**Proof.** During the algorithm a sub-graph is grown, starting from a single node. Such a single node is a sub-graph of any MaxFMST. An edge is called safe if after adding it to such a sub-graph of any MaxFMST the new sub-graph is still a sub-graph of some MaxFMST. Evidently, an algorithm that only adds safe edges yields a MaxFMST.

An edge is a light edge crossing a cut if a) its length is no higher than any other edge crossing the cut, and b) no other edge of equal length crossing the cut has higher freedom.
**COMPARE-TIEBREAKERS**($e_0, e_1$)

1. if \( L_1(e_0) > L_1(e_1) \) then return \( \text{TRUE} \)
2. elseif \( L_1(e_0) < L_1(e_1) \) then return \( \text{FALSE} \)
3. else
   4. do if \( f_{\text{true}}(e_0) > f_{\text{true}}(e_1) \) then return \( \text{TRUE} \)
   5. elseif \( f_{\text{true}}(e_0) < f_{\text{true}}(e_1) \) then return \( \text{FALSE} \)

\[ \triangleright \text{Lengths are equal, use freedom as a tiebreaker} \]
6. else
7. do if \( f_{\text{true}}(e_0) > f_{\text{true}}(e_1) \) then return \( \text{TRUE} \)
8. elseif \( f_{\text{true}}(e_0) < f_{\text{true}}(e_1) \) then return \( \text{FALSE} \)

\[ \triangleright \text{The default value is arbitrary; other tiebreakers could be added} \]
9. return \( id(e_0) < id(e_1) \)

---

**Figure 4.9:** The COMPARE-TIEBREAKERS algorithm.

First we prove that given \( A \), a subset of \( E \) included in some MaxFMST \( T \) of \( G \), and \((S, V−S)\), any cut of \( G \) that respects \( A \), any light edge crossing the cut is safe. Then we prove that the MST algorithm with tiebreakers only adds light edges.

Let \((u, v)\) be a light edge that crosses the cut and is not included in \( T \) (otherwise we are done). There exists at least one other edge \((x, y)\) \(\neq (u, v)\) in \( T \) that crosses the cut. Since the cut respects \( A \), \((x, y)\) \(\notin A \). Now create a new tree \( T' = T − (x, y) ∪ (u, v) \). Since \((u, v)\) is a light edge crossing the cut and \((x, y)\) also crosses this cut, \( L_1((u, v)) \leq L_1((x, y)) \). Therefore, \( L_1(T') = L_1(T) − L_1((x, y)) + L_1((u, v)) \leq L_1(T) \). Since \( T \) is an MST, \( L_1(T) \leq L_1(T') \), so \( L_1(T) = L_1(T') \). Equivalently, \( f_{\text{true}}((u, v)) \geq f_{\text{true}}((x, y)) \). Therefore, \( f_{\text{true}}(T') = f_{\text{true}}(T) − f_{\text{true}}((x, y)) + f_{\text{true}}((u, v)) \geq f_{\text{true}}(T) \). Since \( T \) was a MaxFMST, \( f_{\text{true}}(T) \leq f_{\text{true}}(T') \), so \( f_{\text{true}}(T) = f_{\text{true}}(T') \). Obviously, \( A \subseteq T' \), and \( A \subseteq T \). Therefore, \( A ∪ (u, v) \subseteq T' \). Consequently, \( T' \) is a MaxFMST, and \((u, v)\) is part of it.

During the MST algorithm with tiebreakers a cut \((S, V−S)\) is maintained, where \( S \) represents the selected nodes. The heap contains only edges crossing the cut or between nodes in \( S \). Because of the compare function with tiebreakers the edge on top of the heap is a light edge in the former case. If the edge on top of the heap is between nodes in \( S \) it is discarded. Consequently, only light edges are added and the algorithm yields a MaxFMST.

Using \( \leq_{\text{true}} \) instead of \( \geq_{\text{true}} \) as the first tiebreaker yields a MinFMST.

**Theorem 4.12** (MinFMST). Consider a connected graph \( G(V, E) \). Algorithm PRIM-MST using a COMPARE function with operator \( \leq_{\text{true}} \) as the first tiebreaker yields a Minimum Spanning Tree with the least amount of true freedom (minFMST), i.e. no MST with higher true freedom exists.

**Proof.** Similar to the proof of MaxFMST.

Interestingly, there is no conflict between true freedom and LZ freedom, as illustrated by the following theorem.
Theorem 4.13 (MaxFMST is MaxLZFMST). A MaxFMST generated with Prim’s algorithm with $f_{true}$ as a first tiebreaker is also an MST with maximum LZ freedom.

Proof. During Prim’s algorithm, of all shortest edges connecting the set of visited nodes and the set of unvisited nodes, the one with maximum true freedom is chosen. This edge also has the highest LZ freedom, hence effectively Prim’s algorithm with $\geq_{LZ}$ as the first tiebreaker is executed. Then, the theorem is true by the same argument as the proof of Theorem 4.11. \qed

Equivalently it is found that

Theorem 4.14 (MinFMST is MinLZFMST). A MinFMST generated with Prim’s algorithm with $\leq_{LZ}$ as a first tiebreaker is also an MST with minimum LZ freedom.

Proof. Similar to the proof above. \qed

Usually terms such as minimum and maximum freedom will be used for tiebreakers throughout this thesis, indicating the intention of their use. In our implementation the tiebreakers are set on the command line. In addition to the already mentioned tiebreakers a random tiebreaker is available.

4.7 Extension to spanning graph

The motivation for using Prim’s algorithm as the basis for our MST algorithm is that it has been shown to be suitable for typical VLSI nets. However, it is also possible to design a MaxFMST or MinFMST algorithm of $O(|V| \log |V|)$ based on Zhou’s spanning graph\[147\] which can be seen as an improvement on our implementation of Prim.

Typical MST algorithms have complexity $O(|E| \log |V|)$. Our algorithm works on a complete graph for each net. Since $|E| = O(|V|^2)$ this yields a time complexity of $O(|V|^2 \log |V|)$ for our MST algorithm. Zhou improves upon this time complexity by replacing the complete graph with a spanning graph $G_{sp}(V,E_{sp})$ with $|E_{sp}| = O(|V|)$. A spanning graph is defined to be a graph containing at least one MST (both complete graphs and MSTs are spanning graphs). Zhou’s spanning graph can be constructed in $O(|V| \log |V|)$ time, yielding an $O(|V| \log |V|)$ MST algorithm.

4.7.1 Octal partitions

The basic observation for efficient spanning graph construction is the fact that each node only needs to have an edge to the closest node in each of its octal partitions, as illustrated by Fig. 4.10. Hence, each node has at most eight edges, yielding $|E_{sp}| \leq 8 \cdot |V| = O(|V|)$.

4.7.2 Sweep line algorithm

The nearest edges in each partition is found using a sweep line algorithm. Similar to [147] we focus on the $R1$ partition since the other partitions are similar. The algorithm is outlined in fig. 4.11.

The algorithm maintains an initially empty active set $A$ with nodes for which no nearest node in their $R1$ partitions have been found yet. When such a nearest node is found an
edge is added to $E_{sp}$ and the node is removed from the active set. The nodes are processed in non-decreasing $x+y$ (the sweep line goes from bottom-left to top-right). First, it is checked whether a new node $v$ is in the $R1$ partition of any node in $A$, i.e. it is checked which nodes in $A$ are in the $R5$ partition of $v$. For these nodes, the new node is guaranteed to be a nearest node for the node in $A$ because of the direction of the sweep line.

Algorithm GEt-R5-Nodes exploits the fact that $A$ contains only active nodes. As a result no $v \in A$ can be in the $R1$ partition of another node $w \in A$. Therefore, sorting $A$ by increasing $x$ coordinate implies a non-decreasing order on $x-y$. The set of relevant nodes can be found by finding the first node with $x \leq x_v$, where $x_v$ is the $x$ coordinate of $v$, and then proceed in decreasing order of $x$ until $x-y \geq x_v-y_v$. A simple binary search tree with $O(\log|V|)$ insertion, deletion and query time suffices to find an overall time complexity of $O(|V|\log|V|)$.

SWEEP-LINE-R1(V)

1. Sort $V$ in non-decreasing $x+y$
2. $A \leftarrow$ EMPTY-SET
3. $E_{sp} \leftarrow$ EMPTY-SET
4. for each $v \in V$ in sorted order
5. \hspace{1em} do $S \leftarrow$ GET-R5-Nodes($A, v$)
6. \hspace{2em} for each $w \in S$
7. \hspace{3em} do $E_{sp} \leftarrow E_{sp} \cup (v, w)$
8. \hspace{2em} $A \leftarrow A \setminus w$
9. return $E_{sp}$

4.7.3 Spanning graph for MaxFMST and MinFMST

Let us define a maximum (minimum) freedom spanning graph as a spanning graph containing a maximum (minimum) freedom MST. We will focus on maximum freedom MSTs because minimum freedom MSTs are found similarly.
By ordering the nodes on the sweep line, a maximum or minimum freedom spanning graph can be created.

In some cases Zhou’s algorithm for creating a spanning graph may not yield a maximum freedom spanning graph as illustrated by Fig. 4.12. Node \( w \) has four nodes \( v_0 - v_3 \) at equal distance in its \( R_1 \) partition. In Zhou’s algorithm it is arbitrary to which node \( w \) is connected. However, we observe that edge \((w, v_0)\) has higher true freedom than edge \((w, v_3)\). In fact, for the \( R_1 \) partition the sweep line node with the lowest \( y \) coordinate will always yield the edge with highest freedom.

For \( R_1 \) it is easily seen that for maximum spanning graphs, the ties during sweep line sorting should always be broken with lowest \( y \) first. For the other partitions similar rules are easily found. Now we find the following theorem.

**Theorem 4.15** (Maximum freedom spanning graph). Zhou’s spanning graph algorithm as outlined in Fig. 4.11, with the appropriate tiebreakers during sweep line creation yields a maximum freedom spanning graph in \( O(|V| \log |V|) \).

**Proof.** The algorithm has no higher complexity than the original algorithm since node comparison is still a constant time operation. We will now prove that the theorem is correct for the \( R_1 \) partition. For the other partitions similar arguments hold.

By contradiction. Assume that some MaxFMST contains an edge \((v, w)\) that did not exist in the spanning graph returned by our algorithm, and where \( v \) is in \( R_1 \) of \( w \). Since \((v, w)\) was not in the spanning graph, there must be some other node \( v' \) such that \( L_1(w, v) = L_1(w, v') \), \( v' \) is also in \( R_1 \) of \( w \), and \( f_{true}(w, v') > f_{true}(w, v) \). Now add an edge \((w, v')\), and remove the edge \((w, v)\). The tree is still an MST and the freedom in the tree has increased. This is a contradiction with the assumption that we were dealing with a MaxFMST. \( \square \)

### 4.8 Experimental results

For the experiments the well-known ispd98/ibm benchmark suite for global routing\[72\] is used. Until recently this was the most commonly used benchmark suite for global routing\[7\]. It is based on the ispd98/ibm placement benchmarks that have been placed with the

\[7\text{After the completion of this work the ISPD global routing benchmark suite has become available[91].}\]
4.8 Experimental results

The most important characteristics are summarized in Table 4.1. The benchmark suite is already a few years old so the design sizes are a bit smaller than state-of-the-art designs. It is generally true that net characteristics such as the average number of pins per net have not changed significantly, and for MST and RSMT algorithms this is obviously more important than design size e.g., in terms of number of standard cells or number of nets. Note that the largest design has over 200,000 nets and standard cells which is still reasonably large by today’s standards. The origin of the benchmarks are real VLSI designs and these benchmarks are therefore more representative than randomly generated nets as often used in papers on steiner and minimum spanning trees.

<table>
<thead>
<tr>
<th>design</th>
<th>#cells</th>
<th>#pins</th>
<th>#nets</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>12506</td>
<td>246</td>
<td>11507</td>
</tr>
<tr>
<td>ibm02</td>
<td>19342</td>
<td>259</td>
<td>18429</td>
</tr>
<tr>
<td>ibm03</td>
<td>22853</td>
<td>283</td>
<td>21621</td>
</tr>
<tr>
<td>ibm04</td>
<td>27220</td>
<td>287</td>
<td>26163</td>
</tr>
<tr>
<td>ibm05</td>
<td>28146</td>
<td>1201</td>
<td>27777</td>
</tr>
<tr>
<td>ibm06</td>
<td>32332</td>
<td>166</td>
<td>33354</td>
</tr>
<tr>
<td>ibm07</td>
<td>45639</td>
<td>287</td>
<td>44394</td>
</tr>
<tr>
<td>ibm08</td>
<td>51023</td>
<td>286</td>
<td>47944</td>
</tr>
<tr>
<td>ibm09</td>
<td>53110</td>
<td>285</td>
<td>50393</td>
</tr>
<tr>
<td>ibm10</td>
<td>68685</td>
<td>744</td>
<td>64227</td>
</tr>
</tbody>
</table>

4.8.1 BOI with precalculation

Table 4.2 shows how precalculation can be used to speed up BOI. On average BOI with precalculation is 9 times faster than BOI without precalculation. The numbers are heavily biased by the excellent results on benchmarks ibm02 and ibm08.

<table>
<thead>
<tr>
<th>design</th>
<th>cpu BOI [s]</th>
<th>speed up</th>
<th>design</th>
<th>cpu BOI [s]</th>
<th>speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w/o pcalc</td>
<td>/w pcalc</td>
<td></td>
<td>w/o pcalc</td>
<td>/w pcalc</td>
</tr>
<tr>
<td>ibm01</td>
<td>0.25</td>
<td>0.07</td>
<td>3.6X</td>
<td>ibm06</td>
<td>0.74</td>
</tr>
<tr>
<td>ibm02</td>
<td>7.82</td>
<td>0.18</td>
<td>43.4X</td>
<td>ibm07</td>
<td>0.96</td>
</tr>
<tr>
<td>ibm03</td>
<td>0.55</td>
<td>0.12</td>
<td>4.6X</td>
<td>ibm08</td>
<td>7.50</td>
</tr>
<tr>
<td>ibm04</td>
<td>0.70</td>
<td>0.14</td>
<td>5.0X</td>
<td>ibm09</td>
<td>1.36</td>
</tr>
<tr>
<td>ibm05</td>
<td>1.32</td>
<td>0.32</td>
<td>4.1X</td>
<td>ibm10</td>
<td>1.84</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td></td>
<td>9X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The theory presented previously indicates that precalculation is especially beneficial for large nets because of the super-linear time complexity. In practice, the run time required by a benchmark is mainly determined by the run time of the few largest nets. The net degree distributions of the benchmarks are shown in Table 4.3. It can be seen that all designs have comparable distributions with the majority of nets having only two or three pins. There is however a large difference in size of the largest net. Benchmarks ibm02 and ibm08 have by far the largest nets. As expected, this results in large run times for BOI without precalculation and also in large speedups.

Nets that reside entirely in a single tile after placement have been removed from the global routing benchmarks.
4.8.2 Effect of freedom on BOI

It is assumed that in the few cases that the original BOI algorithm does not yield optimal steiner trees MSTs with more freedom may result in better lengths. MSTs with less freedom on the other hand should result in lower number of bends.

Effect on wire length

The effect of the amount of freedom in the input MSTs on the result of BOI is shown in Table 4.4.

<table>
<thead>
<tr>
<th>design</th>
<th>BOI length</th>
<th>BOI bends</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MaxFMST</td>
<td>MinFMST</td>
</tr>
<tr>
<td>ibm01</td>
<td>60207</td>
<td>60219</td>
</tr>
<tr>
<td>ibm02</td>
<td>166099</td>
<td>166138</td>
</tr>
<tr>
<td>ibm03</td>
<td>145837</td>
<td>145880</td>
</tr>
<tr>
<td>ibm04</td>
<td>162879</td>
<td>162908</td>
</tr>
<tr>
<td>ibm05</td>
<td>410281</td>
<td>410312</td>
</tr>
<tr>
<td>ibm06</td>
<td>276296</td>
<td>276337</td>
</tr>
<tr>
<td>ibm07</td>
<td>363888</td>
<td>363954</td>
</tr>
<tr>
<td>ibm08</td>
<td>403040</td>
<td>403167</td>
</tr>
<tr>
<td>ibm09</td>
<td>411769</td>
<td>411828</td>
</tr>
<tr>
<td>ibm10</td>
<td>575048</td>
<td>575184</td>
</tr>
<tr>
<td>avg</td>
<td>0.20</td>
<td></td>
</tr>
</tbody>
</table>

For all benchmarks, BOI based on MaxFMSTs does indeed yield lower total wire length. The reductions however are only marginal. Again, there seems to be a weak correlation with maximum net degree, i.e. using MaxFMSTs is more effective on larger nets. This is
4.8 Experimental results

explained by the fact that on average large nets are more likely to have different MaxFMSTs and MinFMSTs. It is also more likely that optimal RSMT length and MST length are different for larger nets. For the individual nets a reduction in length may be beneficial but from the perspective of congestion the effect can be ignored.

Effect on bends

Table 4.4 shows the effect of freedom in the input MST on the number of bends in the steiner tree produced by BOI. On average, a reduction of 9.3% is obtained by using MinFMSTs. There appears to be little correlation with e.g. net degree.

4.8.3 Effect of freedom on routing result

The purpose of steiner trees is to break up nets such that global routing algorithms based on two-pin nets can be used. In this section an in-house global router is used for experimentation. This router minimizes overflow aggressively but contrary to many other routers it can also minimize bends. According to our assumptions MaxFMSTs should yield higher routability and MinFMSTs should yield less bends and vias. Table 4.5 shows the effect of freedom on global routing results.

Effect on overflow

The results on overflow indicate that the amount of freedom does not influence the routing results greatly. In some cases the results are slightly better and in some other cases the results are slightly worse. This kind of somewhat random behavior can also be observed when other parameters of the routing algorithm are perturbed. It appears that the results are better on the designs with more overflow. It is possible to create overflow by reducing the routing capacity in order to get better statistical information. Such a design is definitely unroutable and the results are therefore of little practical value. When MaxFMST is used the congestion distribution is improved a bit (not shown).

<table>
<thead>
<tr>
<th>design</th>
<th>overflow</th>
<th>impr [%]</th>
<th>bends</th>
<th>impr [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MaxFMST</td>
<td>MinFMST</td>
<td></td>
<td>MaxFMST</td>
</tr>
<tr>
<td>ibm01</td>
<td>40</td>
<td>59</td>
<td>19</td>
<td>9316</td>
</tr>
<tr>
<td>ibm02</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>17042</td>
</tr>
<tr>
<td>ibm03</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13792</td>
</tr>
<tr>
<td>ibm04</td>
<td>93</td>
<td>106</td>
<td>13</td>
<td>19453</td>
</tr>
<tr>
<td>ibm05</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20376</td>
</tr>
<tr>
<td>ibm06</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>25369</td>
</tr>
<tr>
<td>ibm07</td>
<td>10</td>
<td>2</td>
<td>-8</td>
<td>31690</td>
</tr>
<tr>
<td>ibm08</td>
<td>20</td>
<td>17</td>
<td>-3</td>
<td>36238</td>
</tr>
<tr>
<td>ibm09</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>39200</td>
</tr>
<tr>
<td>ibm10</td>
<td>4</td>
<td>3</td>
<td>-1</td>
<td>46371</td>
</tr>
<tr>
<td>avg</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5: The effect of freedom on global routing results
The above observations are contradictory to the results of [11]. In that work a very different router is used. That router used only considers a very limited number of routes per wire. If this number is reduced due to a different steiner topology this impacts the overflow directly. In contrast, our router considers many alternative routes per wire. If due to a steiner topology with less freedom this number is reduced this does not impact the final result directly since alternatives are likely to be available. Further, some of the benchmarks used in [11] are very small in comparison to our benchmarks. Approaches similar to ours are often used and our conclusions likely apply to many other routers.

Effect on bends

The results on bends are impressive. On average, the number of bends is reduced by 6.2% after routing due to the improved steiner topology. Note that the global router already optimizes the number of bends aggressively. Such a reduction in bends will result in a significant reduction in number of vias as well, although the percentage will be lower due to vias associated with pins.

The reduction in bend count after routing is less than the reduction in bend count in the steiner tree. On average, the number of excess bends \((\text{bends}_{\text{routed}}(T) - \text{bends}(T))\) is about the same, regardless of the freedom. These excess bends are the result of congestion. Congestion causes detours (and detours directly lead to excess bends) and additionally, bends are typically preferred over detour. The improvement in absolute numbers is about the same but due to the larger number of bends the corresponding percentage is lower. Therefore, the reduction in vias can be estimated based on the number of bends before global routing.

4.9 Summary and conclusions

In this chapter several improvements over the BOI heuristic for the Rectilinear Steiner Minimum Tree Problem are presented. Precalculation organizes the required calculations in a smarter way than in the original algorithm, yielding significant speed-up. One of the applications of steiner tree algorithms is to break up nets into two-pin wires that global routers algorithms can use. The tradeoff between routability and minimum bend count due to steiner tree topology has been explored. The BOI steiner tree algorithm is based on minimum spanning trees. The characteristics of the steiner tree therefore largely depend on the characteristics of the input MST. In this chapter MST algorithms that guarantee minimum and maximum freedom and bends are presented.

Although previous work indicates that routability is significantly improved when extra freedom is present in the design this is not really confirmed by our experiments. The traditional overflow metric is only marginally improved when BOI uses maximum freedom MSTs. Additionally, the congestion distribution is improved. The important metric bend count after routing is reduced by 6.2% on average when the MST with minimum freedom is used. This indicates a tradeoff needs to be made. Since in our experiments routability in terms of overflow is not really impaired when using minimum freedom MSTs, the main conclusion of this chapter is that minimum freedom trees should be generated as input for global routers.
Chapter 5

Probabilistic congestion estimation

Congestion is such a difficult problem for physical design because routing is the last implementation step. Design steps such as floorplanning, synthesis and placement have at best a coarse awareness of expected routing problems. In practice therefore iteration is used. Global routing results are fed back to upstream algorithms such as placers enabling a perturbed formulation taking congestion problems into account.

In recent years, algorithms that can deal with congestion up to the level of logic synthesis have been invented. Such algorithms need to explore large solution spaces and the encountered (potential) solutions may be very different from each other. An approach based on iteration between steps of a traditional flow is not feasible. Congestion-aware algorithms need to evaluate many alternatives and using traditional global routing this often is too time consuming.

In this chapter a probabilistic congestion estimator is presented. It can be used either by designers when they are performing interactive tasks, but given its speed it can also be incorporated under the hood in congestion-aware algorithms. The presented approach improves on the classic work by Lou et al. by incorporating observed router behavior.

5.1 Objectives for congestion estimation

Congestion estimators feed-forward congestion information to upstream tools. The objective of congestion estimation is a combination of speed and accuracy. Even though different tradeoffs are possible, in this thesis a congestion estimator must be at least an order of magnitude faster than global routing in order to be useful (also refer to Problem 3.1). Accurate prediction of the locations of congestion hotspots is more important than the absolute values since upstream algorithms using congestion maps try to spread congestion.

In this chapter congestion maps will be visually inspected. Additionally, the quality of the developed estimator will be verified by comparing the estimation results of our estimator against the predictions of an industrial global router. Error maps will be created and allow for visual correlation of errors in congestion maps with features in the final congestion map.
Finally, we observe that in a successful congestion-aware flow congestion estimates are necessarily off. Congestion estimates serve as a warning that is used to avoid the predicted problems. Congestion estimates can be characterized as self-unfulfilling prophecies.

5.2 Previous work

Congestion is a fundamental problem in routing. The nature of congestion has changed over the years with routing style. In early technologies there were only relatively few routing layers. Over the cell routing was not available and usually chips were wired in channels. This impacted the way people looked at routability. For instance, in the early 1980s El Gamal used probabilistic methods to estimate the routing area based on the analysis of routing channels[41, 42].

In [24], Cheng proposes a method based on average steiner trees. Conceptually, it is possible to calculate an average wiring distribution map by generating many random pin placements for a net of given pin count. In the proposed method such distributions are characterized by a net weight that essentially represents the relative length of an average steiner tree with this pin count. A net is evenly distributed over its bounding box, scaled by the net weight. Since the bounding box may also cover less or inaccessible regions of the chips e.g. because of blockages, methods to split the bounding box in sub-bounding boxes are developed.

Kusnadi and Carothers’ method[77] for measuring routability is based on calculating the number of unblocked paths for two-pin nets, similar to the true freedom as presented in this thesis. This method attempts to estimate routability directly and cannot be used for congestion estimation.

Wang and Sarrafzadeh present in [132] a method based on the bounding box of a net. It is found that it correlates not very well with actual global routing. A more accurate method based on incremental global routing is developed. This technique is used within an annealing-based placer with congestion optimization.

In [142], Yang and Sarrafzadeh use Rent’s rule in combination with the assumption of uniform cut net sizes to estimate the maximum routing demand. Unfortunately, the method is only accurate for about half the benchmarks and extracting the Rent parameters is computationally expensive.

The work in this thesis is based on the work by Lou et al.[84] and this work is discussed extensively in Section 5.4. The method is based on a detour-free routing model and was extended in [25] for the case with bounded detour. Analytical formulas to also take detours into account are presented and used to improve the estimation quality.

Kahng and Xu present in [69] a method that is an extension to the work of Lou and in some respects similar to the methods proposed in this thesis. Based on real designs, bend distributions are modeled in a congestion estimator. Detours are predicted based on a first pass. In subsequent passes detours are added to the nets in the most congested regions. The results are good but the algorithm appears to be relatively slow. This may be the result of using multiple passes.

Another method that uses multiple passes to improve congestion estimates is presented in [110] by Sham and Young. The first pass is very simple and based on uniform distribution over the net boxes. This yields a non-uniform distribution over the chip as a whole and this non-uniformity is used to re-distribute the congestion within the net boxes,
i.e. move congestion from dense areas to sparse areas. In the third and final phase, a similar but more extreme measure is taken to move congestion from dense to sparse areas, again within the net boxes.

### 5.3 Preliminaries

The congestion estimation methods in this chapter are based on the tile model presented in Section 3.1.1. The placement area is divided in rectangular tiles\(^1\) by a grid as illustrated by Fig. 5.1. Tile size reflects the tradeoff between accuracy and run times that has been made and in this thesis the same size as for global routing is used. In practice, tiles are more or less square and the height is equal to the height of a cell row as shown in Fig. 5.2. In current technologies this height amounts to a capacity of 8 to 12 routing tracks per layer, also depending on the exact routing layer since design rules for lower metal layers are typically different from design rules for higher metal layers. Similar models are also used e.g. in [24, 142, 69, 110].

The horizontal (vertical) usage of a tile is the number of occupied horizontal (vertical) tracks. This may be a fraction, when for instance a connection ends at a pin in the tile. \(H_{\text{tile}}\) and \(W_{\text{tile}}\) are the height and width of a tile. The distance between the left, right, bottom and top border of a tile and a pin \(p\) in that tile is denoted by \(l_p\), \(r_p\), \(b_p\), and \(t_p\) respectively. A pin \(p\) is a point with coordinates \(x_p\) and \(y_p\). A net \(n\) may span a number of tiles. The width and height of the spanning rectangle called net box are \(w_n\) and \(h_n\), and are expressed in number of tiles. A net \(n\) consisting of pins \(p\) and \(q\) as shown in Fig. 5.1 has \(w_n = 4\) and \(h_n = 3\).

Similar to most other probabilistic methods the probabilistic method as presented in this thesis is based on the analysis of two-pin nets. Since the method aims at mimicking router behavior, multi-pin nets are treated the same as in most global routers: they are broken up into two-pin nets with with Minimum Spanning Tree (MST) algorithms\(^2\) or Rectilinear Steiner Minimum Tree (RSMT) algorithms\(^3\) (see Chapter 4 for a discussion on MSTs and RSMTs). The resulting two-pin nets are referred to as wires. Similar to a net, a wire \(w\) has a box associated with it: the wire box with width \(w_w\) and height \(h_w\).

---

1. Other names for tiles include buckets, bins, and GCells.
2. This is a simplification. In reality, pins of standard cells may be (a set of) rectangles to which a wire has to connect.
Probabilistic congestion estimation

Figure 5.2: The height of a tile is typically equal to the height of a standard cell. There is space for about eight to twelve routes per layer.

In Fig. 5.3 a number of different wire shapes are shown. Flat wires and short wires are in probabilistic congestion estimation typically treated as special cases: they are supposed to be routed without any bends. Their wire boxes reside fully in a single row or column, or in a single tile, respectively. Wires with wire boxes that span both multiple columns and rows have a number of detour-free realizations, for example an L-shape with a single bend or a Z-shape with two bends. Realizations with more than two bends are referred to as multi-bend wires.

![Different kinds of wires and wire shapes.](image)

Figure 5.3: Different kinds of wires and wire shapes.

$U^w_{\text{hor}} (r, c)$ and $U^w_{\text{ver}} (r, c)$ are the horizontal and vertical probabilistic usages due to a wire $w$ in the tile with coordinates $(r, c)$, with $r$ the coordinate in vertical direction (row). The horizontal and vertical maximum capacities of a tile are defined as

$$C^\text{max}_{\text{hor}} (r, c) = H_{\text{tile}} \cdot \sum_{l_{\text{hor}}} 1 / p_{l_{\text{hor}}}
$$

3Because congestion maps are essentially matrices matrix notation is adopted here.
\begin{equation}
C_{\text{ver}}(r, c) = W_{\text{tile}} \cdot \sum_{L_{\text{ver}}} 1/p_{L_{\text{ver}}},
\end{equation}
\begin{equation}
(5.2)
\end{equation}

where $p_l$ denotes the minimum pitch at layer $l$, and $L_{\text{hor}}$ and $L_{\text{ver}}$ are the sets of layers with preferred horizontal and vertical directions, respectively. As explained in Chapter 3.1.1, not all of this capacity may be available and the capacities as modeled for the congestion estimation algorithm are denoted by $C_{\text{hor}}(r, c)$ and $C_{\text{ver}}(r, c)$, respectively. The (probabilistic) horizontal and vertical congestion of a tile is defined to be the ratio between (probabilistic) usage and capacity:

\begin{equation}
C_{\text{hor}}(r, c) = \frac{U_{\text{hor}}(r, c)}{C_{\text{hor}}(r, c)}
\end{equation}
\begin{equation}
C_{\text{ver}}(r, c) = \frac{U_{\text{ver}}(r, c)}{C_{\text{ver}}(r, c)}.
\end{equation}

In order to simplify the discussion, the simple terms usage and congestion will be used to refer to the maximum of horizontal and vertical usage and congestion, respectively.

\begin{equation}
U(r, c) = \max(U_{\text{hor}}(r, c), U_{\text{ver}}(r, c))
\end{equation}
\begin{equation}
C(r, c) = \max(C_{\text{hor}}(r, c), C_{\text{ver}}(r, c)).
\end{equation}

A tile is said to be over-congested in the horizontal (vertical) direction if the (probabilistic) usage in that direction exceeds the capacity:

\begin{equation}
U(i, j) > C(i, j),
\end{equation}

or equivalently, if the congestion-level is greater than one

\begin{equation}
C(i, j) > 1.
\end{equation}

\section*{5.4 Lou’s method}

In [84] Lou et al. develop a congestion estimation approach based on a tile model as described above. Essentially, the routing engine is mimicked with probabilities. A number of paths is considered for each wire, and the usage of the wire is conceptually spread over these paths. The method is based on the following assumptions.

1. All wires are routed in the shortest possible length.
2. All wires make at most one change of direction per tile.
3. Wires do not change direction in the tiles where the pins reside.

The first assumption ensures detour-freeness unless there are blockages. The second assumption models that probabilistically, a wire is routed through the middle of a tile. Finally, the third assumption affects how exact pin positions are taken into account (as shown below).

The second and third assumption reflect global routing practice. The router presented in Chapter 7 but also industrial global routers such as the Magma global router[88] have
these properties. They are a direct consequence of the way global routing models routing resources in a routing graph. Wire length minimization is traditionally the main objective for global routers and in practice routers are able to route most wires without detours, justifying the first assumption. When global routers deviate from detour-freeness this is the result of congestion. In Lou’s congestion estimation method this will be shown as over-congestion. Since detours correspond to slower wires, hotspots in Lou’s estimates may indicate problems even when an overflow-free routing solution exists.

Lou’s method is based on the analysis of two-pin nets and multi-pin nets are broken up with Minimum Spanning Tree (MST) algorithms or Rectilinear Steiner Minimum Tree (RSMT) algorithms as discussed in Chapter 4. MST algorithms are less accurate than RSMT algorithms but faster. Therefore it is advised that in early stages of physical design when design changes are likely the fast MST algorithm is used. In the final stages a more accurate RSMT algorithm should be employed.

**Probabilistic usage calculation**

For the sake of simplicity, pins are initially on the crossing of tile boundaries in the following analysis. Adaptations for the general case with pins at random positions are presented later. For Fig. 5.1 the simplification means that \( r_p = l_q = W_{tile} \), and \( b_p = t_q = H_{tile} \). \( F(m, n) \) is defined as the total number of possible ways to optimally route a wire \( w \) with an \( m \times n \) wire box. This is the true freedom as defined in Section 3.1.3. \( F(m, n) \) has a number of properties:

\[
F(m, 1) = F(1, n) = 1 \tag{5.10}
\]

\[
F(m, n) = F(n, m) \tag{5.11}
\]

\[
F(m, n) = F(m-1, n) + F(m, n-1) \tag{5.12}
\]

From the last property it follows that \( F(m, n) \) can be calculated in a recursive manner. The horizontal and vertical probabilistic usage matrices \( P_{hor}^{(m, n)} \) and \( P_{ver}^{(m, n)} \) are defined as

\[
P_{hor}^{(m, n)} = \begin{bmatrix}
P_{hor}^{(m, 1)} & \cdots & P_{hor}^{(m, n)} \\
\vdots & \ddots & \vdots \\
P_{hor}^{(1, 1)} & \cdots & P_{hor}^{(1, n)}
\end{bmatrix} \tag{5.13}
\]

and

\[
P_{ver}^{(m, n)} = \begin{bmatrix}
P_{ver}^{(m, 1)} & \cdots & P_{ver}^{(m, n)} \\
\vdots & \ddots & \vdots \\
P_{ver}^{(1, 1)} & \cdots & P_{ver}^{(1, n)}
\end{bmatrix} \tag{5.14}
\]

with \( P_{hor}^{(i, j)} \) and \( P_{ver}^{(i, j)} \) the probabilistic usages of the tile corresponding with entry \((i, j)\) of the matrices. The \( P \) matrices have the following properties.

\[
P_{hor|ver}^{(i, j)} = P_{hor|ver}^{(m-i+1, n-j+1)} \tag{5.15}
\]

\[
\sum_{j=1}^{n} P_{ver}^{(i, j)} = 1 \forall i \tag{5.16}
\]

\(^4\)True freedoms are binomial coefficients and properties of those are found in the literature on binomial coefficients and Pascal’s triangle. In this section we merely present what was considered important by Lou et al.
5.4 Lou’s method

\[
\sum_{i=1}^{m} P_{\text{hor}}(i, j) = 1 \forall j.
\]  

(5.17)

The first property stems from the fact that both pins are considered equivalent, e.g. there is no distinction made between source and sink pins. This results in rotational symmetry. Consequently, only the lower triangular part of the usage matrices needs to be calculated.

The relation between \(F(m, n)\) and the lower triangular part of the usage matrices is as follows:

\[
P_{\text{hor}}(i, j) = \begin{cases} 
F(m, n - 1) & \text{if } i = 1 \land j = 1 \\
1 & \text{if } i = 1 \land j = n \\
F(m - i + 1, n - 1) & \text{if } 1 < i < n \land j = 1 \\
\frac{F(i, j) \cdot F(m-i+1, n-j) + F(i-1, j) \cdot F(m-i+1, n-j+1)}{2} & \text{otherwise},
\end{cases}
\]

(5.18)

and

\[
P_{\text{ver}}(i, j) = \begin{cases} 
F(m - 1, n) & \text{if } i = 1 \land j = 1 \\
1 & \text{if } i = 1 \land j = n \\
F(m - 1, n - j + 1) & \text{if } i = 1 \land 1 < j < n \\
\frac{F(i, j) \cdot F(m-i, n-j+1) + F(i-1, j) \cdot F(m-i+1, n-j+1)}{2} & \text{otherwise}.
\end{cases}
\]

(5.19)

Proofs for the above properties are given in [84].

Informally, the following happens. Each wire is considered to be routed without a detour. Under this condition only a limited number of different paths exists. In total there are \(F(m, n)\) paths and in Lou’s method each path is equally likely. This translates in an associated probability of \(\frac{1}{F(m, n)}\), which appears as a common multiplier in the above formula’s. Since all considered routes are detour-free they all have the same total horizontal and vertical usage. The fact that row and column usages sum up to 1 as presented in Eq. 5.16 and Eq. 5.17 is the direct result of using probabilities and can be interpreted as conservation of track usage.

Off-grid pins

When pins are not perfectly on the grid this is taken into account by scaling the first and last rows and columns of the usage matrix according to actual positions of the pins. In the case of Fig. 5.1, the usages of the entries of the first column of the horizontal usage matrix is scaled by \(\frac{l_p}{W_{\text{tile}}}\).

Short and flat wires

Short and flat wires are treated as special cases. They are also considered to be routed without a detour. In the case of a short wire (a wire that resides in a single tile) the horizontal and vertical usages are simply the horizontal and vertical distances between the pins. For a horizontal (vertical) flat wire (a wire that resides fully in a single row (column)), the vertical (horizontal) usages is divided equally between the leftmost (bottom) and rightmost (top) tiles. The horizontal (vertical) usage for the tiles in between is 1. For the leftmost
(bottom) tile it is scaled by $\frac{L_p}{W_{tile}} (\frac{W_{tile}}{H_{tile}})$, and for the rightmost (top) tile it is scaled by $\frac{L_q}{W_{tile}} (\frac{W_{tile}}{H_{tile}})$ to account for off-grid pins.

**Blockages**

In Lou's approach there are three kinds of blockages. The first called simple blockages are blockages of single tiles. They are dealt with in a post-processing step. The probabilistic usages in those tiles are removed and spread over the surrounding tiles. Nearby tiles get a bigger part of the usage. The second kind of blockages are called line blockages. They block an entire row or column of a wire box. The bounding box of the wire is extended such that a route can be found. The usages are based on the shortest path(s) around the blockage. Finally, blockages are called complex blockages if the above techniques cannot be applied. In that case a maze routing algorithm is used.

**Limitations and drawbacks**

The fact that Lou considers all paths equally likely is not realistic. Real routers try to avoid multi-bend paths. This has a large impact on the probabilistic usages as illustrated by Fig. 5.4. Under Lou's model (Fig. 5.4-a), most of the usage density is around the axis connecting the two pins. Under a model that only considers L-shapes (Fig. 5.4-b), there is no density on this axis. All the density is on the boundary tiles. Under a model that only considers routes with two bends (Fig. 5.4-c), most of the density is also on the periphery of the wire box rather than in the center. In reality, routers will prefer L-shapes, and will only if necessary resort to Z-shapes. As a result multi-bend wires are very rare. Therefore Lou's model does not accurately reflect router behavior in practice.

Part of the motivation for Lou's method is that it is independent of actual routers. It is argued that if one router is used for congestion estimation purposes and a different router is used as the final routing tool accuracy cannot be guaranteed. However, also a probabilistic approach such as presented here does not come with such guarantee. Experience suggests that different global routing tools of similar quality tend to produce similar congestion maps although there will definitely be differences. Therefore a congestion estimate based on observed behavior of routing tool A will probably also produce reasonable results when routing tool B is used. In any case, observations made on router A are more representative of another router B than some of Lou's assumptions as supported by section 5.7.

Designers are interested in routability. Ultimately, routability concerns the question whether a design is routable for a given tool. Congestion estimation algorithms should be integrated in an EDA tool set. In that case it is possible to use knowledge about the routers that will be used during congestion analysis. With this in mind there is no strong motivation for pursuing router-independence. It may in fact be desirable to model peculiarities or decision strategies of particular routers. Different routers may for example make different tradeoffs when there is the choice between a few highly congested tiles and many slightly congested tiles.

Another weakness in the approach is the handling of blockages. Although many ideas are proposed in the paper, blockages are essentially handled in a post-processing step. Probabilistic approaches gain much in speed by ignoring interactions with other nets and
5.5 Improved congestion model

In this section, an improvement over Lou’s method is presented. It is based on empirical observations regarding an industrial router. In other words, we present a probabilistic congestion estimation method that is tuned towards a specific industrial router. Instead of deriving probabilities from properties of the design only, as in Lou’s method, the probabilities in the proposed method are derived both the design and measured router behavior.

### 5.5.1 Foundation of the model

The probabilistic congestion estimation method as presented in this chapter is based on a number of observations and ideas:

1. Nets should be decomposed into two-pin wires as in the global router. The industrial router [88] used in this chapter uses a Minimum Spanning Tree algorithm and the same method is used in the congestion estimation method.

2. The large majority of wires are without a detour. When congestion becomes a problem routers use different cost models and rip-up and reroute techniques to dis-
tribute the congestion, typically leading to a few detoured wires. Because of the
complex dynamics of the routing algorithms, detouring of individual wires cannot
be predicted. Detouring is therefore ignored.

3. As a secondary criterion (global) routers minimize bends and vias. Accurate conges-
tion estimation methods must take this into account.

4. The speed of probabilistic methods is due to the fact that interaction between wires
is ignored. After initial prediction it can be taken into account by observing the con-
gestion levels. Probabilistic methods tend to exaggerate congestion problems and
this can be fixed in a post-processing step as is done in other works[69, 110]. Such a
post-processing step takes run time and since the primary objective for probabilistic
methods is speed such a step is (at least initially) not desirable.

The base algorithms that are used in global routing minimize wire length. Experimen-
tal evidence presented in [137] where the same test suite is used as in this chapter shows
that they are effective in this: on average only 1.40% of the two-pin nets in the test suite
are detoured). Consequently, the assumption of detour-freeness is valid and will be used
in the presented algorithms.

Even though little literature exists on the reduction of bends and vias during global
routing we observe that industrial routers do not use excessive amounts of vias. The results
in [137] indicate that only 1.2% of the two-pin nets in the test suite are routed with more
than two bends. Therefore, no wire shapes with more than two bends are considered in
the proposed congestion model. Similar models are used in e.g. [21, 85].

Post-processing steps such as described in [69, 110] can be used to improve the pre-
diction quality of probabilistic methods by simply redistributing congestion from areas
with (very) high congestion to nearby areas with lower congestion. The justification for
this approach is that in practice real (global) routers tend to do better than “probabilistic
routers”. By simply moving all congestion values towards the average the peak and total
error is reduced, but this may not be the only goal of the analysis. More important is that
the locations of congestion hotspots are accurately predicted. In fact, as shown in [51],
pessimistic congestion estimation is useful in practice when it is employed in order to re-
move congestion. If desired post-processing techniques as in [69, 110] can be used on top
of the presented method at the cost of additional run time.

5.5.2 Usage of short wires

Short wires are wires of which both pins reside in the same tile. In our model, they are
treated the same as in Lou’s model. Under the assumption of detour-free routing, the
length of a short wire \( w = (a, b) \) is equal to the manhattan distance between its pins \( a \) and
\( b \). Accordingly, a short wire contributes the horizontal and vertical distance between the
pins to the horizontal and vertical usage of the tile the pins reside in. As illustrated by
Fig. 5.5, these usages are normalized for tile size. The probabilistic usages become

\[
U_{short}^{\text{hor}}(w, r, c) = \begin{cases} 
\frac{|x_a - x_b|}{\text{Tile}} & r = r_{\text{min}}(w) \land c = c_{\text{min}}(w) \\
0 & \text{otherwise,}
\end{cases}
\]

Some academic routers tend to produce many more bends. The presented algorithm mimics the industrial
routers of Magma[88]
5.5 Improved congestion model

\[ U_{\text{hor}} = \frac{|x_a - x_b|}{W_{\text{tile}}} \]

\[ U_{\text{ver}} = \frac{|y_a - y_b|}{H_{\text{tile}}} \]

Figure 5.5: Usages due to short wires.

\[ U^\text{short}_{\text{ver}}(w, r, c) = \begin{cases} \frac{|y_a - y_b|}{H_{\text{tile}}} & r = r_{\text{min}}(w) \land c = c_{\text{min}}(w) \\ 0 & \text{otherwise} \end{cases} \]

\[ (5.21) \]

where implicitly is used that \( w \) consists of the two pins \( a \) and \( b \), and \( r \) and \( c \) represent the row and column coordinate of the tile the pins reside in, respectively. The notation \( r_{\text{min}}(w), r_{\text{max}}(w), c_{\text{min}}(w) \), and \( c_{\text{max}}(w) \) is adopted for the minimum and maximum values of the rows and columns that are spanned by the pins of the wire \( w \), respectively.

5.5.3 Usage of flat wires

A horizontal flat wire \( w = (a, b) \) is a wire of which the two pins \( a \) and \( b \) are in the same row, but not in the same column. A vertical flat wire has two pins in the same column but not in the same row. Note that if the pins of a wire are both in the same row and the same column, it is a short wire. Flat wires are treated similarly as in Lou's model. Without loss of generality, let \( a \) be the leftmost pin. As illustrated by Fig. 5.6, a horizontal flat wire contributes 1 to the horizontal usage of each tile in its wire box. Only the leftmost and rightmost tiles get a scaled usage to account for the actual pin positions:

\[ U^\text{hor flat}_{\text{hor}}(w, r, c) = \begin{cases} \frac{r_a}{W_{\text{tile}}} & r = r_{\text{min}}(w) \land c = c_{\text{min}}(w) \\ \frac{l_b}{W_{\text{tile}}} & r = r_{\text{min}}(w) \land c = c_{\text{max}}(w) \\ 1 & r = r_{\text{min}} \land c_{\text{min}}(w) < c < c_{\text{max}}(w) \\ 0 & \text{otherwise} \end{cases} \]

\[ (5.22) \]

where \( c_{\text{min}}(w) \) and \( c_{\text{max}}(w) \) denote the leftmost and rightmost columns spanned by the pins of the wire. \( r \) is the single row coordinate of the flat net. The vertical usage is simply distributed equally over all involved tiles:

\[ U^\text{ver flat}_{\text{hor}}(w, r, c) = \begin{cases} \frac{|y_a - y_b|}{w_w \cdot H_{\text{tile}}} & r_{\text{min}}(w) \leq r \leq r_{\text{max}}(w) \land c = c_{\text{min}}(w) \\ 0 & \text{otherwise} \end{cases} \]

\[ (5.23) \]

where \( w_w \) represents the width of the wire box of \( w \).

The formulas for vertical flat wires are similar. For the horizontal usages we find

\[ U^\text{ver flat}_{\text{hor}}(w, r, c) = \begin{cases} \frac{|x_a - x_b|}{h_w \cdot W_{\text{tile}}} & r = r_{\text{min}}(w) \land c_{\text{min}}(w) \leq c \leq c_{\text{max}}(w) \\ 0 & \text{otherwise} \end{cases} \]

\[ (5.24) \]
and for the vertical usages we find

\[
U^{ver\ flat}_{ver}(w, r, c) = \begin{cases} 
\frac{r}{H_{tile}} & r = r_{min}(w) \land c = c_{min}(w) \\
\frac{r}{H_{tile}} & r = r_{max}(w) \land c = c_{min}(w) \\
1 & r_{min}(w) < r < r_{max}(w) \land c = c_{min}(w) \\
0 & \text{otherwise},
\end{cases}
\]

The flat wire usage is either horizontal or vertical flat wire usage or zero if the wire is not flat. Mathematically, this is expressed as

\[
U^{flat}_{hor}(w, r, c) = \begin{cases} 
U^{hor\ flat}_{hor}(w, r, c) & \text{if } r_{max}(w) - r_{min}(w) \geq 1 \land c_{min}(w) = c_{max}(w) \\
0 & \text{otherwise},
\end{cases}
\]

and

\[
U^{flat}_{ver}(w, r, c) = \begin{cases} 
U^{ver\ flat}_{ver}(w, r, c) & \text{if } r_{max}(w) - r_{min}(w) \geq 1 \land c_{min}(w) = c_{max}(w) \\
0 & \text{otherwise}.
\end{cases}
\]

### Difference with Lou’s method

Conceptually the differences between the proposed method and Lou’s method are not very big for flat nets, but there can be differences in the results. For a horizontal flat net Lou divides the vertical usage over the two tiles the pins reside in. In our approach this usage is divided over all tiles in the wire box. The key observation here is that not the global router but the track and detailed router decides upon the actual distribution of this usage. In practice detailed routers do not necessarily assign this usage to the tiles containing the pins.

#### 5.5.4 Usage of L-shapes

If the wire box of a wire \( w = (a, b) \) spans at least two rows and two columns (i.e. the wire is not a short or flat wire) a rectilinear wire connecting the two pins needs to have at least one bend. An L-shaped route with a single bend as shown in Fig. 5.7 is most likely since routers are effective in minimizing bends and vias. It can be thought of as the combination
5.5 Improved congestion model

of a horizontal flat wire and a vertical flat wire. The usage due to an L-shape is the sum of
the usages of the two virtual flat wires.

Of course there are always two L-shapes possible as illustrated by Fig. 5.8. One starting
from the bottom-left pin with a horizontal flat wire and one starting with a vertical flat
wire. Each realization is considered equally likely and therefore both L-shapes have an
associated probability of $\frac{1}{2}$. Consequently, the usages due to the L-shape are multiplied
by $\frac{1}{2}$. Effectively, the L-usage can be found by calculating the usage due to four flat wires
and scaling these usages by a factor $\frac{1}{2}$ that represents the probability of each L-shape. The
resulting formulas are shown in Fig. 5.7.

A special case are the small L-shapes. They span a $2 \times 2$ wire box: $w_w = h_w = 2$. The
difference with other L-shapes is that no Z-shapes or multi-bend realizations are possible.
Their calculation is the same as for other L-shapes, but they are treated differently when
their usages are combined with other shapes.
5.5.5 Usage of Z-shapes

Z-shapes are the most interesting case that is considered. If both the width \( w \) and height \( h \) of a wire's wire box \( w \times h \) are larger than 2 there are two possible orientations: horizontal and vertical, named after the orientation of the center piece of the Z-shape. In Fig. 5.9 a vertical Z-shape is shown with the associated usages.

Equivalently to L-shapes that can be thought to exist of two flat wires, a Z-shape can be thought to exist of three flat wires. Exact pin positions are important for the calculation of the usages due to a flat wire. The virtual pin positions of the center virtual flat wire are not entirely determined by the positions of the real pins. A wire passing through a tile will on average go through the center of the tile. Therefore the center piece of a Z-shape is modeled to go through the middle of the tiles in the usage calculation.

It is possible to calculate Z-shape usage by generating all flat nets and multiplying everything with an appropriate probability. Unfortunately, the number of possible Z-shapes grows linearly with the size of the wire box in both dimensions. A wire spanning a 10 × 10 wire box for example has already 16 different Z-shapes or \( 3 \cdot 16 = 48 \) associated virtual flat wires. Each of the three virtual flat wires spans 10 tiles. Therefore, adding the virtual flat wire usages requires \( 3 \cdot 10 \cdot 16 = 480 \) tile visits or \( 2 \cdot 480 = 960 \) map entry visits since each tile has both a horizontal and vertical congestion associated with it. With possibly millions of wires, faster ways to calculate the Z-usage are needed.

Let us consider the combined usage of all vertical Z-shapes for a wire \( w(a, b) \) with \( a \) the leftmost bottom pin as illustrated by Fig. 5.10. The usage in the top and bottom rows are found similarly. Therefore only the bottom row is discussed. As illustrated by Fig. 5.10, let us use \( c^w(d) \) to denote the column of a tile in column \( d \), relative to the leftmost tile in the wire box of a wire \( w \) or equivalently,

\[
c^w(d) = d - c_{\min}(w). \quad (5.28)
\]

Similarly, let us use for the rows

\[
r^w(s) = s - r_{\min}(w). \quad (5.29)
\]
There are two terms contributing to the horizontal usage of a tile in the bottom row: one for the case where a bend occurs in the tile and one for the case where a bend occurs in a tile to the right of the tile since the horizontal wire piece leading to that bend has to pass through the tile. In total, the number of horizontal and vertical Z-shapes is

$$|Z_{\text{hor}}| = h_w - 2$$  \hspace{1cm} (5.30)

and

$$|Z_{\text{ver}}| = w_w - 2,$$  \hspace{1cm} (5.31)

respectively. Assuming all Z-shapes are equally likely the probability that an upward bend occurs in a tile is expressed as

$$p_{ub}(w, r, c) = \begin{cases} \frac{1}{w_w - 2} & r = r_{\text{min}}(w) \land 1 \leq c^w(c) \leq w_w - 2 \\ 0 & \text{otherwise.} \end{cases}$$  \hspace{1cm} (5.32)

The tiles in the first and last column are excluded because if the vertical bar starts in one of those tiles we are dealing with an L-shape instead of a Z-shape. The probability that such a bend occurs to the right of a tile is

$$p_{b2r}(w, r, c) = \begin{cases} \frac{w_w - c^w(c) - 2}{w_w - 2} & r = r_{\text{min}}(w) \land 0 \leq c^w(c) \leq w_w - 1 \\ 0 & \text{otherwise.} \end{cases}$$  \hspace{1cm} (5.33)

In case of a bend in a column the horizontal contribution will be $\frac{1}{2}$ since the wire will on average go through the center of the tile. For a bend to the right the contribution is obviously 1. Now, the total probabilistic horizontal usage is expressed as

$$U_{Z_{\text{hor}}}(w, r, c) = \frac{p_{b2r}(w, r, c) + p_{ub}(w, r, c) \cdot \frac{1}{2}}{W_{\text{tile}}}$$ \hspace{1cm} (5.34)

where the usages in the first and last column take the exact pin positions into account, similar to what is done in the calculation for flat wires.

Vertical usages only appear when a bend occurs in that column. The contribution depends on the exact position of the left-bottom pin:

$$U_{Z_{\text{ver}}}(w, r, c) = \begin{cases} p_{ub}(w, r, c) \cdot \frac{r_a}{W_{\text{tile}}} & r = r_{\text{min}}(w) \land 1 \leq c^w(c) \leq w_w - 2 \\ r = r_{\text{min}}(w) \land c^w(c) = 0 \\ r = r_{\text{min}}(w) \land c^w(c) = w_w - 1, \end{cases}$$  \hspace{1cm} (5.35)

The tiles in the center (not in the leftmost or rightmost columns or in the bottommost or topmost rows) only have vertical usage due to vertical Z-shapes. Since single vertical Z-shapes occupy a full track in the center tiles the probabilistic usage boils down to the probability of a vertical Z-shape going through that tile, $p_{b}(r, c)$:

$$U_{Z_{\text{hor}}}(w, r, c) = 0 \quad 1 \leq c^w(c) \leq w_w - 2 \land 1 \leq r^w(r) \leq h_w - 2,$$  \hspace{1cm} (5.36)

and

$$U_{Z_{\text{ver}}}(w, r, c) = p_{b}(r_{\text{min}}, c) = \frac{1}{w_w - 2} \quad 1 \leq c^w(c) \leq w_w - 2 \land 1 \leq r^w(r) \leq h_w - 2.$$  \hspace{1cm} (5.37)

The formulas are illustrated in Fig. 5.10.
In the analysis above, only vertical Z-shapes are taken into account. The usages due to horizontal Z-shapes are found similarly and denoted by $U_{Z\text{hor}}^Z(w, r, c)$ and $U_{Z\text{ver}}^Z(w, r, c)$. The probability of a Z-shape being vertical is simply the number of vertical Z-shapes over the total number of Z-shapes:

$$s_{ver} = \frac{|Z_{ver}|}{|Z_{hor}| + |Z_{ver}|} = \frac{w_w - 2}{w_w + h_w - 4}. \quad (5.38)$$

Equivalently, the probability of a Z-shape being horizontal is

$$s_{hor} = \frac{|Z_{hor}|}{|Z_{hor}| + |Z_{ver}|} = \frac{h_w - 2}{w_w + h_w - 4}. \quad (5.39)$$

Now, by using these probabilities, the total Z-usage for this wire is found as

$$U_{Z\text{hor}}^Z(w, r, c) = s_{hor} \cdot U_{Z\text{hor}}^Z(w, r, c) + s_{ver} \cdot U_{Z\text{ver}}^Z(w, r, c) \quad (5.40)$$

$$U_{Z\text{ver}}^Z(w, r, c) = s_{hor} \cdot U_{Z\text{ver}}^Z(w, r, c) + s_{ver} \cdot U_{Z\text{ver}}^Z(w, r, c). \quad (5.41)$$

### 5.5.6 Combination of usages

The analysis in the previous sections are based on probabilities. The cases for L- and Z-shapes were treated independently and the resulting probabilistic usages are combined again in probabilistic fashion in this section. Because of the objectives of global routing as
discussed in Chapter 3 and Chapter 7. L-shapes are more attractive than Z-shapes: they have less bends. In practice however some Z-shapes cannot be avoided as shown in section 5.7. Probabilistic usage due to L- and Z-shapes are combined as follows.

\[
U_{\text{hor}}^{LZ}(w, r, c) = \alpha \cdot U_{\text{hor}}^L(w, r, c) + (1 - \alpha) \cdot U_{\text{hor}}^Z(w, r, c) 
\]

(5.42)

\[
U_{\text{ver}}^{LZ}(w, r, c) = \alpha \cdot U_{\text{ver}}^L(w, r, c) + (1 - \alpha) \cdot U_{\text{ver}}^Z(w, r, c),
\]

(5.43)

where \(\alpha\) accounts for the ratio between L- and Z-shapes as measured from actual router behavior:

\[
\alpha = \frac{|L_{routed}^Z|}{|L_{routed}^L| + |Z_{routed}^L|},
\]

(5.44)

where \(L_{routed}^L\) and \(Z_{routed}^L\) are the sets of two pin wires that are routed with a specific router with one and two bends, respectively, over a specific set of benchmarks. These sets only include wires that could potentially be routed with two bends, i.e. they exclude small L-shapes, hence the superscript \(LZ\).

Now the horizontal and vertical usages are expressed as

\[
U_{\text{hor}}(w, r, c) = \begin{cases}
U_{\text{hor}}^L(w, r, c) & \text{if } (w_w \geq 2 \land h_w > 2) \lor (w_w > 2 \land h_w \geq 2) \\
U_{\text{hor}}^Z(w, r, c) & \text{if } w_w = 2 \land h_w = 2 \\
U_{\text{short}}^r(w, r, c) & \text{if } r = r_{\text{min}}(w) \land c = c_{\text{min}}(w) \\
U_{\text{flat}}^r(w, r, c) & \text{otherwise},
\end{cases}
\]

(5.45)

and

\[
U_{\text{ver}}(w, r, c) = \begin{cases}
U_{\text{ver}}^L(w, r, c) & \text{if } (w_w \geq 2 \land h_w > 2) \lor (w_w > 2 \land h_w \geq 2) \\
U_{\text{ver}}^Z(w, r, c) & \text{if } w_w = 2 \land h_w = 2 \\
U_{\text{short}}^r(w, r, c) & \text{if } r = r_{\text{min}}(w) \land c = c_{\text{min}}(w) \\
U_{\text{flat}}^r(w, r, c) & \text{otherwise}.
\end{cases}
\]

(5.46)

The total prediction can now be found by summing over all wires:

\[
U_{\text{hor}}(r, c) = \sum_{w \in W} U_{\text{hor}}(w, r, c),
\]

(5.47)

and

\[
U_{\text{ver}}(r, c) = \sum_{w \in W} U_{\text{ver}}(w, r, c),
\]

(5.48)

where \(W\) represents the set of all wires in the design.

**Practical implementation**

A practical implementation of the above probabilistic congestion estimation procedure is outlined in Fig. 5.11. Essentially, it consists of two steps: first, the multi-pin nets in \(N\) are broken up by either a minimum spanning tree algorithm or a steiner tree algorithm. Next, the probabilistic usages as described above are stamped into the horizontal and vertical usage maps \(H\) and \(V\).

Essentially, \(\alpha\) is the part of the two-pin nets that is optimally routed with only one bend. If this is not possible the router will use two bends, and so on. A better router will
PCE-BASIC($N, \alpha$)

1. $H \leftarrow$ CREATE-EMPTY-MAP()
2. $V \leftarrow$ CREATE-EMPTY-MAP()
3. $W \leftarrow$ CREATE-EMPTY-SET()
4. for each net $n$ in $N$
5.     $W \leftarrow W \cup$ DECOMPOSE-NET($n$) \triangleright Minimum spanning or steiner tree
6. for each wire $w \in W$
7.     do if $w$ is short
8.         then ADD-SHORT-USAGE($w, H, V$)
9.     elseif $w$ is flat
10.        then ADD-FLAT-USAGE($w, H, V$)
11. elseif $w$ is small-L
12.        then ADD-L-USAGE($w, H, V$)
13. else ADD-SCALED-L-USAGE($\alpha, w, H, V$)
14.     ADD-SCALED-Z-USAGE($1 - \alpha, w, H, V$)
15. return $H, V$

Figure 5.11: The PCE-BASIC algorithm.

be able to route more nets optimally, resulting in a larger value of $\alpha$. If a design is more congested a router will have a harder job resulting in a lower value for $\alpha$ for that design. In other words, harder designs should get a lower value for $\alpha$. Average congestion can be estimated using wire length estimation methods and one could make $\alpha$ a function of average congestion. The problem with such an approach however is that the number of bends only increases in congested areas and these congested areas cannot be predicted accurately from wire lengths estimates alone.

Tool quality and tool characteristics are implicitly taken into account by using a particular tool when empirically finding $\alpha$. Different routers may have slightly different objectives or cost models resulting in different values of $\alpha$. Based on experience, contacts and intuition we expect that industrial routers have similar behavior. Therefore we expect that the value for $\alpha$ as derived in the results section is applicable for many routers.

5.5.7 Properties of usages

The usages due to each wire have certain properties. These can be used to speed up implementations or check implementations for correctness.

The sum property

The idea behind probabilistic congestion prediction is to spread a wire over its possible (likely) realizations. The total horizontal and vertical usage can be determined up front, since only detour-free realizations are considered. Conceptually the total horizontal and vertical usages are spread over the tiles in the wire box. As illustrated by Fig. 5.12 this means that the horizontal usages sum up to 1 for each column except the first and last
5.5 Improved congestion model

\[ \sum U_{\text{hor}} = 1 \]
\[ \sum U_{\text{ver}} = 1 \]

\[ \sum U_{\text{hor}} = \frac{r_a}{W_{\text{tile}}} \]

Figure 5.12: The sum property of probabilistic congestion estimation.

ones. In these columns the amount to which the usages sum up depends on the exact pin positions:

\[ \sum_{r=r_{\text{min}}(w)}^{r_{\text{max}}(w)} U_{\text{hor}}(w, r, c) = \begin{cases} \frac{r_a}{W_{\text{tile}}} & c = c_{\text{min}}(w) \\ 1 & c_{\text{min}}(w) < c < c_{\text{max}}(w) \\ \frac{b_b}{W_{\text{tile}}} & c = c_{\text{max}}(w). \end{cases} \]  
\[ (5.49) \]

and

\[ \sum_{c=c_{\text{min}}(w)}^{c_{\text{max}}(w)} U_{\text{ver}}(w, r, c) = \begin{cases} \frac{r_a}{W_{\text{tile}}} & r = r_{\text{min}}(w) \\ 1 & r_{\text{min}}(w) < c < c_{\text{max}}(w) \\ \frac{b_b}{W_{\text{tile}}} & r = r_{\text{max}}(w). \end{cases} \]  
\[ (5.50) \]

Equivalently, the sum over the vertical usages sums up to 1 for each row, except for the first and last row:

\[ \sum_{c=c_{\text{min}}(w)}^{c_{\text{max}}(w)} U_{\text{ver}}(w, r, c) = \begin{cases} \frac{r_a}{W_{\text{tile}}} & r = r_{\text{min}}(w) \\ 1 & r_{\text{min}}(w) < c < c_{\text{max}}(w) \\ \frac{b_b}{W_{\text{tile}}} & r = r_{\text{max}}(w). \end{cases} \]  
\[ (5.51) \]

In these formulas, \( a \) is the lower-left pin, and \( b \) the upper-right pin. Similar formulas can be derived for configurations with one pin in the upper-left corner, and one in the lower-right corner.

The rotation property

The stamps associated with a wire’s wire box are symmetric with respect to a rotation of 180°, except for not symmetric positions of the pins. This property stems from the fact that in our approach both pins are equivalent and a rotation of 180° is equivalent to swapping the pins. The property is illustrated in Fig. 5.13 and the following formulas are found.

\[ U_{\text{hor}}(w, r, c) = \begin{cases} U_{\text{hor}}(w, r', c') \cdot \frac{r_a}{l_b} & c = c_{\text{min}}(w) \\ U_{\text{hor}}(w, r', c') \cdot \frac{l_b}{r_a} & c = c_{\text{max}}(w) \\ U_{\text{hor}}(w, r', c') & \text{otherwise}, \end{cases} \]  
\[ (5.52) \]

and

\[ U_{\text{ver}}(w, r, c) = \begin{cases} U_{\text{ver}}(w, r', c') \cdot \frac{r_a}{b_b} & r = r_{\text{min}}(w) \\ U_{\text{ver}}(w, r', c') \cdot \frac{b_b}{r_a} & r = r_{\text{max}}(w) \\ U_{\text{ver}}(w, r', c') & \text{otherwise}. \end{cases} \]  
\[ (5.53) \]
Here, the rotated coordinates are given by

\[
\begin{align*}
    r' &= r_{\max}(w) + r_{\min}(w) - r \\
    c' &= c_{\max}(w) + c_{\min}(w) - c.
\end{align*}
\]  

(5.54)

Note that for a leftmost pin \( p \), \( 0 \leq l_p < W_{\text{tile}} \) and \( 0 < r_p \leq W_{\text{tile}} \) and for a rightmost pin \( p \), \( 0 < l_p \leq W_{\text{tile}} \) and \( 0 \leq r_p < W_{\text{tile}} \).

Let us illustrate the above formulas with an example derived from Fig. 5.13. In the figure, the horizontal usages of the bottom-left tile and the top-right tile, \( U_{\text{hor}}(0,0) \) and \( U_{\text{ver}}(2,2) \) are equal because both pins are exactly on the corners of their respective tiles. Let us now translate the pins (within the current tiles), and we find that \( U'_{\text{hor}}(0,0) = r_a W_{\text{tile}} U_{\text{hor}}(0,0) \) and \( U'_{\text{hor}}(2,2) = l_b W_{\text{tile}} U_{\text{hor}}(2,2) \). Using the fact that \( U_{\text{hor}}(0,0) = U_{\text{ver}}(2,2) \) we find that \( U'_{\text{hor}}(2,2) = l_b W_{\text{tile}} U_{\text{hor}}(0,0) = l_b W_{\text{tile}} W_{\text{tile}} r_a U'_{\text{hor}}(0,0) \). This is essentially how the above formulas are derived.

The rotation property can be used to more efficiently calculate wire usages.

5.5.8 Blockages

The parts of a layer that may not be used for routing (and placement) due to pre-routes or user constraints, are called blockages. The main motivation for probabilistic congestion estimation is speed. This speed advantage is obtained by ignoring the interaction between wires or with the availability of routing resources. Therefore no completely satisfactory way of dealing with blockages exists in a probabilistic framework. Nonetheless, blockages must be taken into account and if only a few and geometrically simple blockages exist a few tricks are available.

(Partial) blockages are subtracted from the capacity of a given tile and affect the congestion values thereby. Full blockages block all layers in a (set of) tile(s). They are dealt with similarly to simple blockages in [84]: each tile with a distance to the blockage lower than \( D \) gets a weight associated with it: \( w = 2^{-d} \cdot n \), where \( d \) is the manhattan distance to the blockage and \( n \) the number of unblocked neighboring tiles. After running the probabilistic usage estimation the total usage of the blockage is divided over the neighboring tiles proportional to their weights. The difference with [84] is that we can also handle blockages larger than a single tile. Line blockages block an entire row or column of the bounding box of a net. It is solved by introducing two virtual pins at the periphery of the blockage

---

\( ^6 \)Technically pre-routes are treated differently by routers since they also have an electrical effect.
(Fig. 5.14). Unfortunately, calculating which blockages affect which wires takes run time and in the presence of more than a few blockages, probabilistic methods are not effective.

5.6 Implementation

Two different implementations of the probabilistic method described in this chapter have been created. The first implementation called Pce-tcl (Probabilistic Congestion Estimation - TCL) was implemented in M-TCL, the scripting language of the Magma tools. It is a precise implementation of the method as described in this chapter that serves as a proof of concept. Because M-TCL is an interpreted language with only limited speed the method is not very fast. Pce-tcl is described in detail in [7, 137].

A very fast probabilistic congestion estimation program called Pce was created in C++. It uses the basic method as described in this chapter but a number of simplifications are made. Pce is a very fast and practical tool and is described in detail in [138].

5.6.1 M-TCL implementation

Pce-tcl was developed mainly as a proof of concept. It consists of a number of M-TCL scripts. It is a direct implementation of the theory as described in this chapter and illustrated in Fig. 5.15. The inputs are the set of wires and the parameter α for combination of L-shapes and Z-shapes. The usage maps H and V are returned. There are a number of \( W_{tile} \times H_{tile} \) arrays in memory that store the different usages, where \( W_{tile} \times H_{tile} \) depicts the size of the placement area, in terms of tiles. There are horizontal and vertical usage arrays for

1. short wires
2. flat wires
3. small L-shapes
4. remaining L-shapes
5. Z-shapes.

---

7 In physical synthesis systems, some awareness of location exists throughout the flow. Most objects such as placeable objects and wires are therefore stored in data structures such as quadtrees or kdtrees. Such data structures may also be used to see which blockages impact which wires.
PCE-TCL($N, \alpha$)

1. $H\text{short} \leftarrow \text{CREATE-EMPTY-MAP}()$
2. $H\text{flat} \leftarrow \text{CREATE-EMPTY-MAP}()$
3. $H\text{small} \leftarrow \text{CREATE-EMPTY-MAP}()$
4. $H\text{L} \leftarrow \text{CREATE-EMPTY-MAP}()$
5. $H\text{Z} \leftarrow \text{CREATE-EMPTY-MAP}()$
6. $V\text{short} \leftarrow \text{CREATE-EMPTY-MAP}()$
7. $V\text{flat} \leftarrow \text{CREATE-EMPTY-MAP}()$
8. $V\text{small} \leftarrow \text{CREATE-EMPTY-MAP}()$
9. $V\text{L} \leftarrow \text{CREATE-EMPTY-MAP}()$
10. $V\text{Z} \leftarrow \text{CREATE-EMPTY-MAP}()$

for each net $n$ in $N$

\[ W \leftarrow W \cup \text{DECOMPOSE-NET}(n) \]

for each wire $w$

\[ \text{if } w \text{ is short } \]

\[ \text{then ADD-SHORT-USAGE}(w, H\text{short}, V\text{short}) \]

\[ \text{elseif } w \text{ is flat } \]

\[ \text{then ADD-FLAT-USAGE}(w, H\text{flat}, V\text{flat}) \]

\[ \text{elseif } w \text{ is small-L } \]

\[ \text{then ADD-L-USAGE}(w, H\text{small}, V\text{small}) \]

\[ \text{else } \text{ADD-L-USAGE}(w, H\text{L}, V\text{L}) \]

\[ \text{ADD-Z-USAGE}(w, H\text{Z}, V\text{Z}) \]

\[ H \leftarrow H\text{short} + H\text{flat} + H\text{small} + \alpha \cdot H\text{L} + (1 - \alpha) \cdot H\text{Z} \]

\[ V \leftarrow V\text{short} + V\text{flat} + V\text{small} + \alpha \cdot V\text{L} + (1 - \alpha) \cdot V\text{Z} \]

return $H$, $V$

**Figure 5.15:** The PCE-TCL algorithm.

Storing the different usages like this is not strictly necessary but enables the designer to e.g. specify a different value for $\alpha$ or to inspect the different usage maps. The dimensions of the arrays are sufficiently small not to cause problems. The total usages are found by simply adding all the usages; the usage maps for L-shapes and Z-shapes are scaled by $\alpha$ and $(1 - \alpha)$, respectively.

### 5.6.2 C++ implementation

Pce was developed with speed as the main consideration. It was intended to be compared against a publicly available academic global router called Labyrinth[71], and other routers that use similar routing models. For a fair comparison, different methods should be compared on publicly available benchmarks. Since no “congestion estimation benchmarks” exist, the standard global routing benchmarks that come with Labyrinth were selected.
Model of routing area

Pce-tcl and Pce work on different models of the routing area. Pce-tcl works directly on the tile model of the routing area. Pce works on the global routing graph of the routing area, i.e. on a 2-D grid graph. Capacities are associated with the edges in this model and consequently the congestion maps that are generated refer to edges instead of tiles. A chip area that is divided in \( m \times n \) tiles produces \( m \times n \) horizontal and vertical congestion maps with Pce-tcl and an \( m \times n - 1 \) horizontal and \( m - 1 \times n \) vertical congestion map with Pce. It was shown in Chapter 3.1 how the two congestion maps are related.

Dealing with pin positions

The most important difference between Pce and Pce-tcl is how pins are dealt with. In Pce-tcl exact pin positions are taken into account. Pce imports global routing problems from the Labyrinth benchmark suite. In these benchmarks pin positions are described in terms of tiles, i.e. pin positions are rounded to the center of their tiles as illustrated by Fig. 5.16.

The rounding of pin positions definitely has an influence on the outcome of the prediction. This impact should be small however. Because any congested tile contains many pins the pin positions should average out to the center of the tile although in a few cases pin positions may be important. Additionally, models similar to the described model are standard in global routing and give acceptable congestion estimates. In fact, tile size has explicitly been chosen such that global routing gives a satisfactory congestion estimation, i.e. they are sufficiently small.

Stamps and reuse

Pce is tuned for speed. Although calculation of the usages of a single wire is blazingly fast with potentially millions of wires such calculations can add up. Under the assumption of pins in the center many wires contribute the same patterns to the overall congestion maps except for a translation. These patterns are referred to as stamps and illustrated by Fig. 5.17.

Pce exploits the fact that many wires contribute the same stamps by storing and retrieving calculated stamps in and from a hash map; this hash map serves as a stamp library. The algorithm is outlined in Fig. 5.18. With an efficient hash map implementation the run time of Pce is determined by the time the stamping of the stamps in the congestion maps takes. Unfortunately, the size of the stamp grows quadratically with the length of the stamp.

---

8Hash maps are used to map a key to a value. In this case the key is calculated from the content of the stamp, i.e. its \( \Delta r \) and \( \Delta y \) and orientation. The value is the stamp itself.
wire. However, due to effective placement the large majority of wires is (very) short in practice and this does not pose a serious problem. The stamp calculation in \texttt{CREATE-STAMP()} is based on the theory in this chapter. We use $\alpha = 0.60$ since this is the value that was found experimentally.

### 5.7 Experimental results

Many experiments were performed with the tools \texttt{Pce-tcl} and \texttt{Pce}. In this section only results obtained with \texttt{Pce-tcl} will be presented. Experiments with \texttt{Pce} are discussed in chapter 6.

In our experiments the Blast Chip 4.0 physical synthesis software by Magma Design Automation\cite{magma} was used. The benchmarks as summarized in Table \ref{benchmarks} are complete chips or large blocks, mostly from an industrial source. The column “\#cells” represents the number of standard cells in the design. The largest design has almost 200,000 cells. The maximum number of nets as shown in the column “\#nets” is almost 250,000. The number of tiles represented by the column “\#tiles” has an obvious impact on the run time and is maximally a little over 750,000 in this benchmark set. The values in the column “est. wire length” are wire length estimates based on the Magma model that takes both bounding box of a net and pin count into account. The utilization varies roughly between 50%
and 100% so designs of different ‘difficulty’ are covered. In the last column the percentage of nets that is a two-pin net is shown. Since our congestion model is based on two-pin nets the high percentages shown in this column are a strong indicator that such a model is valid.

Table 5.1: The benchmarks

<table>
<thead>
<tr>
<th>chip</th>
<th>#cells</th>
<th>#nets</th>
<th>#tiles</th>
<th>est. wire length [m]</th>
<th>utilization [%]</th>
<th>2-pin [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>194527</td>
<td>242251</td>
<td>752556</td>
<td>23.155</td>
<td>57.4</td>
<td>69</td>
</tr>
<tr>
<td>b</td>
<td>10180</td>
<td>10862</td>
<td>22650</td>
<td>0.745</td>
<td>50.7</td>
<td>71</td>
</tr>
<tr>
<td>c</td>
<td>49709</td>
<td>45315</td>
<td>327750</td>
<td>7.072</td>
<td>71.3</td>
<td>55</td>
</tr>
<tr>
<td>d</td>
<td>7737</td>
<td>7830</td>
<td>408321</td>
<td>2.394</td>
<td>99.1</td>
<td>64</td>
</tr>
<tr>
<td>e</td>
<td>17102</td>
<td>20028</td>
<td>33920</td>
<td>1.140</td>
<td>85.7</td>
<td>80</td>
</tr>
<tr>
<td>f</td>
<td>46871</td>
<td>45801</td>
<td>55328</td>
<td>3.607</td>
<td>84.8</td>
<td>75</td>
</tr>
</tbody>
</table>

5.7.1 Routing probabilities

One of the basic assumptions common to most probabilistic approaches is that nets are not detoured. For the benchmark suite the detour of the two-pin nets after routing was measured. Obviously, the detour depends on the router that was used. In this case the Magma[88] flow was used. As illustrated by Fig. 5.2, it turned out that the number of detoured two-pin nets was negligible: only 1.40% on average. Even the benchmark with utilization of almost 100% only detoured less than 3% of the two-pin nets. Consequently, considering only detour-free routes is an accurate approximation, even for high-density designs.

Table 5.2: Detours of two-pin nets

<table>
<thead>
<tr>
<th>chip</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>% detoured two-pin nets</td>
<td>0.60</td>
<td>0.15</td>
<td>3.73</td>
<td>2.68</td>
<td>0.83</td>
<td>0.39</td>
<td>1.40</td>
</tr>
</tbody>
</table>

The improvement of our approach over Lou’s approach is due to the observation that in practice most nets are routed with only few bends. Let us define a segment to be a rectilinear line piece connecting two tiles in the same row or column. Global routing results are expressed in terms of segments. An L-shape for instance consists of two segments. The relation between segments and bends is simply

\[
\#\text{bends} = \#\text{segments} - 1.
\] (5.55)

In Fig. 5.19 the average segment distribution is shown. Due to effective placement, the large majority of wires are so short that they become short nets; these wires are not shown in the distribution. There are so many flat nets because many very short wires do cross tile boundaries and become flat nets. It is obvious from the distribution that the number

9If the segments have a layer assignment this is not accurate. The number of segments is still a good lower bound on the number of vias however, which is the purpose of \#\text{bends}.  

In Fig. [5.19], the average segment distribution is shown. Due to effective placement, the large majority of wires are so short that they become short nets; these wires are not shown in the distribution. There are so many flat nets because many very short wires do cross tile boundaries and become flat nets. It is obvious from the distribution that the number
of wires with more than two bends can be ignored. The average probability $\alpha$ is found by dividing the number of L-shapes excluding the small L-shapes by the sum of this number plus the number of Z-shapes. Table 5.3 shows the number of bends and $\alpha$ values for two-pin nets for all designs.

Table 5.3: Number of bends and $\alpha$ for all benchmarks

<table>
<thead>
<tr>
<th>chip</th>
<th>#bends&gt;2 [%]</th>
<th>$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0.53</td>
<td>0.60</td>
</tr>
<tr>
<td>b</td>
<td>0.042</td>
<td>0.67</td>
</tr>
<tr>
<td>c</td>
<td>2.0</td>
<td>0.56</td>
</tr>
<tr>
<td>d</td>
<td>3.7</td>
<td>0.50</td>
</tr>
<tr>
<td>e</td>
<td>0.19</td>
<td>0.61</td>
</tr>
<tr>
<td>f</td>
<td>0.69</td>
<td>0.65</td>
</tr>
<tr>
<td><strong>avg</strong></td>
<td><strong>1.20</strong></td>
<td><strong>0.60</strong></td>
</tr>
</tbody>
</table>

As expected, $\alpha$ varies from design to design. It can be seen that $0.50 \leq \alpha \leq 0.67$. There seems to be little correlation between the utilization of the chip and its $\alpha$ value. Although the benchmark suite is too small to draw definitive conclusions the results justify our choice of giving $\alpha$ a fixed value instead of making it a function of e.g. the utilization or estimated wire length.

The segment distribution shown in Fig. 5.19 is totally different from the segment distribution that is implicitly used in Lou's work and illustrated by Fig. 5.20. Obviously, routes with multiple bends have a much higher probability according to this distribution than in reality.

5.7.2 Estimation quality

The results of the prediction, global routing and detailed routing are shown for two chips in Fig. 5.21. Bright colors correspond to higher usage values. The congestion maps are all very much alike. The predictions seem a little *darker* than the other maps, i.e. in the
5.8 Summary and conclusions

Final routing congestion is a bit better distributed. This is exactly what was expected. Real routers can take other wires into account and thus spread the congestion better.

In Fig. 5.22 the distribution of errors over the map is shown on the left. All combinations of prediction, global routing and detailed routing are shown. On the right the error distribution as a percentage of the maximum usage is shown. The spreading of the errors appears to be noise-like. If both the probabilistic prediction and the global router make more or less the same error indicating that for some reason the detailed router has unpredictable behavior in that spot. Close inspection learns that there appears to be a small correlation between hotspots in the congestion map and hotspots in the error map. Congestion prediction methods tend to exaggerate, or rather to underestimate the creativity of the detailed router. The global router has more possibilities than the probabilistic method to avoid unrealistic high congestion values. Therefore, exactly in those hotspots there is also an error between the prediction and global routing.

For all three cases about 50% of the tiles have less than 2.5% error, indicating that the predictions are relatively accurate. Hardly any tile has an error over 20%. Note that large relative errors do not necessarily correspond to large absolute errors. The three distributions are very similar. Apparently Pce can be used to estimate global routing as well as detailed routing. Remarkably, global routing does not really appear to be more accurate than Pce.

Independent study

An independent study in [110] also shows that our method is indeed more accurate than Lou’s. Due to the simplicity of the approach the method is also about 25% faster.

5.8 Summary and conclusions

Based on the results of an industrial routing engine a method for probabilistic congestion estimation is developed. The method is an improvement over Lou’s method[84]. As in other methods, detour-freeness is assumed. Additionally it is assumed that a global router does not route with more than two bends. This assumption is backed by analysis of router behavior on industrial designs. In the proposed method each of the routes with two bends or less is assigned a probability based on this analysis. The usages due to each of these
Figure 5.21: Predicted usage (a), global routing (b), and final usage (c) for two chips.
Figure 5.22: Error maps and distributions.
routes are multiplied by these probabilities and result in *probabilistic usage maps*.

Experimental results show that the method produces accurate congestion maps in reasonable time. Congestion hotspots are identified correctly but they are somewhat exaggerated. Real routers spread congestion better than probabilistic methods because they have a more global view over routing resources and demand. Essentially, probabilistic congestion estimation as proposed in this chapter is much faster than real global routing but it is overly pessimistic. If congestion estimates are used as a kind of warning or in a feed-back loop this is not necessarily a bad thing and can in fact be beneficial.
Chapter 6

Congestion estimation by fast degenerate global routing

Probabilistic congestion estimation as discussed in this thesis has proven to be relatively efficient and accurate. Results of probabilistic congestion estimation tools such as described in [84, 69, 137, 110] and this thesis are typically compared against detailed routed end-results. This more or less implies that the published results are typically obtained on routable designs. In reality, during a design flow congestion estimators are more frequently run to explore the design space and are effectively run on many unroutable designs as well. Routability is such a fundamental and difficult problem that during the design flow measures have to be taken at many stages to ensure the design is eventually routable. Fast and accurate congestion estimation methods are necessary to prune off parts of the design space that most likely lead to unroutable designs. Essentially, there is a clear need to test and develop congestion estimation methods for unroutable designs as well as for routable designs.

Another issue dealt with in this chapter is that probabilistic congestion estimation is overly pessimistic. Designers are typically interested in the locations where congestion problems could potentially appear. In practice, probabilistic methods are especially inaccurate in exactly these areas. In this chapter a congestion estimation method that aims at accuracy in these regions is developed. The technique is based on techniques from global routing and is called degenerate global routing. A large number of tradeoffs in the design of the algorithm are discussed. By making different choices it is possible to either have a quick but relatively inaccurate estimate, or spend a bit more time and obtain more accurate congestion maps. In this chapter the aim is to show that it is possible to compete with probabilistic methods on run time. The method is more accurate than Pce and perhaps surprisingly, requires only little more run time.

6.1 Objectives for fast degenerate global routing

Both probabilistic methods and methods based on global routing are congestion estimation methods and therefore have the same main objective as discussed in Section 3.2.
Essentially, a method that accurately predicts hot spots at least an order of magnitude faster than global routing is required. Additionally, the methods proposed in this chapter improve upon the main weakness of probabilistic congestion estimation: congestion pessimism. Another important issue that is discussed are unroutable designs. During exploration stages many unroutable designs may be considered. In order to take routing considerations into account congestion estimation should be employed. Previous methods were only evaluated against routable designs, but discarding *unroutable designs* is at least as important.

The congestion estimation tool developed in this chapter is essentially a global router with an entirely different set of priorities, settings and tradeoffs than real routers. Therefore, the estimator is often discussed as a global router. If the estimator performs reasonably well on global routing criteria it is likely to yield a reasonably accurate congestion estimate. In the end, the resulting tool is compared against an actual global router.

### 6.1.1 Suitability of global routing for congestion estimation

Global routing was developed as the first step of the routing stage and that is still its main task. When congestion became an issue as a result of increasing design sizes, densities and changes in manufacturing technology, global routing also became a tool for congestion assessment. Although the base algorithms underlying global routers can be very fast, all kinds of heuristics have been added in order to improve the quality of results. Especially in a physical synthesis flow the router is often fully integrated with e.g. the timer and logic synthesis algorithms. Run times of global routing can be considerable as a result. Nonetheless, in this chapter an algorithm for fast congestion estimation based on global routing is proposed. The main reason why this is possible is illustrated by Fig. 6.1 and formulated as follows.

**Observation 6.1.** The objectives of global routing and congestion estimation are somewhat perpendicular. Essentially, the task of global routing is to optimize the congestion map within *reasonable run time* while the task of congestion estimation is to optimize run time for finding a *reasonable congestion map*. Most global routing algorithms use refinement techniques to improve upon an existing congestion map. Creating the initial congestion map can be relatively cheap while improving it is increasingly more expensive.

![Figure 6.1](image.png)

**Figure 6.1:** An initial routing solution can usually be found in a small amount of time. Refining the solution as performed by global routing takes an increasing amount of time.
In essence, the problem with using global routers as congestion estimators is that they are tweaked towards overflow elimination. Run time is of less importance. One example of a computationally expensive method is rip-up and reroute (refer to Chapter 3.4.1). It may be easy to switch this off for most routers, but global routers and their infrastructure are tuned towards congestion removal and it is generally not possible to remove or circumvent all the overhead associated with this. Additionally, a different class of heuristics is needed for speedup and this is a direction of research that has not been explored much. First, the heuristics that improve the solution only little at the cost of lots of run time need to be removed. Next, new heuristics that speed up the global router considerably at relatively little solution degradation need to be added. Different global routing algorithms yield different results. Since the proposed congestion estimator is based on maze routing it will at least partially employ the same tricks to avoid overflow. Therefore it is expected that the quality of the predicted congestion maps is better than those produced by a probabilistic method.

6.1.2 Weaknesses of probabilistic methods

Analysis of the method as discussed in this thesis but also in e.g. [110] indicates that probabilistic methods are pessimistic: overflow in a probabilistic congestion map does not necessarily correspond to unroutability\(^1\). This means essentially that probabilistic congestion maps are suitable for algorithms that try to spread congestion such as [112], but less suitable for algorithms that use congestion as constraints. Well spread congestion will solve most routability issues but this may happen at the expense of other metrics. A problem for human designers assessing e.g. floorplans is that it is difficult if not impossible to judge whether the resulting probabilistic congestion maps depict a feasible design or that measures must be taken in order to improve routability.

In this chapter a global routing algorithm that is tuned for speed is discussed. As a global router it performs poorly: it would be easy to improve the solution for metrics such as overflow. Compared to probabilistic methods however the resulting congestion maps are more accurate. For routability the congestion hot spots are of most importance. Compared to other approaches, the proposed algorithm performs especially well at exactly these locations.

6.1.3 Metrics

Congestion maps are used to base decisions on during floorplanning. Visual inspection is important and congestion maps and predictions are compared qualitatively. Error maps show how errors are distributed over the chip. They are used to visualize peak values and see if there is a correlation with congestion.

For the routability of a chip the areas with high congestion values are problematic. Edges in the routing graph with congestion levels around 1 and higher deserve extra attention and algorithms and designers focus on these edges. Therefore, let us introduce the notion of wrongly congested edges. These are edges that are considered to be over-congested by a congestion estimator, while in reality they are not. This obviously depends on the router that is used, but we speculate that the differences between routers in quality of results is less than the differences between routing results and congestion estima-

---

\(^1\)The opposite however is almost guaranteed to be true.
tion results. In other words, these edges are *wrongly flagged to be congested*. Because of conservative resource estimation and because congestion estimation methods are always expected to exaggerate, the threshold is set at 1.1. Let \( W_C \) denote the set of wrongly congested edges, \( c(r, c) \) a congestion estimate, and \( C(r, c) \) the actual congestion value of an edge \( e(r, c) \). Then we define

**Definition 6.1** (Wrongly congested edges).

\[
e(r, c) \in W_C \quad \text{if} \quad c(r, c) > 1.1 \land C(r, c) \leq 1.1.
\] (6.1)

Equivalently, the set of *wrongly uncongested* edges \( W_U \) is defined as follows.

**Definition 6.2** (Wrongly uncongested edges).

\[
e(r, c) \in W_U \quad \text{if} \quad c(r, c) < 0.9 \land C(r, c) \geq 0.9.
\] (6.2)

Estimation results will be evaluated by the cardinality of the sets \( W_C \) and \( W_U \). If \( W_C \) is too large a designer or optimization algorithm will wrongly decide that a design is infeasible while in reality it may be routable. The distribution of these edges over the routing area is also important. This aspect is addressed by inspection of the congestion and error maps.

### 6.2 Previous work

There is a lot of work on global routing, an excellent survey is [57]. An overview of more recent advances is given in [90]. However, there is not a lot of literature on the application of global routing techniques to fast congestion estimation.

Algorithms used in floorplanners are typically not suitable for stand-alone evaluation of large flat placements. The estimators in floorplanners rely on a channel-based model of the routing region that is not valid for the general case. Nonetheless, interesting work has been performed in this field. Chen *et al.* use a fast router in a floorplanner to take congestion into account[21]. Because it is used in an inner-loop of the floorplanning algorithm speed is obviously very important and they restrict the router to search only for L- and Z-shaped routes. In [114] Shen and Chu interpret the global routing problem as a flow problem of several commodities. Since the resulting congestion estimate is used in simulated annealing-based floorplanning, the integral flow constraint can be relaxed such that no actual global routing solution results. This is somewhat comparable to probabilistic congestion estimation. A global routing solution can be obtained by rounding techniques. The algorithm slows the floorplanner down by a factor of 2.7 on average.

In [97] Parakh, Brown and Sakallah demonstrate a quadratic placer with an A*-based routing algorithm under the hood. The routing algorithm is not congestion-aware and thus very inaccurate, again somewhat comparable to probabilistic estimation. Further, the tile model is quite coarse and only relatively small benchmarks are used.

Chang and Cong demonstrate a congestion-aware placement algorithm in [19]. Congestion is analyzed by an LZ router that only considers L- and Z-shapes. The congestion analysis slows the algorithm down by a factor of almost 25. Since placement algorithms change the locations of the cells (pins), net topology needs to be changed routinely. The
6.3 Fast degenerate global routing

Many global routers including our own global router (which is discussed in Chapter 7) are based on sequential routing of nets or wires. For each wire a minimum-cost path is found using shortest path algorithms such as Dijkstra\[^{32}\] or A*\[^{92}\]. Typically the cost is determined by a combination of the length of the path and the amount of congestion on the path. The shortest path algorithms that are used are optimal for individual wires but the global optimum for the whole set of wires (in terms of wire length or in terms of congestion) cannot be guaranteed due to the wire ordering problem. Rip-up and reroute methods (R&R) are used in order to mitigate this problem. The use of cost functions and R&R take up a lot of run time. The basic approach is outlined in Fig. 6.2 although more sophisticated methods to perform R&R are used in practice.

The proposed approach for congestion estimation is also based on the procedure outlined above. Within this framework, algorithms and heuristics focusing on speedup are implemented. The resulting tool is called FaDGloR\[^{2}\]. The methods that are used are highly heuristic and although the results can be explained largely, fine tuning these parameters is perhaps more an art than a science. In this section these heuristics are discussed. Note that the tool is effective as a result of the interplay between the different heuristics more than each of the heuristics individually.

\[^{2}\]FaDGloR stands for FAst Degenerate GLObal Router.
6.3.1 Degenerate routing graph

Global routing can be seen as a problem defined on a three dimensional grid graph. In many routers this is reduced to a two dimensional problem by the introduction of layer assignment. Since speed is of utmost importance in congestion estimation our estimator also works on a two dimensional routing graph. This graph is created from a three dimensional grid graph by collapsing the edges connecting the different layers and merging capacities of parallel edges. This is called a degenerate routing model.

Using a two dimensional model instead of a three dimensional model reduces the search space for the shortest-path algorithms dramatically. In modern technologies up to nine routing layers are available. Roughly, a 9X reduction in number of nodes and a 4.5X reduction in number of edges in the layers is achieved\(^3\). The between layer edges disappear altogether. In a simplified model\(^4\) the reduction is approximated by the following formulas.

\[
|V_{3-D}| = w \cdot h \cdot (|L_{hor}| + |L_{ver}|) \quad (6.3)
\]

\[
|E_{3-D}| = h \cdot (w - 1) \cdot |L_{hor}| \quad \text{horizontal}
\]

\[
+ w \cdot (h - 1) \cdot |L_{ver}| \quad \text{vertical}
\]

\[
+ w \cdot h \cdot (|L_{hor}| + |L_{ver}| - 1) \quad \text{between layers} \quad (6.5)
\]

\[
|V_{2-D}| = w \cdot h \quad (6.7)
\]

\[
|E_{2-D}| = h \cdot (w - 1) + w \cdot (h - 1), \quad (6.8)
\]

where \(V_{3-D}, E_{3-D}, V_{2-D} \) and \(E_{2-D} \) represent the nodes and edges in the three-dimensional and two-dimensional models, respectively, \(w \) and \(h \) are the width and height of the tile grid, and \(L_{hor} \) and \(L_{ver} \) the sets of horizontal and vertical routing layers.

6.3.2 Shortest-path algorithms and the choice for A* 

The algorithm outlined in Fig. 6.2 indicates that the run time of LOWEST-COST-PATH is crucial for the run time for the algorithm as a whole. The edges in the global routing graph have an associated cost. These costs are fixed and non-negative during the search for a path and are updated only after a path has been found. Therefore, the problem as posed to LOWEST-COST-PATH is a classical shortest path problem.

In the literature the problem for finding a minimum-length (minimum-cost) path between two nodes in a graph is known as the single-pair shortest-path problem\(^{[32]}\). This problem can be considered a specialization of the single-source shortest-path problem in which a shortest path to any node in the graph is searched for. Although no algorithm that

\(^{3}\)A strict preferred direction routing style is assumed with equal number of horizontal and vertical layers.

\(^{4}\)This model assumes the same wire pitch on all layers which is not generally the case, although it is known to be used in some mature technologies.
runs asymptotically faster than the latter problem is known for the former problem, for practical cases algorithms that run much faster are known.

Most efficient shortest-path algorithms are so-called best-first search algorithms as outlined in Fig. 6.3. In this algorithm, $G$ is the graph with $s$ and $t$ the source and target nodes, respectively. $c$ is the cost function returning the cost of an edge between two nodes and $h$ is a heuristic function that compares two nodes and returns which one is most attractive based on some criterion. $Q$ is a priority queue usually implemented as a binary heap although e.g. Fibonacci or Binomial heaps[32] are also possible. $d[v]$ represents the distance (or cost) between the source node and a node $v$, and $\pi[v]$ is the predecessor of $v$. When the algorithm terminates the found path can be traced back from the target node $t$ through the predecessors and the length (cost) of the path is $d[t]$.

```
BEST-FIRST-SEARCH($G, c, h, s, t$)
1 for each node $v \in V[G]$ do $d[v] \leftarrow \infty$ // Initialize distances (costs)
2 $f[v] \leftarrow \infty$ // Initialize cost estimate
3 $\pi[v] \leftarrow \text{NIL}$ // Initialize predecessors
4 $d[s] \leftarrow 0$
5 $f[s] \leftarrow h(s, t)$
6 $Q \leftarrow \{s\}$
7 $u \leftarrow \text{EXTRACT-MIN}(Q, f)$
8 while $u \neq t$ do for each node $v \in \text{Adj}[u]$ do if $d[v] > d[u] + c(u, v)$ then $d[v] \leftarrow d[u] + c(u, v)$
9 $f[v] \leftarrow d[v] + h(v, t)$
10 if $\pi[v] = \text{NIL}$ then $Q \leftarrow Q \cup v$
11 $\pi[v] \leftarrow u$
12 $u \leftarrow \text{EXTRACT-MIN}(Q, h)$
13 return $t$
```

**Figure 6.3:** A generic best-first search algorithm.

Essentially, the algorithm starts by visiting the source node. The priority queue contains all visited nodes. At each iteration the most promising node in the queue (as determined by the heuristic function) is expanded: the minimum distance from the source of all its neighbors is updated for the path through this node and the neighbors are added to the queue. When the most promising node is the target node the algorithm terminates.

**Dijkstra’s algorithm**

The algorithm by Dijkstra[34,32] is perhaps the best-known shortest-path algorithm. In its original formulation it is an algorithm for the single-source shortest-path problem: it
has no target node but finds the shortest-path to any node in the graph. In a sense, it is an undirected search.

Dijkstra’s algorithm uses $d$ (the distance from the source node) directly in the comparison function, i.e. $h(v, t) = 0$. Thus, the EXTRACT-MIN operation returns nodes closest to the source. The use of this heuristic function is the signature of a Dijkstra-style search algorithm and makes Dijkstra’s algorithm essentially a breadth-first search\cite{32}. As a result of this strategy Dijkstra is guaranteed to yield the shortest path correctly.

The run time complexity of Dijkstra’s algorithm depends on the data structures that are used. In a trivial implementation where the priority queue is an unsorted array, the run time is $O(|V|^2)$. Most implementations use a binary heap\cite{32} instead yielding $O(|E|\log|V|)$. With advanced data structures such as Fibonacci heaps\cite{32} an amortized run time complexity of $O(|V|\log|V| + |E|)$ is obtained. In the case of global routing graphs such as discussed in this thesis, $O(|E|) = O(|V|)$ and both heaps yield complexities of $O(|V|\log|V|)$. Binary heaps are easily implemented on top of simple data structures such as arrays or vectors\cite{123} and the constants are in practice much lower than the constants for a Fibonacci heap.

### 6.3.3 The choice for $A^*$

Dijkstra is an undirected search algorithm. It is possible to visit and expand less nodes in the routing graph based on the following observation.

**Observation 6.2.** The routing graph is a grid graph. The distance between two nodes can therefore be calculated exactly. This means we can calculate a relatively accurate and optimistic estimate of the cost between the nodes. This estimate is exact in absence of congestion.

**Algorithms A and $A^*$**

Algorithm A is a graph search algorithm developed in the artificial intelligence community\cite{92}. The algorithm is a so-called informed-search algorithm: it uses knowledge about the target node. The heuristic function $h(v, t)$ is typically not zero: it is an estimate of the length (cost) of the path between $v$ and $t$ such that $f[v]$ becomes an estimate of the minimum-cost between $s$ and $t$ through $v$. If this heuristic is optimistic, i.e. $h(v, t) \leq h^*(v, t)$, where $h^*(v, t)$ is the actual lowest cost between $v$ and $t$, the algorithm is said to be admissible and is guaranteed to return a shortest path correctly. If this is the case, algorithm A is called $A^*$. In this thesis this property of the algorithm will be referred to as the optimality of $A^*$.

The worst case run time complexity for $A^*$ is no better than for Dijkstra’s algorithm. Under very mild conditions it has been shown that $A^*$ will never expand more nodes than Dijkstra’s algorithm\cite{92}. Essentially, as long as $h_{A^*} > 0 A^*$ is a better algorithm. This condition implies that a non-zero estimate of the cost between a node and the target node must be used. Since the cost for routers at least partially consists of wire length this condition is easily met. Because it uses information about the target node, $A^*$ is said to be more informed than Dijkstra. The run time of $A^*$ depends on the heuristic function. It has been shown in \cite{106} that if $|h(v, t) - h^*(v, t)| \leq O(\log h^*(v, t))$ the number of nodes that is expanded is polynomial in the length of the shortest path.
According to Observation 6.2 and [92], A* typically expands less nodes than Dijkstra's algorithm if the distance between nodes in the grid graph is properly used. In particular we note the following.

**Observation 6.3.** If A* uses the distance between nodes in the grid graph in the heuristic function it essentially becomes a depth-first algorithm aimed at the target node.

In experiments it was found that A* performs much better than Dijkstra’s algorithm on the benchmarks used throughout this thesis.

### 6.3.4 Rip-up and reroute

R&R can be regarded as a way to refine global routing solutions. Essentially, the situation before ripping a wire (removing it) is an accurate congestion estimate of the situation after rerouting it. In FaDGloR the focus is on run time and R&R is found to be too time consuming.

During global routing most time is spend in the R&R phase [57]. Many wires are routed multiple times (in the global router discussed in Chapter 7 typically 9 times in total). Finding a path for a single wire is usually faster during R&R than it is during initial routing because of the ripping strategy. If a single wire is ripped and rerouted, ripping usually leaves a trench in the cost landscape. Thus, rerouting will often yield (partially) the same path and many neighboring nodes are not expanded because of their higher costs. In the case of ripping a whole region at once the search space is in practice almost confined to the ripped region that has much lower cost than the surrounding (congested) regions. Although R&R does improve the result this is relatively expensive in terms of run time.

Common R&R strategies in global routing can be characterized as erosion [51]. Initially a lot of overflow is allowed, yielding high peaks in the congestion landscape. During R&R these peaks are redistributed to the surrounding (less congested) area. Initially overflow is allowed because initially detours are undesirable due to the net ordering problem. After initial routing a relatively accurate congestion estimate is available and wires are rerouted through less congested regions. During the first phases of R&R the focus is mainly on finding alternative detour-free paths. In later stages the focus shifts to removing overflow, typically at the cost of detours. This behavior is obtained by changing the cost model during R&R. Initially primarily wire length is penalized and this gradually changes to larger congestion costs.

**R&R in FaDGloR**

Compared to global routers, FaDGloR puts a higher emphasis on speed. By disabling R&R in a global router a simple congestion estimator can be obtained. However, R&R strategies are based on the assumption R&R actually takes place. Thus, when only initial routing takes place this is “optimized” for subsequent R&R and not to be an accurate representation of routing resources and demands.

During experimentation it was found that R&R is a very time consuming way to improve routing results and FaDGloR does not employ it. Instead, a two-phase strategy to generate the best possible results is used. The result is not necessarily a good starting point for subsequent R&R but compared to probabilistic estimation, results are more accurate since the underlying A* algorithm is allowed to detour.
Two-Phase-Routing\((G, W)\)

\[
\begin{align*}
1 & \quad U \leftarrow \text{MAKE-SET}() \\
2 & \quad \text{for each } w \in W \\
3 & \quad \quad \text{do } p \leftarrow \text{FIND-MINIMUMLENGTHPATH-NOOVERFLOW}(G, w) \\
4 & \quad \quad \text{if } p = \text{NIL} \\
5 & \quad \quad \quad \text{then } U \leftarrow U \cup w \\
6 & \quad \quad \text{else } \text{SET-PATH}(w, p) \\
7 & \quad \quad \text{UPDATEGRAPH}(G, p) \\
8 & \quad \quad \triangleright \text{Second phase} \\
9 & \quad \quad \text{for each } w \in U \\
10 & \quad \quad \quad p \leftarrow \text{FIND-MINIMUMOVERFLOWPATH}(G, w) \\
11 & \quad \quad \quad \text{SET-PATH}(w, p) \\
12 & \quad \quad \text{UPDATEGRAPH}(G, p)
\end{align*}
\]

**Figure 6.4:** The Two-Phase-Routing algorithm.

### 6.3.5 Two-phase strategy

Due to conservative resource modeling, capacity constraints on edges in the global routing graph are not always hard. However, the main focus of global routing is to remove overflow. In FaDGloR, we try to achieve this by simply *not allowing overflow* initially. Obviously, this may lead to unroutable wires. These wires are routed in a second phase where $A^*$ routes for minimum overflow.

The reason why wires are not directly routed for minimum overflow after failing overflow-free routing is the following. Many routes with equal minimum overflow may exist. Some of these routes may block wires routed subsequently. It was found empirically that it is better to postpone routing for minimum overflow until all wire have been tried by the overflow-free phase.

The above approach is outlined in Fig. 6.4, where $W$ denotes the set of wires and $G$ the global routing graph. In the first phase each wire is routed using a minimum length path that causes no overflow. If no such path is found, routing the wire is postponed until the second phase. During this phase the remaining wires are routed using a minimum overflow objective.

### 6.3.6 Wire order

An issue that turned out to be important for the performance of the two-phase strategy is the order in which the wires are routed. Let us first introduce some terminology. By routed or unrouted *distance* of a wire the manhattan distance between its two pins is denoted. By the routed *length* of a wire the actual length after routing is meant.

As mentioned before, the difficulty of global routing is in the fact that different wires compete for the same routing resources. In a sequential approach such as ours wire order is obviously of influence: wires that are routed first get first choice. Wires that are routed later are more likely to fail overflow-free routing. In a global router R&R is used as a way to
mitigate the impact of the *wire ordering problem*. Since no R&R is performed in FaDG1or the influence of wire ordering is higher. It was found empirically that the best solution is to use *shortest wires first*.

**Empirical observations**

In order to find an optimal wire ordering strategy, *shortest wires first* and *longest wires first* strategies were tried in our experiments. The *unrouted distance* was measured after the first phase and the *overflow* after the second phase.

Remarkably, *longest wires first* yields the best results in terms of unrouted distance while *shortest wires first* yields the best results in terms of overflow. These results are somewhat counter-intuitive since one might expect that one approach would perform best on both metrics.

When the longest wires first wire order is used the set of wires that is routed in the second phase consists mainly of short wires, e.g. of length 1. In the most extreme case when all unrouted wires have this length the minimum overflow that can be obtained in the second phase is equal to the unrouted distance. In the opposite case (shortest wires first) the set of unrouted wires consists mainly of very long wires. Evidently, large parts of these wires can be routed without overflow. Hence the overflow will be far less than the unrouted distance.

In most global routers based on sequential techniques shortest wires first is used because the results in terms of overflow tend to be better. This is also observed in the two-phase strategy. Longer wires tend to have more detour-free realizations and routers are more likely to find a detour-free path when congestion emerges. Additionally, congestion starts to play a role only after a certain amount of distance has been routed. If the remaining wires are mainly short, relatively many pins are involved. If these pins are in congested regions this leads inevitably to overflow. Less pins (long wires last) suffers less from this problem.

**6.3.7 Cost function**

Shortest path algorithms need a distance (cost) metric. In FaDG1or with its emphasis on run time a very simple cost function is used.

Global routers use cost functions as shown in Fig. 6.5 to map a congestion level to a cost. Essentially, instead of a shortest path a lowest cost path needs to be found. The purpose of non-zero costs before edge capacity has been reached is to guide wires to less-congested parts of the routing graph. This effect has a cost in terms of run time because of the following.

1. Distance-based cost estimates become a looser bound on the actual cost. In other words: A* becomes *less informed*.

2. Equal-length paths may have different associated costs. More paths with different costs means A* needs to explore more.

Both factors result in more nodes being visited by A*. If for example A* has expanded to almost the target node, making the last step can take a long time. As long as a potentially better path exists A* will examine that path first.
Experiments with cost functions have been conducted. It turned out that for congestion estimation it was best \textit{not to use cost functions at all}. More precisely, congestion is ignored and the cost of an edge is always 1. Overflow is more effectively handled by the two-phase strategy.

### 6.3.8 Detour bounding

FaDGloR reduces the search space for the shortest path algorithm by bounding the allowed detour to a maximum. The smaller the allowed detour, the more the congestion picture will look like the outcome of a probabilistic method. Allowing some detour gives A* the opportunity to remove overflow at the cost of run time (and routing resources that are not available for subsequent wires). We observe that a very large part of the run time of global routers is spend on a relative few number of (long) problematic nets. Essentially, by bounding the detour the router is instructed to more quickly accept overflow in these cases.

During experiments it was observed that run time grows almost exponentially with allowed detour. For our benchmark set, it was found that 12 is a good detour bound. This is obviously somewhat arbitrary. Smaller bounds are also effective but larger detours are not generally usable since then run time explodes.

### 6.4 Experimental results

The methods presented in this chapter are highly heuristic in nature and their merit needs to be verified by experiments. The Labyrinth benchmark suite that was also used in Chapter 4 is used and summarized in Table 6.1. The widely-used global router Labyrinth is used to compare the FaDGloR results against. It would also have been possible to use our own in-house global router Grawet (described in Chapter 7) but since this router shares partially the same engine as FaDGloR this does not seem to be a fair comparison. Labyrinth is based on \textit{pattern routing}, a different technology than maze/A*-based routing. Congestion estimation techniques should also be able to predict the behavior of routers based on other techniques accurately. Furthermore, Labyrinth appears to be the most common reference in papers on congestion estimation or global routing. Recently global routing has gained a lot of new interest and new academic routers have become publicly available \cite{90}.

\footnote{Unfortunately the experiments presented in this thesis had already been performed at that point in time.}
6.4 Experimental results

There were some issues with the benchmarks. They were discussed with the creators of the benchmarks and this led to an improved version of the benchmark suite. Benchmark ibm05 has a very low utilization and is not very representative. In papers such as [51] for example the same benchmark suite is used but ibm05 is left out.

### 6.4.1 Varying capacity

One of the objectives of this chapter is to study the performance of different congestion estimation methods on designs of varying difficulty. In total 16 different benchmark suites were created based on the original benchmark suite. Each of these suites has a different difficulty. They were created by changing the capacity of the edges in the routing graph. The capacities in the original suite were adjusted by -5 through +10. The suite with 5 subtracted yielded entirely unroutable benchmarks while the suite with 10 added yielded designs that were very easy to route. Entirely infeasible designs may be encountered during relative early stages of the design. Very easy blocks may be encountered in a hierarchical flow where the available routing resources have been determined by the needs of other blocks.

Table 6.2 characterizes the difficulty of the designs by showing how much additional capacity was necessary to make them routable for Labyrinth. Design ibm05 was routable in all suites because of its low utilization.

<table>
<thead>
<tr>
<th>design</th>
<th>grid</th>
<th>nets</th>
<th>wires</th>
<th>design</th>
<th>grid</th>
<th>nets</th>
<th>wires</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>64 × 64</td>
<td>12k</td>
<td>27k</td>
<td>ibm06</td>
<td>128 × 64</td>
<td>33k</td>
<td>79k</td>
</tr>
<tr>
<td>ibm02</td>
<td>80 × 64</td>
<td>18k</td>
<td>53k</td>
<td>ibm07</td>
<td>192 × 64</td>
<td>44k</td>
<td>105k</td>
</tr>
<tr>
<td>ibm03</td>
<td>80 × 64</td>
<td>22k</td>
<td>44k</td>
<td>ibm08</td>
<td>192 × 64</td>
<td>48k</td>
<td>128k</td>
</tr>
<tr>
<td>ibm04</td>
<td>96 × 64</td>
<td>26k</td>
<td>52k</td>
<td>ibm09</td>
<td>256 × 64</td>
<td>50k</td>
<td>124k</td>
</tr>
<tr>
<td>ibm05</td>
<td>128 × 64</td>
<td>28k</td>
<td>90k</td>
<td>ibm10</td>
<td>256 × 64</td>
<td>64k</td>
<td>175k</td>
</tr>
</tbody>
</table>

### 6.4.2 Visual inspection

Fig. 6.6 shows example horizontal usage maps produced by the tools Pce (refer to Chapter 5), FaDGloR and Labyrinth for two designs. Brighter colors correspond to higher usage. Globally, the maps produced by the three tools have the same characteristics. The two prediction tools both predict the locations of the hot spots accurately. The main difference
Figure 6.6: Usage maps for (a) Pce, (b) FaDGloR, and (c) Labyrinth.
between the three tools is the size of the hot spots. As expected, the global router is capable of spreading the congestion better than the two predictors, leading to larger but less severe hot spots. The probabilistic method has small hot spots with higher congestion values: the pessimism probabilistic methods are known to have. The congestion maps produced by FaDGloR are in between the maps produced by the probabilistic method and the global router, respectively. Evidently, the quality of the estimate is therefore higher than the quality of the estimate produced by the probabilistic method. These observations are valid for the whole benchmark suite.

![Error maps for (a) Pce, and (b) FaDGloR.](image)

**Figure 6.7:** Error maps for (a) Pce, and (b) FaDGloR.
6.4.3 Error maps

Let $C(r, c)$ denote the actual congestion value as found by a global router for an edge with coordinates $(r, c)$, and let $c(r, c)$ denote the congestion value as predicted by an estimator. In Fig. 6.7 two error maps are shown for each of the two chips shown in Fig. 6.6. One for Pce and one for FaDGloR. They show how the errors $|c(r, c) - C(r, c)|$ are distributed over the routing area.

In both error maps the peak error produced by Pce is much higher than the peak error produced by FaDGloR. The highest errors for Pce are found in and around congested areas. Unfortunately, these areas are the most important ones when it comes to routability. The errors for FaDGloR are spread over the routing area more in a noise-like fashion. This confirms that FaDGloR also has inaccuracies but less than Pce and more randomly distributed. A random distribution is better because it is known that small congestion spots are easily solved by global routers. Therefore tiny congestion hotspots can safely be ignored and it does not really matter if such spots are an artifact of congestion estimation.

![Graph showing number of wrongly congested edges](image1)

![Graph showing number of wrongly congested edges](image2)

**Figure 6.8:** The number of wrongly congested edges for unroutable designs (top) and for routable designs (bottom).

6.4.4 Wrongly congested and wrongly uncongested edges

Fig. 6.8 shows the number of wrongly congested edges for both estimators for all designs. The top picture shows the numbers for the benchmark suite without extra capacity. These benchmarks (except ibm05) are all unroutable. The bottom picture shows the same numbers, but for the benchmark suite with 4 added to the capacity. Here, all benchmarks are
routable (refer to Table 6.2). It can be seen that FaDGloR reduces the number of wrongly congested edges by roughly 50% compared to Pce for the unroutable benchmarks. For the routable designs the reduction is even greater: up to 100% for some benchmarks. These reductions are huge and show that FaDGloR suffers less from congestion exaggeration than Pce. Especially for routable designs this is the case, which is important since based on the congestion estimate of Pce, designers may erroneously discard implementation directions for routability reasons.

Fig. 6.9 shows the number of wrongly uncongested edges produced by both tools. In this case FaDGloR performs only slightly better than Pce. The difference on this metric is not as great as on the number of wrongly congested edges. Wrongly uncongested edges are mainly found in relatively easily routable regions, near the periphery of really congested areas. In such regions usage can be moved around without impacting any of the important criteria. Labyrinth makes different choices here than FaDGloR and this leads to a relatively bad prediction in those areas. Additionally, wrongly uncongested edges are the result of trading overflow for detour. Labyrinth does this much more than FaDGloR and Pce and this results in many wrongly uncongested edges around the congested regions. This makes the relative improvement of FaDGloR over Pce small.

Fig. 6.10 shows how the number of wrongly congested edges typically varies with capacity. With increasing capacity all three tools should label less edges congested. Also the number of edges that are wrongly marked congested should therefore decrease although this is not always the case because of apparently random effects. FaDGloR evidently performs much better for all capacities. Pce even labels some edges as over-congested when
there is abundant capacity. FaDGloR suffers much less from this effect. Because it does not use cost functions or rip-up and reroute it needs a little more capacity than a real global router to achieve an over-congestion free solution.

Table 6.3 shows the results for all benchmarks in all benchmarks suites. Improvements of FaDGloR over Pce are shown as a percentage. The column and row marked with “A” give the average. Over all benchmarks, FaDGloR is about 65% better on this metric than Pce. It can be seen that especially on routable and nearly routable designs FaDGloR produces superior results.

### Table 6.3: Improvements of FaDGloR over Pce on #wrongly congested edges as a percentage.

| ibm | -5 | -4 | -3 | -2 | -1 | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | A  |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 01  | -29| 25 | 25 | 20 | 23 | 36 | 42 | 47 | 55 | 79 | 88 | 87 | 100| 100| 100| 100| 56 |
| 02  | -2 | 15 | 15 | 23 | 31 | 43 | 54 | 69 | 77 | 80 | 83 | 100| 100| 100| 100| 100| 62 |
| 03  | 7  | 19 | 34 | 56 | 73 | 94 | 96 | 99 | 100| 100| 100| 100| 100| 100| 100| 100| 80 |
| 04  | -12| 5  | 46 | 50 | 59 | 59 | 51 | 55 | 57 | 56 | 58 | 58 | 85 | 100| 100| 100| 58 |
| 05  | 100| 100| 100| 100| 100| 100| 100| 100| 100| 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 63 |
| 06  | 2  | 9  | 31 | 37 | 47 | 57 | 73 | 87 | 89 | 95 | 100| 100| 100| 100| 100| 100| 70 |
| 07  | 29 | 26 | 32 | 43 | 54 | 61 | 73 | 79 | 87 | 100| 100| 100| 100| 100| 100| 100| 74 |
| 08  | 7  | 15 | 21 | 42 | 47 | 59 | 62 | 72 | 83 | 93 | 100| 100| 100| 100| 100| 100| 69 |
| 09  | -1 | 9  | 11 | 17 | 30 | 40 | 44 | 63 | 81 | 93 | 100| 100| 100| 100| 100| 100| 62 |
| 10  | 3  | 5  | 6  | 14 | 18 | 49 | 58 | 68 | 75 | 83 | 83 | 93 | 94 | 100| 100| 100| 59 |
| A   | 10 | 23 | 32 | 40 | 48 | 60 | 65 | 74 | 80 | 88 | 81 | 84 | 88 | 90 | 90 | 90 | 65 |

### 6.4.5 Run time

The target of congestion estimation is to be much faster than global routing. Fig. 6.11 shows that both estimation tools are roughly an order of magnitude faster than Labyrinth as required by the Congestion Estimation Problem (3.1). As expected, run times increase with problem size. The run times of the two routers also vary with difficulty, so for a fair
comparison capacity was added until a design was routable. FaDGloR is slightly slower than Pce but the difference is only marginal.

The proposed degenerate global routing algorithm is much more complicated than the probabilistic approach. Therefore it is perhaps surprising that the run time advantage of Pce is so small. In order to explain this let us consider a wire spanning an $m \times n$ box. For Pce, a stamp consisting of $m \times n$ elements needs to be copied into the usage map. For FaDGloR much depends on the congestion in and around the wire box. For the large majority of wires congestion is not important. In that case the A* algorithm only needs to visit $O(m + n)$ nodes since it behaves like a depth-first search. Also, the nodes that are added to the heap do not require many compares since nodes closer to the destination are preferred over nodes further away\(^6\). When congestion starts to play a role A* needs to visit more nodes. Eventually Pce is a bit faster than FaDGloR, but not as much as perhaps expected.

The shortest wires first scheme is a large contributor to the success in terms of run time. If the pins of the wires are more or less evenly distributed over the area congestion starts to play a role after a certain amount of routed distance. Only a few long wires are left at this point in the case of shortest wires first. Each of these wires has a detour bound of 12, thus limiting the search space of A*. In the opposite situation (longest wires first) many more shorter wires would be left with each an allowed detour of 12. The total amount of allowed detour would be much higher in this case. Congestion in combination with more allowed detour leads to more node visits and eventually more run time.

The run times of both routers depends on the difficulty of the design. Both Labyrinth and FaDGloR need to consider increasingly more paths with decreasing capacity. Additionally, Labyrinth spends more time on R&R if there is overflow. Fig. 6.12 shows how the run times of the three tools typically vary with capacity. For this particular design FaDGloR is for the easiest designs even faster than Pce but this is not always the case. For most designs the run time of FaDGloR decreases with increasing capacity but is higher than the run time of Pce, even for the easiest designs.

Since we do not have access to all other tools mentioned in the literature the run times of our implementations of the congestion estimation algorithms discussed in this thesis are compared with data from the literature. The implementations of Pce and FaDGloR

\(6\) This is a detail of how A* was implemented.
labeled with “I” are run on a 1GHz Linux server and were compiled with GCC 3.2.2, and the implementations labeled with “II” are run on a 1.4GHz laptop running Linux and were compiled with GCC 3.3.2. Since the run times of FaDGloR depend much on the available capacity and many of our benchmarks are unroutable, we also report the run times of the “first routable” benchmarks, i.e. capacity was added to each design until it was routable. The run time numbers as reported in the literature are usually obtained from routable designs, so this is perhaps a more fair comparison.

Although a direct comparison is not possible, our implementations appear to be much faster than the other tools. This is very important since run time determines how practical a method is.

### 6.5 Discussion

There are two metrics by which the quality of a congestion estimator should be evaluated. Firstly, the quality of the estimates should be usable for routability analysis. Secondly, the run time should be much lower than the run time for global routing. In this chapter a congestion estimator based on global routing techniques is developed. Experimental results on a large benchmark suite show that the quality of results of the degenerate global router is superior to the quality of results of the probabilistic method developed in the previous chapter. The method is especially better at predicting the usage in the edges that are cru-
cial for routability. Both tools have been evaluated on benchmarks with varying difficulty and the routing-based approach is superior on all difficulties. Both methods predict routing difficulties where there are in fact none, but the probabilistic method suffers much more from this effect than the degenerate global router. Congestion maps produced by the latter approach are between the maps produced by the former method and an actual router.

The improvement in quality of congestion maps by using degenerate global routing is achieved at the price of only little run time. This is due to a very efficient implementation of the underlying A* algorithm and a number of heuristics that focus on improving run time. Crucial is the observation that for the large majority of wires congestion does not play a role. These wires are routed in essentially linear time. Probabilistic methods stamp patterns in usage maps and the size of these stamps grows quadratically with length. In comparison with other methods from the literature both tools developed in this thesis perform excellent on run time.

6.5.1 Estimation refinement

Modern design flows for physical design are based on refinement. Steps that alter the netlist are traditionally part of logic synthesis but may now be used e.g. during placement. Buffer insertion and gate sizing for instance are typically (partially) performed after (coarse) placement. Because of high design densities routability is an issue that needs to be addressed during floorplanning and placement. One can imagine congestion maps that are refined concurrently with the netlist refinement. Unfortunately, probabilistic methods can not really provide such refinement. They can incrementally adapt the congestion maps to the new netlist but they do not have ways to invest run time in more accurate estimation. Degenerate global routing on the other hand is essentially a global router with a number of heuristics for speedup. It is relatively easy to add these heuristics to a sequential A*-based global router or to extend a degenerate global router to a real global router. Both FaDGloR and the global router developed in Chapter 7 share for instance a large part of their source code. Then, options can be provided to switch certain heuristics on and off and generate increasingly accurate congestion estimates. It is also possible to generate coarse congestion estimates first and only refine where routability is questionable. Depending on the algorithms that use the congestion estimator under the hood, the stage of the design and the preference of the designer the tradeoff between run time and accuracy can be set as desired.

6.5.2 Blockages

In this chapter blockages have not been discussed so far. In previous approaches such as [84, 69, 137, 138], methods to deal with routing blockages are described. Essentially, all these methods redistribute the congestion of the blocked area over the neighboring region in a post-processing step. Probabilistic methods spend as little time as possible on evaluating interaction between nets and routing resources and are therefore not very good at dealing with blockages. Published congestion estimation methods therefore first ignore the blockages, resulting in blockage violations. Post-processing is one way of dealing with this and can take into account that especially at the boundaries of blockages problems are
expected. Such methods can be made to fit observed router behavior quite closely.

Since FaDGloR is essentially a router it can inherently deal with routing blockages. Blockages are simply modeled as absent edges and nodes in the routing graph. While expanding, A* is obviously unaware of absent routing edges and this will typically result in congestion near the blockages. Essentially, FaDGloR has the same problems as global routers using the same basic technology. Since FaDGloR is allowed to spend little time this will also result in a pessimistic congestion picture but not to the extend as probabilistic methods.

6.6 Summary and conclusions

Global routers are tuned towards *enhancing routability* while congestion estimators are tuned towards *fast run times*. Essentially, an entirely different tradeoff is made. Although many routers have options to reduce run times their infrastructure is not suitable for congestion estimation purposes. The proposed tool FaDGloR was developed especially for congestion estimation and employs many of the techniques that are used in global routers, but also a number of new heuristics for speedup are used.

The success of FaDGloR depends on tuning and the interaction between a number of heuristics. The wires are routed in order of increasing length and only little detour is allowed. Instead of rip-up and reroute the routing process consists of two phases. During the first phase wire length is minimized and no overflow is allowed at all. During the second phase the unroutable wires are routed with minimum overflow. The cost function is length-based and does not take congestion into account.

In comparison with probabilistic methods FaDGloR is not much slower, and in some cases even faster. The tool is much more accurate in the regions of a chip that are important with respect to routing. Especially on designs and areas that are routable but do not have a lot of excess routing capacity probabilistic methods tend to flag so many overflows that it appears as if the design or area is not routable at all. FaDGloR produces much more realistic results.

A tool such as FaDGloR has many options that can be used to guide its behavior. For a designer such settings may not always be clear but a vendor can provide a set of predefined options such that the behavior of the tool can be varied from very fast but very crude congestion estimation all the way to full fledged global routing. FaDGloR is closer to the crude congestion estimation tool but it is possible to improve the accuracy for example by allowing more detouring or adding cost functions and R&R. This way it is in principle possible to seamlessly go from congestion estimation to global routing, although the requirements of physical synthesis make this difficult in practice.
Chapter 7

Global routing

The subject of this thesis is congestion: how to analyze it and how to deal with it. Global routing is the last phase in the design process that has a global view over the congestion problem and the freedom to change the congestion distribution. Following steps such as detailed routing face the congestion as distributed by global routing and only have limited freedom to solve their problems. On the one hand, global routing is a versatile tool that is used to feed back for example timing, cross-talk and of course congestion information to algorithms higher in the flow. On the other hand, global routing is the first routing step and is therefore crucial for chip implementation. The latter task is the main focus of this chapter: finding approximate routes for wires such that detailed routers can complete the routing task.

In this chapter a global router is presented. Its main novelty is the optimization of secondary criteria such as bends and vias using a *tie breaking* mechanism. The main focus is the optimization of congestion and overflow. This has always been the main purpose of global routers, and the presented router does not sacrifice routability related criteria in order to achieve e.g. better bend counts.

7.1 Purposes of global routing

Global routers are used in a number of different ways depending on design style, manufacturing technology and flows and preferences of companies or individual (groups of) designers. The global router as described in this thesis is a traditional global router in the sense that it is versatile and does not focus solely on a single optimization metric. In order to understand the design of the router the most important purposes of global routers are shortly discussed.

7.1.1 Complexity reduction

Because of sheer design size the routing phase is split up in (at least) two phases of which global routing is the first. Let us make a (simplified) analysis of the benefits of this approach using the *true freedom* metric.
Detailed routing typically uses a very fine grid model of the routing area\(^1\). This grid is derived from the manufacturing grid and design rules. Fig. 7.1 illustrates how the number of possible routes is reduced dramatically by lumping four original nodes into one single node. The task of global routing is to find routes in such a coarser graph while detailed routing refines these results by finding paths in those parts of the original graph associated with the path in the coarse graph.

\[ f_{true}(a, b) = \frac{|r_a - r_b| + |c_a - c_b|!}{|r_a - r_b|! \cdot |c_a - c_b|!} = \frac{6!}{3! \cdot 3!} = 20. \tag{7.1} \]

In the coarse global routing graph (Fig. 7.1, right), the number is only 2. The same analysis using LZ freedom yields 6 and 2 for the original and coarsened graph, respectively. Obviously, the search space for the router has reduced dramatically even in this small example.

Since almost all wires are routed without a detour, let us focus on the detour-free paths in a routing graph in order to have a realistic view on the number of paths global routing considers. The number of detour-free paths between nodes \(a\) and \(b\) in Fig. 7.1, left is according to Eq. 3.17

\[ f_{true}(a, b) = \frac{|r_a - r_b| + |c_a - c_b|!}{|r_a - r_b|! \cdot |c_a - c_b|!} = \frac{6!}{3! \cdot 3!} = 20. \tag{7.1} \]

The freedom reduction due to coarsening depends not only on the distance between the pins but also on the ratio between row and column distance. In this particular case the ratio is one, yielding the maximum reduction for this pin distance. In practice many wires reside in a single row or column. In those cases there is no freedom reduction by graph coarsening\(^2\). Nonetheless, the factorial in the numerator of the expression for true freedom ensures that for paths with non-zero row and column distances the number of paths explodes with increasing path lengths. If only relatively few such wires are present, linear coarsening of the routing graph yields super-linear reduction in total freedom. In practice a global routing graph is constructed such that each row of edges represents a row of standard cells. In modern technologies this is about 10 routing tracks per layer. In this thesis the grid size is not considered an optimization variable. Instead, the coarseness of the global routing graph is considered to be input of the router.

---

\(^1\) Gridless routing approaches also exist but do not seem to be popular in the industry.

\(^2\) Note that a coarser graph still yields better run times since the distance in terms of nodes in the routing graph has decreased.
7.1.2 Wire delay estimation

In order to achieve timing closure designers need reasonably accurate estimates of wire delay. In modern technologies wire delay has become more important than gate delay and using global routing to estimate wire delay is much more accurate than e.g. bounding box- or distance-based methods. Global routing decides which wires are detoured in the presence of congestion and also how nets are decomposed in wires. Note that the decomposition methods of Chapter 4 are indeed part of the global router presented here. If necessary the global routing result can be used to create an RC tree and methods to calculate Elmore delay or Asymptotic Wavefront Evaluation (AWE) can be used to acquire accurate delay estimates.

7.1.3 Congestion control and routability

As argued in the foremost important goal of global routing is congestion management. This is interpreted in two ways. Firstly, if congestion is not properly distributed, detailed routing will not be able to complete the design. Secondly, global routing results are used to aid other tools and designers to make sure that the design will be routable. This topic has been discussed extensively earlier in this thesis but we do want to stress that our router needs to be considered with this in mind.

7.2 Objectives for global routing algorithms

The main purpose of global routing is to enable detailed routers to complete the design. This can only be tested by running these routers and is not very useful as a metric to be used within a global routing algorithm. Instead this goal is represented by traditional metrics such as overflow and congestion.

Because of the paramount importance of routability the focus is traditionally on overflow and congestion related metrics. However, other metrics to measure router quality are also important. Examples of such consideration are timing, reliability, yield and run time. In this section both the motivation to use these metrics and the way they are quantified are discussed.

7.2.1 Overflow and congestion distribution

Overflow is the amount by which routing demand exceeds routing supply. It is the most direct way to model routability in the global routing abstraction. Because overflow is defined on the edges in the global routing graph the presence of overflow guarantees unroutability unless the detailed routers deviate from the global route (or resources have been modeled conservatively). Unfortunately the opposite is not necessarily true: the absence of overflow does not guarantee routability. In practice resource modeling is such that routability is likely in absence of overflow, i.e. the modeled routing supply is less than

3 Especially when the same router is used for estimation and actual implementation of the wires the estimate has essentially become a self-fulfilling prophecy which is accurate almost by definition.

4 When overflow is defined on the nodes in the global routing graph (tiles) overflow does necessarily yield unroutability: for global routing the pins are essentially moved to the center of their tiles and hence the length of the connections may be overestimated.
the actual routing supply (about 80-85%). As a result, a little overflow will typically not be disastrous. Detailed routers are very creative in using spare capacity in neighboring tiles to solve overflow problems. Overflow is defined in Definition 3.3 as the amount by which routing usage exceeds routing resources. In this chapter the numbers reported as overflow are found by summing up the overflow of all the edges in the routing graph.

Simply counting overlap does not capture the entire picture. Congestion distributions\(^5\) as shown in Fig. 7.2 give a better picture. Based only on the overflow criterion the first distribution is superior. The problem is that it has many edges with a congestion level between 0.9 and 1. Routability cannot be guaranteed at such levels. Because of the high congestion levels of most edges the detailed router has little room to manoeuvre, and routability is unlikely. The second distribution has a little overflow. Because of conservative resource modeling and the low congestion of most edges detailed routers are likely to complete the design.\(^6\) Due to lack of space, instead of reporting the congestion distributions for each design and each run the number of edges with a congestion level of at least 0.9 are reported.

7.2.2 Bends and vias

Contacts and vias are used to connect two layers. Vias are between routing layers while contacts are between active and the first routing layer. Both are undesirable for a number of reasons as outlined below.

Congestion

Vias and contacts are created in a different process step than the wires and polygons they connect. In recent technologies they can be the made of the same material (copper) while in older technologies they are typically made from other materials such as tungsten (wires are typically made from aluminum). In order to avoid overlay problems it is necessary to have margins around the polygons that are to be connected. In some technologies vias are also larger than the wires they connect. As illustrated by Fig. 7.3 this means vias and contacts occupy valuable routing resources.

\(^5\)The congestion distribution is not to be confused with the congestion distribution maps of Chapter 5.

\(^6\)Also, a few relatively minor changes in placement or circuit structure may solve problems.
Yield-loss

Contacts and vias have notoriously high yield-loss. This is closely related to the manufacturing process. Random particles are in some cases to blame but also lithography and etching have their limitations. Additionally, the crystallization process does sometimes yield contacts and vias that have grains and are not perfectly connected to the routing layers. Heath gradients that exist during manufacturing also cause stress in contacts and vias and may cause breakdown. In standard cell libraries single contacts are the most important source of yield-loss, and in a standard cell optimizer such as Takumi Enhance contact doubling as illustrated by Fig. 7.4 is given priority over e.g. printability-related layout improvements.

Resistance

Vias can have a relatively high resistance. This can simply be because of dimensions and material properties, but it may also be the case that a via is not well-connected to the wires due to the aforementioned manufacturing-related issues. This resistance may lead to problems with heath or timing.

Reliability

Contacts and vias are also an important cause of reliability issues. During the lifetime of a chip currents flow through most wires in a single direction. Unfortunately it is not only current that flows and some material also moves through the wires as a result of net electron movement. This is known as electromigration[81]. Electromigration may cause opens especially in contacts and vias. These are sensitive because they consist of only little material and the geometries produced by the manufacturing process are simply sensitive to the phenomenon.
Stacked vias

A special case that is often ignored in the literature is stacked vias. These are two or more vias on top of each other. As illustrated by Fig. 7.5 this consumes routing resources in the intermediate layer(s). Unfortunately this is not captured in the global routing model that focuses on horizontal and vertical edges. Stacked vias are especially likely to occur in congested areas\(^7\). Consequently, the global routing graph overestimates the available routing resources especially in congested areas. Finally, some processes do not allow stacked vias at all causing difficulty for designs with many bends.

Quantifying bend minimization

Due to the tree decomposition of nets a certain number of bends cannot be avoided by the router. Each non-zero freedom wire requires at least one bend. In this chapter, we distinguish unavoidable bends, those are the bends associated with the tree decomposition, and routed bends, which is the number of bends produced by a router. With excess bends we denote the difference between routed and unavoidable bends. Note that potential reductions in bends for the global routing algorithm are limited to the excess bends\(^8\).

7.2.3 Detour and detour distribution

The amount of detour produced by routers is low compared to total wire length. Therefore it is better to compare the total detour when comparing two routing results than to compare the total wire length. In the literature however wire length is after overflow and run time the most reported metric.

As stated in [139], wire length is not a very good metric for global routing performance. If overflow can be avoided at the cost of wire length a router should usually detour, as illustrated by Fig. 7.6. Higher wire length corresponds to higher average congestion and therefore it is sometimes argued that the wire length metric is necessary for routability reasons. However, we note that that consideration is already covered by the metrics overflow and congestion distribution. These capture routability better than wire length or detour metrics.

---

\(^7\) There may be other reasons to change layers related to manufacturing such as electrostatic discharge (antenna effect) as well.

\(^8\) This is not entirely accurate as will be shown later.
Detour distribution

When global routing results in detours on some wires or nets this is usually unexpected. Tricks such as gate sizing, logic restructuring or partial re-placement may be necessary to make sure time budgets are met. Small detours can often be ignored due to the presence of slack\(^9\) that has been budgeted in order to be able to cope with this kind of issues. Timing problems due to large detours however are not so easily fixed. Note that it is often arbitrary which wire is detoured by a router if multiple wires are competing for the same routing resources. We argue that the distribution of detours should be considered rather than the total amount of detour.

Fig. 7.7 shows two detour distributions. The first one is superior if only the total detour is taken into account. Unfortunately it has a number of large detours that may cause problems for the convergence of the design flow. The second distribution has many low detours which are more easily repaired. In practice the second distribution is preferable.

Note that the problem with large detours is aggravated by the fact that wire delay is roughly quadratic with wire length\(^10\). In other words: wire delay is relatively insensitive to small detours but increasingly sensitive to larger detours.

7.2.4 Run time

In a physical synthesis flow a significant part of the run time is spend on global routing. In order for designers to be productive it is important that they can run the router at least

---

\(^9\) Slack is the difference between the required arrival time of a signal and the its actual (simulated) arrival time.

\(^10\) Long wires are buffered in practice yielding linear delay but also yielding difficult buffering problems.
over night. In this chapter run times of our router are compared with run times of other tools to make sure run times are practical. Overall run time is considered to be of less importance than the other metrics.

### 7.3 Implementation and experimental setup

This chapter discusses many heuristics to improve the routing quality. Many experiments were conducted to verify the assumptions and evaluate the results on the aforementioned criteria. For the sake of clarity experimental results will be presented right after a discussion instead of in a separate section as in the previous chapters.

#### 7.3.1 Overview of the router

The ideas in this chapter have been implemented in a global router called Grawet\(^\text{11}\). It is based on the same classical architecture as FaDGloR and in fact shares some of the same source code. The basic architecture was discussed in section 6.3. Figure 6.2 gives a good overview and a graphical illustration is given in Fig. 7.8. This chapter deals with the Routing step which is broken up in the sub-steps Wire ordering, Initial routing and Rip-up & reroute. Step Steiner Tree decomposition uses the algorithms as discussed in Chapter 4. The router has a large number of options and algorithms. Details are given in the following sections.

\(^{11}\text{Grawet stands for Global Router with A*, Wavefront Expansion and Tiebreakers.}\)
### Table 7.1: The Labyrinth benchmarks.

<table>
<thead>
<tr>
<th>design</th>
<th>grid</th>
<th>nets</th>
<th>wires</th>
<th>design</th>
<th>grid</th>
<th>nets</th>
<th>wires</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>64×64</td>
<td>12k</td>
<td>27k</td>
<td>ibm06</td>
<td>128×64</td>
<td>33k</td>
<td>79k</td>
</tr>
<tr>
<td>ibm02</td>
<td>80×64</td>
<td>18k</td>
<td>53k</td>
<td>ibm07</td>
<td>192×64</td>
<td>44k</td>
<td>105k</td>
</tr>
<tr>
<td>ibm03</td>
<td>80×64</td>
<td>22k</td>
<td>44k</td>
<td>ibm08</td>
<td>192×64</td>
<td>48k</td>
<td>128k</td>
</tr>
<tr>
<td>ibm04</td>
<td>96×64</td>
<td>26k</td>
<td>52k</td>
<td>ibm09</td>
<td>256×64</td>
<td>50k</td>
<td>124k</td>
</tr>
<tr>
<td>ibm05</td>
<td>128×64</td>
<td>28k</td>
<td>90k</td>
<td>ibm10</td>
<td>256×64</td>
<td>64k</td>
<td>175k</td>
</tr>
</tbody>
</table>

### 7.3.2 Implementation details

Grawet has been implemented in the C++ programming language [123] and the GCC compiler has been used for compilation. The router does not use any external libraries. It natively reads the Labyrinth input format and a configuration file. All options from the configuration file are also available as command-line options. The output consists among others of a text output with statistics on metrics such as overflow, congestion, and run time and also a number of output files that can be used for inspection using e.g. Matlab.

### 7.3.3 Benchmarks

To test Grawet the well-known Labyrinth benchmarks are used. Until recently these were the most commonly used global routing benchmark [12]. Note that these benchmarks were also used in Chapters 4 and 6. The most important characteristics are repeated in Table 7.1 for the reader’s convenience.

### 7.4 Wire ordering

Sequential routing techniques are known to suffer from the net ordering problem. Loosely formulated, the problem is to choose the order in which the nets (wires) are routed optimally. Fig. 7.9 shows how choosing a different wire ordering can solve problems. If the wire ordering is such that the middle wire is routed before the left-most wire the best solution is found (right). Otherwise the left-most wire may occupy routing resources needed by the middle wire (left).

It is common practice to use a shortest-first strategy, i.e. nets with the lowest expected length (based on RSMT or bounding box estimates) are routed first. The reasoning is that nets that are routed first are less likely to have problems. Longer nets will have more possibilities to find alternative paths and are therefore routed last. In our analysis we also note that pin positions are important as well. If a pin is in a congested area the router cannot avoid going through that area. Long wires have few pins relative to their length.

Although there seems to be some understanding of the wire ordering problem in the routing community there is little recent literature on it. Possibly this is the case because the availability of multiple routing layers has made analysis based on planarity less useful. Although anecdotal evidence suggests that the shortest-first approach is standard, according to [30] there is little impact of net ordering during R&R on the solution quality.

---

12 The ISPD benchmark set [91] was introduced after completion of the experiments in this thesis.
Amongst others shortest-first and longest-first methods are evaluated in that paper. In the presented router multi-pin nets are broken up in two-pin wires that are treated (largely) independently. Instead of net ordering, wire ordering is proposed. Since many routers are based on point-to-point routing this method is likely to be applicable elsewhere. Additionally, in the presented method the wires are ordered before initial routing contrary to [30] where the nets were ordered after initial routing and before R&R.

### 7.4.1 Wire ordering with single wire optimal router

Solving the wire ordering problem is not equivalent to solving the global routing problem. A route that is optimal from the perspective of a single wire may block subsequent wires. Thus, a perfect single wire router may even in combination with the best possible wire ordering not return the best overall result as illustrated by Fig. 7.10. Consider for example the first wire in the left picture. It may be routed optimally with the bend either top-left or bottom-right. Since the router does not have a preference it may choose the bottom-right route. If the wire ordering is reversed there is a similar problem as illustrated in the middle picture. On the right the optimal solution is shown.

### 7.4.2 Wire ordering based on freedom

The main motivation for a shortest wires-first approach is that longer wires are supposed to have more possibilities to avoid congested areas. In Chapter 3 a better metric was introduced for this.

**Observation 7.1.** The true freedom of a wire is the number of detour-free realizations that exist for that wire. If high-freedom wires are routed last they are more likely to be routed without problems than low-freedom wires.

On average long wires have higher freedom than short wires. However, the short wire shown in Fig. 7.11, left, has larger freedom than the long wire. On the right it is shown that wires with equal true freedom are not necessarily of equal length. Typically, shortest
7.4 Wire ordering

Figure 7.11: Short wires may have more freedom than longer wires (left). Wires with equal amount of freedom do not need to be equally long (right).

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>length lo1st / hi1st</th>
<th>true freedom lo1st / hi1st</th>
<th>lolength1st</th>
<th>lol.freed.1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>40</td>
<td>27 / 72</td>
<td>27 / 56</td>
<td>21</td>
<td>24</td>
</tr>
<tr>
<td>ibm02</td>
<td>0</td>
<td>0 / 0</td>
<td>0 / 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ibm03</td>
<td>0</td>
<td>0 / 0</td>
<td>0 / 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ibm04</td>
<td>129</td>
<td>93 / 156</td>
<td>115 / 137</td>
<td>115</td>
<td>122</td>
</tr>
<tr>
<td>ibm05</td>
<td>0</td>
<td>0 / 0</td>
<td>0 / 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ibm06</td>
<td>0</td>
<td>0 / 0</td>
<td>0 / 2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ibm07</td>
<td>7</td>
<td>0 / 37</td>
<td>0 / 22</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ibm08</td>
<td>3</td>
<td>1 / 7</td>
<td>2 / 11</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>ibm09</td>
<td>0</td>
<td>0 / 0</td>
<td>0 / 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ibm10</td>
<td>0</td>
<td>0 / 4</td>
<td>0 / 3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

7.4.3 Experimental results

Tables 7.2–7.5 show the results of experiments with different wire orderings. These results were obtained with Grawet using the same settings except for the wire ordering. In all tables the first column shows the design and the second column the reference results with a random wire ordering. The third column shows the results of using length for wire ordering, first using a lowest length first strategy and then using a highest length first strategy. The next column shows the results for using true freedom for wire ordering, also distinguishing between lowest and highest first. Since many wires may have the same freedom but different lengths, length is used as a tiebreaker. The results are shown in column 5. The results for swapping the two criteria are shown in the final column. Note that for these last two experiments only lowest first strategies are used.

Overflow and congestion distribution

Table 7.2 shows the overflow under different wire orderings. There are differences and the low length first and low true freedom first criteria seem to be better than the reversed orders. The results are not conclusive since the numbers are too small to draw reliable conclusions but they do support intuition and common practice. As argued in Chapter 7.2.
Table 7.3: Congestion > 0.9 under different wire orderings.

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>length</th>
<th>true freedom</th>
<th>lolenth1st</th>
<th>lolened1st</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>lo1st / hi1st</td>
<td>lo1st / hi1st</td>
<td>lo1st / hi1st</td>
<td>lolenth1st</td>
<td>lolened1st</td>
</tr>
<tr>
<td>ibm01</td>
<td>2849</td>
<td>2841 / 2861</td>
<td>2795 / 2995</td>
<td>2863</td>
<td>2755</td>
</tr>
<tr>
<td>ibm02</td>
<td>3370</td>
<td>3046 / 3356</td>
<td>3051 / 3424</td>
<td>3020</td>
<td>3106</td>
</tr>
<tr>
<td>ibm03</td>
<td>2338</td>
<td>2134 / 2470</td>
<td>2028 / 2582</td>
<td>2086</td>
<td>2028</td>
</tr>
<tr>
<td>ibm04</td>
<td>3804</td>
<td>3594 / 3759</td>
<td>3416 / 3814</td>
<td>3671</td>
<td>3397</td>
</tr>
<tr>
<td>ibm05</td>
<td>374</td>
<td>271 / 475</td>
<td>253 / 407</td>
<td>271</td>
<td>254</td>
</tr>
<tr>
<td>ibm06</td>
<td>4613</td>
<td>4125 / 4809</td>
<td>4153 / 4923</td>
<td>4140</td>
<td>4064</td>
</tr>
<tr>
<td>ibm07</td>
<td>4487</td>
<td>4165 / 4647</td>
<td>4143 / 4783</td>
<td>4094</td>
<td>4123</td>
</tr>
<tr>
<td>ibm08</td>
<td>5811</td>
<td>5578 / 5924</td>
<td>5344 / 6039</td>
<td>5501</td>
<td>5346</td>
</tr>
<tr>
<td>ibm09</td>
<td>7641</td>
<td>6687 / 8120</td>
<td>7070 / 8067</td>
<td>6712</td>
<td>6909</td>
</tr>
<tr>
<td>ibm10</td>
<td>4770</td>
<td>4037 / 4858</td>
<td>4358 / 4909</td>
<td>4114</td>
<td>4269</td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>90 / 105</td>
<td>89 / 105</td>
<td>90</td>
<td>88</td>
</tr>
</tbody>
</table>

it is better to consider the whole congestion distribution instead of only counting overflow. Not all distributions are printed due to lack of space but instead the number of congested edges ($C(e) > 0.9$) are reported in Table 7.3. This is the tail of the distribution and the reported edges are referred to as congested edges.

Clearly, low length first and low true freedom first are superior, with freedom the best with only a small margin. The difference with the random wire ordering is about 11%, and this results in improved routability for the subsequent detailed routing step. Using some combination of freedom and length for wire ordering does not significantly improve the results. The experiments involve many edges and from a statistical point of view the results give a good impression of average behavior.

Note that the results of random wire ordering are not in the middle between e.g. low length first and high length first. The results of random wire ordering is closer to high length or high true freedom first than they are to the results of low length or true freedom first. This suggests that wire ordering is beneficial for routers that do not perform wire ordering. This observation may also explain the results of [30]. In that paper net ordering is used. A net may of course consist of multiple wires. On average shorter nets will usually consist of shorter and less wires, but in practice a shortest first net ordering differs enough from a shortest wire first ordering to make the approach ineffective.

Bends

Table 7.4 shows how the number of excess bends varies with the wire ordering. The number of excess bends is the difference between the number of routed bends and the minimum number of bends of the steiner trees representing the nets. Low true freedom first is evidently the best. An improvement of about 34% is made over a random wire ordering. This reduction is the result of the increased routability and a direct consequence of the improvement in how congestion is distributed over the map. Once more, note that a combination of wire orderings does not help significantly.

Benchmark ibm05 has some negative excess bends. This issue is discussed in Chapter 7.6.3.
7.4 Wire ordering

Table 7.4: Excess bends under different wire orderings.

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>length</th>
<th>true freedom</th>
<th>length1st</th>
<th>freed.1st</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>lo1st / hi1st</td>
<td>lo1st / hi1st</td>
<td>lolength1st</td>
<td>lolot.freed.1st</td>
</tr>
<tr>
<td>ibm01</td>
<td>4776</td>
<td>4834 / 5444</td>
<td>4096 / 5851</td>
<td>4736</td>
<td>3887</td>
</tr>
<tr>
<td>ibm02</td>
<td>8732</td>
<td>6868 / 10042</td>
<td>5680 / 10561</td>
<td>6541</td>
<td>5861</td>
</tr>
<tr>
<td>ibm03</td>
<td>6656</td>
<td>4659 / 8569</td>
<td>3688 / 9379</td>
<td>4692</td>
<td>3893</td>
</tr>
<tr>
<td>ibm04</td>
<td>9258</td>
<td>7958 / 10449</td>
<td>6246 / 10938</td>
<td>8092</td>
<td>6357</td>
</tr>
<tr>
<td>ibm05</td>
<td>-149</td>
<td>-814 / 798</td>
<td>-1017 / 643</td>
<td>-873</td>
<td>-1018</td>
</tr>
<tr>
<td>ibm06</td>
<td>10120</td>
<td>7719 / 14603</td>
<td>6528 / 13554</td>
<td>7698</td>
<td>6050</td>
</tr>
<tr>
<td>ibm07</td>
<td>10307</td>
<td>8241 / 13392</td>
<td>6996 / 14365</td>
<td>7967</td>
<td>6634</td>
</tr>
<tr>
<td>ibm08</td>
<td>10557</td>
<td>9020 / 13696</td>
<td>6958 / 14546</td>
<td>8470</td>
<td>6845</td>
</tr>
<tr>
<td>ibm09</td>
<td>17260</td>
<td>12476 / 22731</td>
<td>10859 / 22937</td>
<td>12311</td>
<td>10613</td>
</tr>
<tr>
<td>ibm10</td>
<td>13914</td>
<td>8988 / 17005</td>
<td>8264 / 17598</td>
<td>9049</td>
<td>7844</td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>79 / 125</td>
<td>66 / 130</td>
<td>78</td>
<td>65</td>
</tr>
</tbody>
</table>

Table 7.5: Run times [s] under different wire orderings.

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>length</th>
<th>true freedom</th>
<th>length1st</th>
<th>freed.1st</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>lo1st / hi1st</td>
<td>lo1st / hi1st</td>
<td>lolength1st</td>
<td>lolot.freed.1st</td>
</tr>
<tr>
<td>ibm01</td>
<td>66</td>
<td>38 / 94</td>
<td>50 / 64</td>
<td>36</td>
<td>43</td>
</tr>
<tr>
<td>ibm02</td>
<td>120</td>
<td>31 / 181</td>
<td>24 / 136</td>
<td>22</td>
<td>33</td>
</tr>
<tr>
<td>ibm03</td>
<td>55</td>
<td>20 / 72</td>
<td>17 / 57</td>
<td>19</td>
<td>17</td>
</tr>
<tr>
<td>ibm04</td>
<td>191</td>
<td>171 / 242</td>
<td>178 / 212</td>
<td>165</td>
<td>175</td>
</tr>
<tr>
<td>ibm05</td>
<td>6</td>
<td>5 / 7</td>
<td>5 / 6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>ibm06</td>
<td>99</td>
<td>35 / 200</td>
<td>49 / 194</td>
<td>34</td>
<td>93</td>
</tr>
<tr>
<td>ibm07</td>
<td>239</td>
<td>77 / 366</td>
<td>98 / 343</td>
<td>44</td>
<td>125</td>
</tr>
<tr>
<td>ibm08</td>
<td>186</td>
<td>146 / 295</td>
<td>117 / 231</td>
<td>136</td>
<td>111</td>
</tr>
<tr>
<td>ibm09</td>
<td>186</td>
<td>87 / 476</td>
<td>105 / 328</td>
<td>88</td>
<td>70</td>
</tr>
<tr>
<td>ibm10</td>
<td>307</td>
<td>94 / 490</td>
<td>111 / 409</td>
<td>96</td>
<td>169</td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>52 / 160</td>
<td>55 / 130</td>
<td>48</td>
<td>60</td>
</tr>
</tbody>
</table>

Run time

Wire ordering also has an impact on run time as shown in Table 7.5. Sorting the wires with low length first yields the best run times. A gain of about 50% compared to a random wire ordering is obtained and the ordering is about three times faster than the reversed order, which is the worst wire ordering in our experiment. Combinations of wire orderings can improve the run times a little bit. The results for low true freedom first are almost as good as for low length first. Note that low length first does not consistently outperform low true freedom first.

Grawet stops with R&R when there is no more overflow. Wire orders that are more effective on overflow removal have lower run times as a result. If Grawet finishes the same number of R&R round for all designs we observe the same trend but with less extreme percentages.
Table 7.6: Easy benchmarks: Congestion > 0.9 under different wire orderings.

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>length lo1st / hi1st</th>
<th>true freedom lo1st / hi1st</th>
<th>lolength1st</th>
<th>lot.freed.1st lolength1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01*</td>
<td>1275</td>
<td>1053 / 1344</td>
<td>1088 / 1230</td>
<td>1042</td>
<td>1060</td>
</tr>
<tr>
<td>ibm02*</td>
<td>2103</td>
<td>1804 / 2015</td>
<td>1835 / 2044</td>
<td>1769</td>
<td>1781</td>
</tr>
<tr>
<td>ibm03*</td>
<td>1470</td>
<td>1432 / 1526</td>
<td>1396 / 1498</td>
<td>1446</td>
<td>1393</td>
</tr>
<tr>
<td>ibm04*</td>
<td>2111</td>
<td>1944 / 2228</td>
<td>2014 / 2215</td>
<td>1937</td>
<td>1983</td>
</tr>
<tr>
<td>ibm05*</td>
<td>205</td>
<td>174 / 212</td>
<td>155 / 196</td>
<td>173</td>
<td>154</td>
</tr>
<tr>
<td>ibm06*</td>
<td>2637</td>
<td>2379 / 2743</td>
<td>2380 / 2723</td>
<td>2375</td>
<td>2379</td>
</tr>
<tr>
<td>ibm07*</td>
<td>2641</td>
<td>2347 / 2841</td>
<td>2302 / 2882</td>
<td>2366</td>
<td>2267</td>
</tr>
<tr>
<td>ibm08*</td>
<td>3361</td>
<td>3046 / 3468</td>
<td>2918 / 3426</td>
<td>3022</td>
<td>2911</td>
</tr>
<tr>
<td>ibm09*</td>
<td>3493</td>
<td>3150 / 3884</td>
<td>3188 / 3599</td>
<td>3095</td>
<td>3123</td>
</tr>
<tr>
<td>ibm10*</td>
<td>3276</td>
<td>3082 / 3328</td>
<td>3119 / 3340</td>
<td>3031</td>
<td>3077</td>
</tr>
<tr>
<td>normal.</td>
<td>100</td>
<td>90 / 104</td>
<td>89 / 102</td>
<td>89</td>
<td>88</td>
</tr>
</tbody>
</table>

Table 7.7: Easy benchmarks: Run time under different wire orderings.

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>length lo1st / hi1st</th>
<th>true freedom lo1st / hi1st</th>
<th>lolength1st</th>
<th>lot.freed.1st lolength1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01*</td>
<td>17</td>
<td>8 / 32</td>
<td>9 / 19</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>ibm02*</td>
<td>34</td>
<td>24 / 67</td>
<td>21 / 40</td>
<td>24</td>
<td>20</td>
</tr>
<tr>
<td>ibm03*</td>
<td>21</td>
<td>18 / 30</td>
<td>14 / 25</td>
<td>18</td>
<td>14</td>
</tr>
<tr>
<td>ibm04*</td>
<td>43</td>
<td>28 / 78</td>
<td>34 / 53</td>
<td>28</td>
<td>34</td>
</tr>
<tr>
<td>ibm05*</td>
<td>24</td>
<td>20 / 26</td>
<td>21 / 23</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>ibm06*</td>
<td>35</td>
<td>42 / 67</td>
<td>31 / 43</td>
<td>42</td>
<td>31</td>
</tr>
<tr>
<td>ibm07*</td>
<td>76</td>
<td>48 / 127</td>
<td>45 / 106</td>
<td>47</td>
<td>44</td>
</tr>
<tr>
<td>ibm08*</td>
<td>56</td>
<td>65 / 107</td>
<td>52 / 73</td>
<td>65</td>
<td>50</td>
</tr>
<tr>
<td>ibm09*</td>
<td>69</td>
<td>125 / 191</td>
<td>64 / 136</td>
<td>119</td>
<td>65</td>
</tr>
<tr>
<td>ibm10*</td>
<td>192</td>
<td>213 / 269</td>
<td>175 / 222</td>
<td>211</td>
<td>175</td>
</tr>
<tr>
<td>normal.</td>
<td>100</td>
<td>94 / 178</td>
<td>77 / 127</td>
<td>93</td>
<td>76</td>
</tr>
</tbody>
</table>

Results for easy designs

The benchmark suite consists entirely of difficult designs with the exception of ibm05. Some designs, macros or parts of designs can be routed without much difficulty in practice. In such cases the primary metric for routability (overflow) will always be zero and secondary criteria such as run time and number of bends are the decisive metrics.

Tables 7.6-7.8 show the results of using different wire orderings for easy benchmarks. These benchmarks have been obtained by adding 2 routing tracks to the routing resources per routing edge. This is not much compared to the routing resources that are already available. In practice macros and designs of this difficulty are not uncommon. As expected, Gravel is able to complete all designs without overflow.

The different wire orderings are compared on three criteria: congested edges, run time and excess bends. The wire ordering with low true freedom first is on average the best for
all three criteria. Even for these easy designs the number of congested edges is reduced by more than 10% compared to a random wire ordering. Run time is also significantly reduced by almost 25%. Remarkably, on the important excess bends criterion the low true freedom first wire ordering yields less bends than the minimum number of bends that exist in the steiner trees representing the nets. This is the result of using wavefront expansion and is explained in Sec. 7.6.3.

7.5 Cost function and routing tiebreakers

Traditional sequential global routers use cost functions to guide shortest path algorithms. In many cases, many paths of equal cost exist for a given wire. During an A* search nodes are added to the heap (refer to Section 6.3.2 and Fig. 6.3 for a discussion on this). The most promising node is next selected for expansion. In many cases, many nodes are equally promising. As illustrated by Fig. 7.12 this is especially the case when there is little or no congestion and A* is essentially length-based. All nodes with a detour-free path going through them are equally attractive because the cost estimate is exact and there are many ways to achieve this cost.

Observation 7.2. The purpose of cost functions as encountered in global routing is to optimize routability. For a given wire multiple minimum-cost paths may exist. This is a degree of freedom that can be exploited to optimize other criteria including run time, detour distribution and number of bends.

In the following sections a number of routing tiebreakers are discussed. When the router compares two (partial) paths, ties in the cost function value are broken using these. Tiebreakers are easy to introduce in existing routers. It is also possible that tiebreakers do not break the tie. In that case a next tiebreaker can be used. In Grawet such a fall-through scheme is used. The method does not rely on tuning of cost functions with all kinds of

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>length</th>
<th>true freedom</th>
<th>lolength</th>
<th>lot.freed.1st</th>
<th>lot.freed.1st</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>lo1st / hi1st</td>
<td>lo1st / hi1st</td>
<td>lolength1st</td>
<td>lot.freed.1st</td>
<td>lolength1st</td>
</tr>
<tr>
<td>ibm01*</td>
<td>2030</td>
<td>1351 / 2812</td>
<td>1132 / 2497</td>
<td>1307</td>
<td>1008</td>
<td></td>
</tr>
<tr>
<td>ibm02*</td>
<td>4913</td>
<td>3186 / 6114</td>
<td>2355 / 6099</td>
<td>3126</td>
<td>2266</td>
<td></td>
</tr>
<tr>
<td>ibm03*</td>
<td>2458</td>
<td>1999 / 3345</td>
<td>1297 / 3568</td>
<td>2033</td>
<td>1362</td>
<td></td>
</tr>
<tr>
<td>ibm04*</td>
<td>4441</td>
<td>3235 / 5871</td>
<td>2758 / 5985</td>
<td>3070</td>
<td>2782</td>
<td></td>
</tr>
<tr>
<td>ibm05*</td>
<td>-292</td>
<td>-868 / 180</td>
<td>-1068 / 218</td>
<td>-930</td>
<td>-1070</td>
<td></td>
</tr>
<tr>
<td>ibm06*</td>
<td>4489</td>
<td>3189 / 6479</td>
<td>2285 / 6320</td>
<td>3122</td>
<td>2267</td>
<td></td>
</tr>
<tr>
<td>ibm07*</td>
<td>6032</td>
<td>3733 / 8726</td>
<td>2808 / 9179</td>
<td>3571</td>
<td>2697</td>
<td></td>
</tr>
<tr>
<td>ibm08*</td>
<td>5521</td>
<td>3737 / 8440</td>
<td>2172 / 8721</td>
<td>3503</td>
<td>2248</td>
<td></td>
</tr>
<tr>
<td>ibm09*</td>
<td>5857</td>
<td>4102 / 10206</td>
<td>2762 / 9231</td>
<td>3927</td>
<td>2648</td>
<td></td>
</tr>
<tr>
<td>ibm10*</td>
<td>7600</td>
<td>4888 / 10310</td>
<td>3481 / 10229</td>
<td>4633</td>
<td>3504</td>
<td></td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>92 / 122</td>
<td>81 / 120</td>
<td>92</td>
<td>81</td>
<td></td>
</tr>
</tbody>
</table>

Table 7.8: Easy benchmarks: Excess bends under different wire orderings.
parameters. Although the router as a whole does not come with a guarantee on optimality some claims are made for some tiebreakers and individual wires.

### 7.5.1 Tiebreakers versus cost scaling

Cost functions are usually based on very local supply and demand of routing resources. Generally however, anything that can be quantified can be used when comparing two nodes. A tiebreaker may for instance have access to the net associated with the wire that is being routed. Then the location of pins of other wires of the same net can be used to break ties. Another example is to take the congestion of nearby edges into account.

It is possible to use scaling to lump the values of all tiebreakers into a single value that can be used in a more traditional scheme. Without predefined bounds on the values of all tiebreakers this is not possible however. There are also practical issues regarding numerical stability when scaling factors are applied. Another issue is that in such a scaling scheme the values for all tiebreakers need to be computed. In our scheme with explicit tiebreakers this is not the case. Only when comparison on a given tiebreaker is necessary the value needs to be computed. As indicated in the previous paragraph, all kinds of tiebreakers can be used and some are relatively expensive to compute. Although it may seem to be more complicated at first, tiebreakers are more efficient in practice.

### 7.5.2 Information theoretic interpretation

By using tiebreakers information is added in information theoretic sense. There is only a limited number of values of the cost function that is actually used during routing. In information theory, the entropy of a message is a measure for the amount of information in a message space \( M \).\(^{111}\)

\[
H(M) = - \sum_{m \in M} p(m) \log p(m),
\]

where \( p(m) \) is the probability of the occurrence of a message \( m \).

Let us now interpret the values of the cost function as encountered by the router as the messages. There is relatively little information in this value since only a limited number

---

**Figure 7.12:** For an A* search between nodes \( s \) and \( t \) all dark nodes have equal distance (cost) estimates. It is therefore arbitrary in which order they are expanded if only wire length is taken into account.
of values of the cost function are used. When two nodes are equal based on their cost function values they can be distinguished based on a tiebreaker. Thus, the number of different nodes increases which in turn increases the entropy.

### 7.5.3 A* with tiebreakers

Based on the discussion above and the discussion of A* in Section 6.3.2, A* with tiebreakers is summarized in Fig. 7.13 and Fig. 7.14. Note that the pseudo-code assumes the tiebreakers work on a set of attributes. This is not strictly necessary but simplifies the discussion. For efficiency reasons it may be necessary to calculate attributes only when they are needed in e.g. a comparison. Attribute $\pi$ denotes the predecessor of a node. In the described scheme it is also possible to use this in the comparison. It may for instance be of interest how many pins are associated with it.

$$A^*-\text{TIEBREAKERS}(G, s, t, \text{COMPARE})$$

1. for each node $v \in V[G]$
2. do INIT-ATTRIBUTES($v$)
3. INIT-SOURCE-ATTRIBUTES($s$)
4. $Q \leftarrow$ MAKE-HEAP(COMPARE)
5. INSERT-HEAP($Q, s$)
6. $u \leftarrow$ EXTRACT-MIN($Q$)
7. while $u \neq t$
8. do for each node $v \in \text{Adj}[u]$
9. do $a \leftarrow$ attributes[$v$]  
   \[\triangleright \text{Get attributes using } u \text{ as } v \text{’s predecessor instead of the current node}\]
10. $b \leftarrow$ GET-ATTRIBUTES($v, u$)
11. if COMPARE($b, a$)
12. then \[\triangleright \text{The route to } v \text{ through } u \text{ is better than the current route}\]
13. attributes[$v$] $\leftarrow$ $b$
14. if $a[\pi] = \text{NIL}$
15. then INSERT-HEAP($Q, v$)
16. else DECREASE-KEY($Q, v$)
17. $u \leftarrow$ EXTRACT-MIN($Q, h$)
18. return $t$

**Figure 7.13:** The A* algorithm with tiebreakers.

### 7.5.4 Cost function

The cost function that is used by a global router has a huge impact on the quality of results. In [30] a number of cost functions is discussed and compared. Since cost functions are relatively well-understood the cost function used in Gravet is based on the discussion there and some additional experimentation (not discussed in this thesis). The cost function shown in Fig. 7.15 was found to be effective.
**COMPARE-TIEBREAKERS**(\(v, w\))

▷ return whether \(v\) is a better candidate than \(w\)

1. \(a ← attributes[v]\)
2. \(b ← attributes[w]\)
3. if \(a[\text{cost}] < b[\text{cost}]\)
   then return \text{TRUE}
4. elseif \(a[\text{cost}] > b[\text{cost}]\)
   then return \text{FALSE}
5. else
   ▷ Same cost: go to tiebreakers
   do if \(a[\text{freedom}] < b[\text{freedom}]\)
      then return \text{TRUE}
6. elseif \(a[\text{freedom}] > b[\text{freedom}]\)
   then return \text{FALSE}
7. elseif \(a[\text{id}] < b[\text{id}]\)

**Figure 7.14:** An example of a tiebreaker compare function.

Cost functions apply to *edges* in the global routing graph. \(A^*\) needs costs for nodes and these are found by summing up edge costs on the path from the source to a given node (using the predecessor attributes). The cost function consists of two parts. The constant part (1) is essentially the *length* of the edge. When congestion is low in the design only this constant part is used. In this case \(A^*\) yields a true shortest-path. The linearly increasing part of the cost function encourages \(A^*\) to explore alternative paths when the congestion of the edge is a higher than some threshold value (0.95 in this case). In that case, \(A^*\) becomes a *lowest cost* algorithm and is no longer guaranteed to find a minimum-length path. The specific location of the break point and the gradient of the cost function were found to be effective after experimentation. Slightly other settings might be better for benchmarks with different characteristics. The benchmarks on which our settings are based are difficult so at least for difficult designs the chosen settings are effective. Since the global routing graph is a grid graph it is easy to calculate the minimum cost between two nodes: it is the \(L_1\) distance \((L_1 = \Delta \text{row} + \Delta \text{col})\) between the two nodes. As long as congestion is low this lower bound is exact and \(A^*\) is very efficient.

![Figure 7.15: The cost function used in Grawet.](image)
7.5 Cost function and routing tiebreakers

7.5.5 Tiebreaker true freedom

Let us consider the situation of Fig. 7.16 where an $A^*$ search for a wire $w = (s, t)$ starts from the node $s$ and $v(s) \leq h(s)$, where $v(n)$ and $h(n)$ represent the vertical and horizontal distance between a node $n$ and the target node $t$, respectively. The wire is called a north-east wire. Only this situation (north-east wire, $v(s) \leq h(s)$) is discussed. The analysis for other cases is similar because of symmetry.

In Chapter 4 one of the objectives was to create as much freedom in the design as possible for routability improvement. Similar ideas can be used during routing.

Observation 7.3. During an $A^*$ search a path is grown. A node $n$ may be expanded during the search, but it will be discarded if no path is found to the target node $t$. The more minimum-length paths exist between $n$ and $t$, the more likely it is that an attractive path exists. Thus, it is a good idea to use freedom preservation during $A^*$.

The true freedom of the wire $w = (s, t)$ is the number of detour-free paths between the two nodes of the wire. Let us define the true freedom $f_{true}(n)$ of a node $n$ during $A^*$ as the true freedom of a virtual wire connecting $n$ and $t$. This number can be used as a tiebreaker during $A^*$ expansion.

Tiebreaker high true freedom

Tiebreaker high true freedom preserves freedom. This means that $A^*$ is expanding towards nodes with high freedom numbers. Using this tiebreaker is not expected to improve the quality of the path found. In case there are multiple minimum-cost paths there is no reason to believe that the path that is found this way is better for the routability of subsequent wires than a path that would be found otherwise. The idea behind this approach is that $A^*$ keeps its options open. Because there are more paths between the node that is currently expanded to the target node, the node is less likely to be rejected. Therefore the main impact of this tiebreaker is on run time.

Intuitively it is easy to see that such a strategy yields a path that is as close as possible to the bold line in Fig. 7.16, left. This yields a staircase path with many bends. In fact, under reasonable conditions it yields the maximum number of bends as proven by Theorem 7.8. In order to prove the theorem some lemmas are needed.

All lemmas and theorems in this section apply to the sketched situation: a north-east wire. In the analysis pure wire length minimization is assumed. Cost functions can be anything and complicate the analysis. The global routing result is to a large extent determined by initial routing. For the cost functions that are used in practice the large majority of wires do not encounter congestion and $A^*$ yields optimal shortest paths. Detours are therefore usually ignored in the theorems.

Lemma 7.4 (East freedom versus north freedom). Consider a north-east wire $w = (s, t)$ in a global routing graph with $s$ the source of an $A^*$ search. Consider a node $n$ in the wire box with $v(n) < h(n)$ and its east and north nodes $n_e$ and $n_n$, respectively. Then $f_{true}(n_e) > f_{true}(n_n)$.

Proof. The lemma is illustrated by Fig. 7.16 and applies to the nodes in the gray area. Intuitively, the lemma must be true because the east nodes in this area are closer to the nodes on the line $v(n) = h(n)$. 
Let us abbreviate $h = h(n)$ and $v = \nu(n)$. Then, the freedom of the neighboring nodes $n_e$ and $n_n$ can be expressed as

$$f_{true}(n_e) = \frac{(h - 1 + v)!}{(h - 1)! \cdot v!} \quad \text{(7.3)}$$

and

$$f_{true}(n_n) = \frac{(h + v - 1)!}{h! \cdot (v - 1)!} = \frac{(h - 1 + v)!}{(h - 1)! \cdot (v - 1)! \cdot h}. \quad \text{(7.4)}$$

Since $\nu < h$, it must be that $f_{true}(n_e) > f_{true}(n_n)$.

**Lemma 7.5** (North freedom versus east freedom). Consider a north-east wire $w = (s, t)$ in a global routing graph with $s$ the source of an A* search. Consider a node $n$ in the wire box with $\nu(n) > h(n)$ and its east and north nodes $n_e$ and $n_n$, respectively. Then $f_{true}(n_n) > f_{true}(n_e)$.

**Proof.** Similar to the proof of Lemma 7.4.

**Lemma 7.6** (East expansion first). Consider a north-east wire $w = (s, t)$ in a global routing graph with $s$ the source node of a fully wire length-driven A* search with high true freedom as a tiebreaker. Consider a node $n$ in the wire box with $\nu(n) < h(n)$ and its east and north nodes $n_e$ and $n_n$, respectively. Then, $n_e$ is expanded before $n_n$.

**Proof.** The proof is illustrated by Fig. 7.16 and applies to the nodes in the gray area.

First we prove that the lemma is necessarily correct for all nodes $n$ in the wire box with $h(n) = h(s)$ and $\nu(n) < h(n)$. Then we show that if the lemma applies to a node $n$ in the wire box with $\nu(n) < h(n) - 1$, the lemma also applies to its east neighbor. Then the lemma must be correct by induction.

1. Consider a node $n$ with $h(n) = h(s)$ and $\nu(n) < h(n)$ and its north and east nodes $n_n$ and $n_e$, respectively. Node $n_n$ can only be added to the heap after an expansion of $n$. If at that time $n_e$ has already been expanded the lemma is correct. If this is not the case both $n_e$ an $n_n$ are in the heap after expansion of $n$. In that case the lemma must be correct because of Lemma 7.4. Hence, $n_e$ is always expanded before $n_n$ and the lemma applies to $n$. 

---

**Figure 7.16:** Illustration of Lemmas 7.4 and 7.6.
2. Consider a node \( n \) with its north and east neighbors \( n_n \) and \( n_e \), respectively, and the north and east neighbors of \( n_e \), \( a \) and \( b \) respectively. If the lemma applies to \( n \), \( n_e \) is expanded before \( n_n \). Then \( a \) is added to the heap after expansion of \( n_e \). If at that time \( b \) has already been expanded the lemma is correct for \( n_e \). If this is not the case, then both \( a \) and \( b \) are in the heap after the expansion of \( n_e \). Then, \( b \) is expanded before \( a \) by Lemma 7.4. Thus, if the lemma applies to \( n \) it necessarily applies to \( n_e \). Then the lemma must be correct by induction.

**Lemma 7.7 (North expansion first).** Consider a north-east wire \( w = (s, t) \) in a global routing graph with \( s \) the source node of a fully wire length-driven \( A^* \) search with high true freedom as a tiebreaker. Consider a node \( n \) in the wire box with \( v(n) > h(n) \) and its east and north nodes \( n_e \) and \( n_n \), respectively. Then, \( n_n \) is expanded before \( n_e \).

**Proof.** The proof is illustrated by Fig. 7.17 and applies to the nodes in the gray area.

First we prove that the lemma is correct for each node \( n \) in the wire box with \( v(n) = v(s) \) and \( v(n) > h(n) \). Then we show that if the lemma applies to a node \( n \) in the wire box with \( v(n) > h(n) + 1 \) it also applies to its north neighbor. Then the lemma must be correct by induction.

1. Consider a node \( n \) with \( v(n) = v(s) \) and \( v(n) > h(n) \) and its north and east neighbors \( n_n \) and \( n_e \), respectively. Node \( n_e \) can only be added to the heap after expansion of \( n \). If at this time \( n_n \) has already been expanded the lemma is correct. If this is not the case both \( n_n \) and \( n_e \) are in the heap and the lemma is correct because of Lemma 7.5.

2. Consider a node \( m \) with its north and east neighbors \( m_n \) and \( m_e \), respectively, and the north and east neighbors of \( m_n \), \( a \) and \( b \), respectively. If the lemma applies to \( m \), \( m_n \) is expanded before \( m_e \). Then, \( b \) is added to the heap after expansion of \( m_n \). If at that time \( a \) has already been expanded, the lemma is correct for \( m_n \). If this is not the case then both \( a \) and \( b \) are in the heap after the expansion of \( m_n \). Then \( a \) is expanded before \( b \) by Lemma 7.5. Thus, if the lemma applies to \( m \) it necessarily applies to \( m_n \). Then the lemma must be correct by induction.

The above lemmas are the basis for the following theorem. When tiebreaker high true freedom is used there may also be nodes with both equal distance estimates and equal
freedom numbers. In that case the assumption is made that $A^*$ will expand those nodes in fixed order, e.g. nodes to the east first.

**Theorem 7.8** (Tiebreaker high true freedom). Consider a north-east wire $w = (s, t)$ in a global routing graph with $s$ and $t$ the source and target nodes in a fully wire length-driven $A^*$ search, respectively. If *high true freedom* is used as a tiebreaker and the tie between two nodes with equal true freedom is broken consistently then a detour-free path with the maximum number of bends is found.

**Proof.** The proof is illustrated by Fig. 7.18. Without loss of generality, let us assume that $A^*$ always breaks ties between nodes with equal true freedom by expanding to the east first. In this case the path $p$ is shown in bold and consists of nodes $n$ in the wire box such that $h(n) \leq v(n) \leq h(n) + 1 \lor v(n) = v(s), h(n) > v(n)$, in order of decreasing $L_1$ distance to $t$.

Let us also define the sets $NW$ consisting of the nodes $n$ in the wire box with $v(n) < h(n)$, and $SE$ consisting of the nodes $n$ in the wire box with $v(n) > h(n) + 1$.

It is easily verified that the bold path is a maximum-bend path. Any alternative path (represented by a dashed line in the figure) needs to have a node $n_b$ with a north-east bend such that $n_b \in NW$ (case I), and/or a node $m_b$ with an east-north bend such that $m_b \in SE$ (case II). We prove that neither can happen by contradiction.

I. Consider a north-east bend at a node $n_b \in NW$, its south and east neighbors $n_p$ and $n_s$, respectively, and the east neighbor of $n_p$, $n'_b$. Node $n'_b$ must have been expanded before $n_b$. If $n_p \in NW$ by Lemma 7.6 otherwise, $(n_p \in p, \text{gray in the figure})$ because of the east-preference of $A^*$. Evidently, there exists a path $n'_b - n_s - t$. Since $n'_b$ was expanded before $n_b$, the path $s \rightarrow n_p - n'_b - n_s \rightarrow t$ is found. This is a contradiction with the fact that we started from a path $s \rightarrow n_p - n_b - n_s \rightarrow t$.

II. Consider an east-north bend at a node $m_b \in SE$, its north and west neighbors $m_s$ and $m_p$, respectively, and the north neighbor of $m_p$, $m'_b$. By Lemma 7.7, $m'_b$ is always expanded before $m_b$ ($m_p$ cannot be a gray node in the figure, since then $m_b \in p$). Then by the same reasoning as case I, a path $s \rightarrow m_p - m'_b - m_s \rightarrow t$ is found. This is a contradiction with the actual path.

Note that east-preference (or alternatively, north-preference) is needed in the proof.
In practice an A* algorithm is likely to have such a preference because when a node is expanded the neighboring nodes are usually added to the heap in the same order.

Theorem 7.8 applies to the uncongested case. In the presence of congestion the maximum number of bends is not guaranteed, but evidently many more bends than necessary result. This analysis demonstrates that high true freedom is not a good tiebreaker.

**Tiebreaker low true freedom**

Since tiebreaker high true freedom yields the maximum number of bends, doing the exact opposite (removing freedom as quickly as possible) might yield the minimum number of bends. Theorem 7.10 proves that this is indeed the case under certain conditions. The tiebreaker that is used is the opposite of high true freedom and is referred to as tiebreaker low true freedom.

First the following lemma is needed.

**Lemma 7.9** (Monotonic freedom). Consider a wire \( w = (s, t) \), with \( s \) and \( t \) the source and target node during an A* search, respectively. The true freedom \( f_{true}(n) \) of a node \( n \) on any detour-free path from \( s \) to \( t \) is never more than the true freedom \( f_{true}(m) \) of its predecessor \( m \) on the path. More specifically,

i. \( f_{true}(n) = f_{true}(m) \) if \( h(m) = 0 \) or \( v(m) = 0 \),

ii. \( f_{true}(n) < f_{true}(m) \) otherwise.

**Proof.** The two cases are proven independently.

i. If \( h(m) = 0 \) or \( v(m) = 0 \) only one detour-free sub-path \( m \rightarrow t \) exist. Node \( n \) must be on this path. Both \( n \) and \( m \) have a true freedom of 1.

ii. Node \( m \) has exactly two neighbors which are on some detour-free path from \( m \) to \( t \). Only one of these neighbors is \( n \). Therefore \( f_{true}(m) > f_{true}(n) \).

Using the above lemma it is proven that purely wire length-driven A* with low true freedom as a tiebreaker always yields a minimum-bend route. This is interesting since freedom is only a local criterion. Essentially, monotonicity of freedom guarantees that once a direction of expansion is chosen A* continues in this direction since the true freedom decreases with each expansion.

**Theorem 7.10** (Tiebreaker low true freedom). Consider a north-east wire \( w = (s, t) \) in a global routing graph with \( s \) and \( t \) the source and target nodes in a fully wire length-driven A* search, respectively. If low true freedom is used as a tiebreaker then a detour-free path with the minimum number of bends will be found.

**Proof.** Three cases are distinguished.

I. \( v(s) = 0 \). In this case only one detour-free path exists without any bends.

II. \( v(s) < h(s) \). The north node \( s_n \) of the source node \( s \) has lower true freedom than its east node \( s_e \) by Lemma 7.4. Since \( s \) is the first node to expand, \( s_n \) is expanded next. Now assume that after an expansion of a node \( n \) with \( h(n) = h(s) \) and \( v(n) > 0 \) its north node \( n_n \) expands. By Lemma 7.9 the true freedom of both the north and east...
neighbors of $n_n$ must be lower than that of any node in the heap. Then because of Lemma [7.4], the north neighbor of $n_n$ is expanded after $n_n$. It follows by induction that A* expands northwards until a node $n_b$ is added to the heap with $v(n_b) = 0$.

All nodes on the detour-free path from $n_b$ to $t$ have true freedom 1 which is lower than the freedom of any node in the heap except $n_b$. Hence, $n_b$ is expanded next and by induction it follows that A* expands directly towards $t$, yielding a single bend at $n_b$. A single bend is minimal in this case.

III. $v(s) = h(s)$. The north and east nodes $n_n$ and $n_e$ of $s$ have equal freedoms. Thus, it is arbitrary which of these two is expanded first. Then, because of the monotonicity of true freedom it follows by the same reasoning as in the previous case that A* continues expanding in that direction until no longer possible. The single bend is then created and A* expands directly to $t$. A single bend is optimal.

Unfortunately the above theorem only applies to the case where congestion does not play a role. In the presence of congestion optimality is not guaranteed as illustrated by Fig. 7.19. The behavior of A* is better characterized as following the short side than as minimizing bends. If the remaining part of the wire has such a short side, A* will follow it regardless of the current direction of expansion. In a shortest-first wire ordering the wires which may cause the largest number of bends are routed last. In case of a dense design these wires will surely encounter congestion and the router may choose a path with more bends than strictly necessary.

**Theorem 7.11** (Tiebreaker low true freedom and congestion). Using low true freedom as a tiebreaker in an A* search will not necessarily lead to a minimum-cost minimum-bend path in the presence of congestion (edges with increased cost).

**Proof.** By example. Consider Fig. 7.20. All shown edges have a cost of one, except the one marked with two. There are two minimum-bend paths of which only one has minimum cost: the path through the bottom-right node. We prove that A* with low true freedom as a tiebreaker does not find this path.

By the reasoning of Theorem 7.10 A* starts its search from $s$ by expanding to the north. When it encounters the edge with increased cost, nodes $n$ and $m$ have been added to the heap. For both nodes $m$ and $n$ a minimum-cost path going through it exists and also a minimum-cost path going through both of them with node $m$ being the predecessor of $n$ in this case. By Lemma [7.9], $n$ has lower true freedom than $m$ and is therefore expanded before $m$. By induction it now follows that the path shown in the figure will be found. This is obviously not the minimum-cost path with the least number of bends. 

![Figure 7.19: Low true freedom as a tiebreaker in the presence of congestion is best characterized as following the short side.](image-url)
Although no optimality can be guaranteed in practice because of the above theorem, in practice the method may work reasonably well. This should be verified by experiments.

### 7.5.6 Tiebreaker bends

Bends are an important quality metric as discussed in Section 7.2. Experimental results indicate that true freedom is not very effective as a tiebreaker for bend minimization. Therefore a more direct way to minimize the number of bends is proposed: a tiebreaker based on tracking the number of bends exactly. With this method the minimum number of bends given the locations of the other wires is guaranteed to be found.

Cost functions and the tiebreakers so far are based on local information. In A* the cost estimate of a node $n$ is based on the actual cost between its predecessor $m$ and the source node $s$ and the distance to the target node $t$. Only information from the predecessor node is needed. There is no need to know about the actual path between $s$ and $m$.

$$\text{cost}(n) = \text{cost}(s, m) + \text{cost}(m, n) + \mathcal{L}_1(m, t)$$ (7.5)

Evidently, the freedom of a node is also local information.

**Observation 7.12.** It is not possible to minimize the number of bends of a path by changing the costs as used during A* based on local information only. Efficient length and cost-based implementations of A* all use such local information only.

Keeping track of the number of bends that was used to reach a node is sufficient for optimality but this number of bends cannot be found by querying the predecessor node only. Since the algorithm is running on a grid graph precise lower bounds on the number of bend from a node to the target can also be calculated. Essentially, for keeping track of bends we apply the same trick that A* uses to improve upon Dijkstra’s algorithm.

Tie breaking occurs based on the *bends* number of a node $n$, which is defined as

$$\text{bends}(n) = \text{bends}(s, n) + \text{bends}_{\text{est}}(n, t)$$ (7.6)

where $\text{bends}(s, n)$ represents the number of bends between the source node $s$ and $n$ and $\text{bends}_{\text{est}}(n, t)$ is a lower bound on the number of bends between $n$ (inclusive) and the target node. The bends number is a lower bound on the total number of bends on the final path. Details on how to calculate this number are given in the next section.

The approach yields an optimal result as stated in the following theorem.

**Theorem 7.13 (Optimality of tiebreaker bends).** Consider an A* search for a wire $w = (s, t)$ in a global routing graph with costs on the edges. Let A* use the cost as a first criterion,
and the lower bound on the number of bends as described in this section as a tiebreaker. Then, of all minimum-cost routes one with the minimum number of bends will be found.

Proof. By contradiction. Let us say that a path $p_1$ has been chosen but that another path $p_2$ with equal cost but a lower number of bends exists.

When node $t$ is extracted from the heap some node $n \in p_2$ must still be in the heap (possibly a neighbor of the source node). Since $t$ is extracted it must have a) the lowest cost estimate of all nodes in the heap, and b) the lowest bends number of all nodes with lowest cost in the heap. Since $n \in p_2$ both its cost estimate can not be more than the cost of $p_1$ and its bend number must be lower than the number of bends in $p_1$. This is a contradiction with the fact that $t$ was extracted.

Calculation of the bends number

Although the approach is similar to cost propagation, bend propagation is a bit more complicated as illustrated by Fig. 7.21. Cost propagation was discussed before and only information from the predecessor of a node is needed. Refer to Fig. 7.21, left. From a cost-perspective, nodes $o_1$ and $o_2$ are may be equal but from a bends-perspective they are not: expanding to $o_1$ causes an additional bend. This can only be seen by keeping track of the direction of expansion, i.e. looking back to the predecessor of the predecessor of a node.

Note that this is still a constant-time operation that does not add to the worst-case time complexity of the algorithm.

In Gravet an exact lower bound on the number of bends between a node and the target node is used. The routing area is divided in six regions relative to the node and the direction of the path leading to that node. In Fig. 7.21, right the regions are marked with roman numbers and the grayscale corresponds to the minimum number of bends needed to reach a target node in that region (also shown). The best case is if $A^*$ is expanding exactly in its direction (0 future bends). The worst case is if $A^*$ is expanding in the opposite direction (3 future bends). In order to explain the latter it must be noted that due to strict positive costs $A^*$ yields simple paths.

Note that using the previous direction of expansion makes a large difference. A bends estimate based only on the position of a node relative to its target may lead to a large underestimation of the actual number of bends needed. If the direction is not taken into

---

13This is somewhat comparable to the octal partitioning of Chapter 4.
14A simple path is a path in which no nodes appear more than once.
account regions I and IV in Fig. 7.21 are equivalent. These are precisely the two regions with minimum and maximum bends estimates, respectively. Underestimating the bends number has a negative impact on run time since A* will explore many more paths than necessary.

### 7.5.7 Tiebreaker random

Some effective methods for optimization are based on randomization[119]. The idea explored in this section is only loosely related to those ideas. If the router is stuck in some suboptimal configuration a bit of randomization can help to get out of it. Essentially, if a wire is rerouted differently during R&R this may trigger a chain-reaction causing many more wires to shift resulting in lower congestion.

Consider the example in Fig. 7.22. The wires are routed and rerouted in the order \([w_0, w_1, w_2]\). Consider the case where \(w_0\) is rerouted. The router needs to choose between the two configurations on the left and in the center which both cause the same amount of congestion. If A* chooses the configuration with a bend on the lower-right (left), this overflow cannot be resolved by rerouting one of the other wires. If on the other hand A* chooses the other configuration, the overflow can be resolved by rerouting \(w_1\) as shown on the right.

A* does not have any reason to prefer the upper-left configuration over the bottom-right configuration. Given the deterministic implementation of A* the default choice is likely the same throughout R&R. If in such a situation randomization is used to break the tie between the two configurations it is expected that the optimal solution is found.

### Implementation

Tiebreaker random is implemented by assigning a random value to each node before each (re)route round. Simply using randomization in the compare function of the heap is not possible because this could invalidate the heap order of the nodes already in the heap.

### 7.5.8 Distance to destination

When two nodes have equal costs run time may be improved if the node that is closest to the destination node is expanded first. There are two reasons for this.

1. Less expansions are necessary to reach the target node.  

---

**Figure 7.22**: Random as a tiebreaker.
2. It is less likely that the heuristic function was overly optimistic. Smart implementations of A* usually have somewhat similar behavior. In case of a tie the node that was added last to the heap is expanded first. Essentially this steers A* in the direction of the target more in depth first-like fashion. As long as no additional cost (compared to the lower bound estimate) is encountered A* expands in the direction of the target. Despite the above, having this tiebreaker explicitly in the system is useful for two reasons.

1. A node closest to the target may not be unique. In this case, tiebreaker random may be used after distance to destination.

2. In the presence of congestion, the node added last to the heap is not necessarily the one closest to the target node.

Tiebreaker \textit{dist2dest} is implemented in \texttt{Gravet}. It simply uses the $L_1$ distance to the target as a tiebreaker.

### 7.5.9 Experimental results

In Tables 7.9-7.12 the routing tiebreakers \textit{bends} and \textit{true freedom} are evaluated, respectively. In the second column the results for the reference tiebreaker \textit{random} are reported. The third and fourth columns contain the results for \textit{bends} and \textit{true freedom} as tiebreakers, respectively.

#### Overflow and congestion

Table 7.9 shows that the routing tiebreakers have little influence on the overflow. Congestion is absent or very low and the number of congested edges is a better measure for routability. From Table 7.10 it is clear that by using low bends first as a routing tiebreaker the routing quality is improved. Compared to a random routing tiebreaker the number of congested edges is reduced by 5%. The reason why reducing bends improves the congestion picture is because of the interaction with wavefront expansion as explained in Section 7.6.3.

The true freedom tiebreakers can be characterized as greedy tiebreakers since they do not have a global view. Compared to the other tiebreakers they perform rather poorly on congestion-related criteria. The results with tiebreaker random indicate that there are indeed local minima and randomization helps to get out of them.

#### Bends

Table 7.11 shows that using bends as a tiebreaker is very effective. On average the number of excess bends is reduced by 73% compared to a randomized tiebreaker. The remaining excess edges were apparently necessary to avoid overflow. Destroying true freedom also helps to reduce the number of bends, but not as effectively as using bends directly. This illustrates the importance of Theorem 7.11. The reason for the sub-optimality of tiebreaker true freedom is the presence of congestion. It can be seen from the table that using random as a tiebreaker yields routes with a very high number of bends: closer to the maximum number of bends that is possible than to the minimum number of bends. This
experiment clearly shows that although randomization may reduce the overflow in some cases it should be used with care since the number of vias may become problematic.

### Run time

Routing tiebreakers have a large impact on run time as shown in Table 7.12. On average, there is almost a factor two between the best and worst performing tiebreaker. For individual cases the difference can be a factor five. Especially the easy design ibm05 is very sensitive on this metric. The results indicate that randomization should be used with care. Randomization is a somewhat undirected effort and many optimization algorithms based on randomization are notoriously slow.

Perhaps surprisingly, the best run times are achieved with the low true freedom first tiebreaker. This is the result of monotonicity of detour-free paths. Essentially, with this tiebreaker the A* search results in a depth-first search without regard of e.g. the bends that
result.

As a result of congestion bend minimization does cost some run time compared to the other tiebreakers. This is the case because the bends tiebreaker finds the absolute minimum number of bends for a wire. Whenever congestion necessitates a bend, A* tries (potentially many) different routes.

**Randomized tiebreaker in combination with other tiebreakers**

In the previous sections it was demonstrated how randomization is used to optimize overflow and congestion. Unfortunately, the improvements come at the price of run time. Tiebreaker dist2dest on the other hand has a somewhat opposite behavior: it is good for run time but in terms of overflow and congestion sub-optimal.

In this experiment a combination of the tiebreakers dist2dest and random is evaluated. The results are shown in Table 7.13. The two extreme cases from the benchmark suite are

### Table 7.11: Excess bends with different routing tiebreakers.

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>bends</th>
<th>t.freed.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>lo1st / hi1st</td>
<td>lo1st / hi1st</td>
</tr>
<tr>
<td>ibm01</td>
<td>7967</td>
<td>4069 / 11152</td>
<td>6988 / 10639</td>
</tr>
<tr>
<td>ibm02</td>
<td>17255</td>
<td>5252 / 25287</td>
<td>12098 / 23228</td>
</tr>
<tr>
<td>ibm03</td>
<td>15572</td>
<td>3480 / 23848</td>
<td>8267 / 22390</td>
</tr>
<tr>
<td>ibm04</td>
<td>19289</td>
<td>6246 / 28797</td>
<td>12886 / 26687</td>
</tr>
<tr>
<td>ibm05</td>
<td>34524</td>
<td>1017 / 58998</td>
<td>2414 / 57784</td>
</tr>
<tr>
<td>ibm06</td>
<td>25989</td>
<td>6444 / 39051</td>
<td>14576 / 36545</td>
</tr>
<tr>
<td>ibm07</td>
<td>34931</td>
<td>6689 / 53601</td>
<td>15271 / 49920</td>
</tr>
<tr>
<td>ibm08</td>
<td>35520</td>
<td>6958 / 53814</td>
<td>16386 / 49541</td>
</tr>
<tr>
<td>ibm09</td>
<td>37204</td>
<td>10376 / 56856</td>
<td>23146 / 52480</td>
</tr>
<tr>
<td>ibm10</td>
<td>55927</td>
<td>7740 / 88302</td>
<td>26590 / 84243</td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>27 / 153</td>
<td>54 / 143</td>
</tr>
</tbody>
</table>

### Table 7.12: Run time [s] with different routing tiebreakers.

<table>
<thead>
<tr>
<th>design</th>
<th>rand</th>
<th>bends</th>
<th>t.freed.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>lo1st / hi1st</td>
<td>lo1st / hi1st</td>
</tr>
<tr>
<td>ibm01</td>
<td>64</td>
<td>50 / 38</td>
<td>35 / 31</td>
</tr>
<tr>
<td>ibm02</td>
<td>89</td>
<td>68 / 58</td>
<td>54 / 56</td>
</tr>
<tr>
<td>ibm03</td>
<td>82</td>
<td>58 / 51</td>
<td>43 / 47</td>
</tr>
<tr>
<td>ibm04</td>
<td>227</td>
<td>180 / 132</td>
<td>117 / 121</td>
</tr>
<tr>
<td>ibm05</td>
<td>108</td>
<td>21 / 21</td>
<td>22 / 38</td>
</tr>
<tr>
<td>ibm06</td>
<td>156</td>
<td>115 / 108</td>
<td>97 / 96</td>
</tr>
<tr>
<td>ibm07</td>
<td>199</td>
<td>149 / 124</td>
<td>108 / 115</td>
</tr>
<tr>
<td>ibm08</td>
<td>165</td>
<td>116 / 105</td>
<td>95 / 100</td>
</tr>
<tr>
<td>ibm09</td>
<td>275</td>
<td>246 / 206</td>
<td>175 / 166</td>
</tr>
<tr>
<td>ibm10</td>
<td>405</td>
<td>266 / 216</td>
<td>185 / 196</td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>70 / 59</td>
<td>52 / 55</td>
</tr>
</tbody>
</table>
Table 7.13: Results on several criteria for ten runs on two different benchmarks. Tiebreaker random is used in two different ways.

<table>
<thead>
<tr>
<th>tiebreakers</th>
<th>dist2dest random</th>
<th>random</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>overflow C&gt;0.9 cpu [s]</td>
<td>overflow C&gt;0.9 cpu [s]</td>
</tr>
<tr>
<td>ibm01</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>23 2715 58</td>
<td>20 2672 65</td>
</tr>
<tr>
<td></td>
<td>32 2736 60</td>
<td>22 2620 67</td>
</tr>
<tr>
<td></td>
<td>22 2734 59</td>
<td>30 2710 63</td>
</tr>
<tr>
<td></td>
<td>19 2736 57</td>
<td>24 2675 63</td>
</tr>
<tr>
<td></td>
<td>22 2741 59</td>
<td>31 2572 65</td>
</tr>
<tr>
<td></td>
<td>19 2704 58</td>
<td>24 2714 62</td>
</tr>
<tr>
<td></td>
<td>20 2710 64</td>
<td>23 2685 65</td>
</tr>
<tr>
<td></td>
<td>24 2748 59</td>
<td>19 2693 59</td>
</tr>
<tr>
<td></td>
<td>22 2722 59</td>
<td>18 2721 60</td>
</tr>
<tr>
<td></td>
<td>21 2685 61</td>
<td>21 2637 61</td>
</tr>
<tr>
<td>ibm05</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 466 22</td>
<td>0 368 111</td>
</tr>
<tr>
<td></td>
<td>0 481 22</td>
<td>0 385 109</td>
</tr>
<tr>
<td></td>
<td>0 466 22</td>
<td>0 391 111</td>
</tr>
<tr>
<td></td>
<td>0 463 22</td>
<td>0 399 109</td>
</tr>
<tr>
<td></td>
<td>0 429 22</td>
<td>0 367 110</td>
</tr>
<tr>
<td></td>
<td>0 441 22</td>
<td>0 361 109</td>
</tr>
<tr>
<td></td>
<td>0 464 22</td>
<td>0 367 109</td>
</tr>
<tr>
<td></td>
<td>0 460 22</td>
<td>0 407 109</td>
</tr>
<tr>
<td></td>
<td>0 475 22</td>
<td>0 364 109</td>
</tr>
<tr>
<td></td>
<td>0 437 22</td>
<td>0 384 109</td>
</tr>
</tbody>
</table>

used in the experiment. Ibm01 is a very difficult design while ibm05 on the other hand is a relatively easy design. In the first main column random is used as a tiebreaker in combination with dist2dest. In the second main column only random is used as a tiebreaker. For each case the results of ten runs are shown.

For the difficult case the overflow varies between 18 and 31, indicating that several runs with randomization can be used to remove overflow if run time is not an issue. Note that the overflow of Grawet without randomization is 31 for the same settings (not shown in the table). Using tiebreaker dist2dest additionally to random has a relatively small impact. Congestion becomes a bit better on average and run time a bit worse. Such small differences may not be important and are not conclusive evidence for the superiority of either approach.

For the easy benchmark the overflow is always zero. The number of congested edges is consistently lower for the purely random case. The biggest difference however is in the run time. Using dist2dest as a tiebreaker results in a run time five times lower. Evidently,
Table 7.14: Easy benchmarks: Congested edges with different routing tiebreakers.

<table>
<thead>
<tr>
<th>design</th>
<th>random</th>
<th>bends lo1st / hi1st</th>
<th>truefreedom lo1st / hi1st</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01*</td>
<td>1081</td>
<td>1088 / 1429</td>
<td>1328 / 1379</td>
</tr>
<tr>
<td>ibm02*</td>
<td>1995</td>
<td>1835 / 2416</td>
<td>2258 / 2398</td>
</tr>
<tr>
<td>ibm03*</td>
<td>1502</td>
<td>1396 / 1744</td>
<td>1632 / 1807</td>
</tr>
<tr>
<td>ibm04*</td>
<td>2152</td>
<td>2014 / 2610</td>
<td>2515 / 2652</td>
</tr>
<tr>
<td>ibm05*</td>
<td>253</td>
<td>155 / 377</td>
<td>346 / 348</td>
</tr>
<tr>
<td>ibm06*</td>
<td>2565</td>
<td>2380 / 2982</td>
<td>2850 / 2878</td>
</tr>
<tr>
<td>ibm07*</td>
<td>2470</td>
<td>2302 / 2935</td>
<td>2710 / 2934</td>
</tr>
<tr>
<td>ibm08*</td>
<td>3108</td>
<td>2918 / 3845</td>
<td>3603 / 3739</td>
</tr>
<tr>
<td>ibm09*</td>
<td>3063</td>
<td>3188 / 4313</td>
<td>3988 / 4095</td>
</tr>
<tr>
<td>ibm10*</td>
<td>3390</td>
<td>3119 / 4245</td>
<td>4012 / 3992</td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>92 / 126</td>
<td>119 / 123</td>
</tr>
</tbody>
</table>

this is attractive. Note that the benchmarks in the suite are difficult and ibm05 perhaps is more representative for average designs.

The main conclusion from this experiment is that for really difficult routing problems dist2dest should not be used as a tiebreaker. The congestion prediction methods described earlier in this thesis can be used to decide on this. For routing problems of average difficulty however this tiebreaker should be used because it has such a big impact on run time.

Results for easy designs

Since not all designs in practice are very difficult the router was tested with different tiebreakers on the set of easy benchmarks (refer to Chapter 7.4.3). All designs were routed without overflow except one (with an overflow of two) illustrating that the designs are easier than the original suite, but not unrealistically easy.

Tables 7.14-7.16 show the results for different metrics. On the congestion metric low bends first performs best again. Compared to the difficult cases the relative performance has improved even further. For the easy designs bend minimization does not yield a significant overhead in terms of run time. Since many routing resources are available the router is usually able to complete the first excess bend free path it tries in many cases. Although there are abundant routing resources, some excess bends are still used as a result of congestion. The low bends first tiebreaker is very efficient in bend removal. Only twelve percent of the excess bends are left after routing. The next best tiebreakers require almost four times as many excess bends.

7.6 Wavefront expansion

The steiner tree algorithm discussed in Chapter 4 optimizes steiner tree length which is the minimum length with which a net can be routed. As long as each wire is routed with
minimum length the net as a whole is routed with minimum length. In practice detouring due to congestion cannot be avoided. During steiner tree decomposition it is not known which wires are detoured and it is also not known how the detoured wires are routed.

Wavefront expansion as proposed in this section minimizes the impact of detouring and is based on the following observations.

Observation 7.14 (Path sharing). (Part of) the detour of a wire can be used by another wire of the same net. In practice this observation can be used in two ways:

1. A wire can use an already routed wire as part of its detour.

2. The detour of a wire can be used by another wire of the same net if it is routed later.

Both ways can be used to avoid either additional detour and/or overflow or congestion.
Wavefront expansion adds steiner points to a net to exploit the above observations. The resulting steiner tree may no longer be optimal according to the original steiner tree criterion, but given congestion and the order in which the wires are routed it optimizes the steiner decomposition for routability.

### 7.6.1 Examples

Fig. 7.23 shows how previously routed wires can be used to improve a steiner tree. On the left a wire needs to detour due to congestion. It is possible to route the wire without overflow at the cost of a detour. By using part of an existing wire the amount of detour is minimized\(^ {15}\).

The center situation is somewhat similar to the first situation. In this situation overflow (congestion) cannot be avoided at all unless the existing wire is used.

The situation on the right-hand side shows the opposite case: a wire can be routed without congestion or detour. However, by using part of an existing detoured wire the amount of wire needed by the net as a whole is minimized.

Note that in all the above cases the net as a whole is still longer than a minimum-length implementation. The amount of detour (in terms of wire, not path lengths) is relatively low due to the addition of new steiner points at locations the steiner tree algorithm could not predict.

### 7.6.2 Using pseudo-edges to model potential steiner points

In \texttt{Gravet} each node of a routed wire is a potential steiner point for subsequent wires of the same net. Once a wire has been routed there is no cost associated with going from the wire’s nodes to the nodes on the routed path for another wire of the same net. This can be modeled as zero-length edges in the routing graph.

Fig. 7.24 illustrates the principle. Wire \(w = (n_1, n_2)\) is the wire that is to be routed, and \(n_1\) is the source for \(A^\ast\). The wire \((n_0, n_1)\) belongs to the same net and has been routed before (with a detour in this case). \textit{Each of the nodes on the path from }\(n_0\) to \(n_1\) \textit{is a potential}

\(^{15}\) In this case the path length does not improve.
During wavefront expansion potential steiner points are modeled by adding pseudo edges to the grid.

In order to model this a pseudo edge (shown as a dotted line) from $n_1$ to each potential steiner point is added to the grid graph. The cost associated with this pseudo edge is 0. Now $A^*$ is run as usual. When the path is traced back from $n_2$, a steiner node is added whenever a pseudo edge is encountered.

Using zero-cost pseudo-edges between random nodes in the grid graph could potentially break the optimality of $A^*$ since the estimated costs between a node in the heap and the target node based on distances in the grid graph is no longer guaranteed to be a lower bound: a pseudo-edge may serve as a shortcut. However, the described method only adds pseudo-edges between the source node and other nodes and optimality is still guaranteed as stated in the following theorem.

**Theorem 7.15** (Optimality of $A^*$ with wavefront expansion). $A^*$ still yields optimal paths if pseudo-edges are added to the routing graph due to wavefront expansion.

**Proof.** If a pseudo-edge connects the source and target nodes of the wire, the theorem is trivially true. Otherwise, by contradiction.

Consider a wire $w = (s,t)$ with $s$ and $t$ the source and target of an $A^*$ search respectively. Consider an optimal path $p_1 = s \rightarrow t$ and consider the path $p_2$ found by $A^*$ with some higher cost. Now consider the largest sub-path $p_s = n \rightarrow t$ of $p_1$, such that $p_s$ does not contain any pseudo-edges (possibly, $n = s$). Obviously, the distance-based cost estimates of all nodes on the sub-path are lower bounds on the actual costs and strict lower bounds on the cost of path $p_2$. It is easy to see that $n$ must either be $s$ or one of the nodes reached from $s$ through a pseudo-edge. Since $s$ must have been expanded $n$ must have been in the heap. It follows that at any time during the algorithm some node on the sub-path must be in the heap. All these nodes have strict lower bounds on the cost of the path actually found and this is a contradiction with the fact that $t$ was expanded when its cost estimate was based on path $p_2$.

The set of nodes in the heap is often called the wavefront. In Dijkstra-style shortest-path algorithms the search is like a wave: a circle of increasing radius. By adding shortcut edges to the grid graph the search wave becomes wider, hence the name wavefront expansion.

### 7.6.3 Experimental results

Wavefront expansion makes a large difference as illustrated by Table 7.17. Without wavefront expansion `Gravet` is already quite effective in comparison with other tools but the
Table 7.17: Results with and without wavefront expansion.

<table>
<thead>
<tr>
<th>expansion</th>
<th>overflow</th>
<th>cpu [s]</th>
<th>excess bends</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>off</td>
<td>on</td>
<td>off</td>
</tr>
<tr>
<td>ibm01</td>
<td>161</td>
<td>30</td>
<td>56</td>
</tr>
<tr>
<td>ibm02</td>
<td>28</td>
<td>0</td>
<td>121</td>
</tr>
<tr>
<td>ibm03</td>
<td>1</td>
<td>0</td>
<td>58</td>
</tr>
<tr>
<td>ibm04</td>
<td>382</td>
<td>111</td>
<td>211</td>
</tr>
<tr>
<td>ibm05</td>
<td>0</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>ibm06</td>
<td>9</td>
<td>0</td>
<td>126</td>
</tr>
<tr>
<td>ibm07</td>
<td>42</td>
<td>0</td>
<td>196</td>
</tr>
<tr>
<td>ibm08</td>
<td>44</td>
<td>3</td>
<td>117</td>
</tr>
<tr>
<td>ibm09</td>
<td>2</td>
<td>0</td>
<td>260</td>
</tr>
<tr>
<td>ibm10</td>
<td>21</td>
<td>0</td>
<td>295</td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>89</td>
<td>100</td>
</tr>
</tbody>
</table>

second main column shows that wavefront expansion still makes a big difference. With wavefront expansion on the overflow improves for all designs that could not be routed without overflow without wavefront expansion.

Although wavefront expansion increases the size of the search space this larger search space does not lead to larger run time. In fact, run time is reduced by 11\%\(^{16}\). The run time reduction stems from the fact that wavefront expansion leads to less wire length and less congestion.

The number of bends is significantly reduced by almost 25% because of wavefront expansion. As explained in the next section this somewhat unexpected phenomenon is the result of the interplay between wavefront expansion and bend reduction.

Interplay between wavefront expansion and bend reduction

Bend reduction through the tie breaking mechanism and wavefront expansion cooperate to improve both the number of bends and congestion. Using wavefront expansion allows the addition of steiner points in order to avoid bends while on the other hand bend minimization makes wavefront expansion more effective.

In Table 7.17 in some cases the number of excess bends is negative: the number of bends is lower than predicted by the number of bends in the steiner tree. This can only happen because the steiner topology is changed due to wavefront expansion. Fig. 7.25 shows an example from the benchmark suite. The hatched lines depict the steiner tree decomposition of length eight and two bends. The bold lines denote the routing solution as found by Grawet. This solution has the minimum net length of eight, and uses only one bend because a steiner point was added.

From our experiments it became apparent that bend reduction always coincided with congestion reduction. The explanation is that bend reduction makes wavefront expansion more effective as illustrated by Fig. 7.26. Minimum-bend wire realizations are always on the boundary of their wire box. Roughly speaking, the more bends, the more a wire is

\(^{16}\)Note that these results can be improved upon by letting Grawet exit as soon as an overflow-free solution has been found. In order to allow for a fair comparison this option has been switched off in this experiment.
7.7 Improving the detour distribution

The detour distribution is an important quality metric of a global routing result. In sequential routing detours and therefore the detour distribution is largely determined by the order in which the wires are routed. During initial routing a wire may be forced to detour due to previously routed wires. When these wires were routed they were oblivious to the fact that they were using a scarce routing resource. There may have been a realization using other resources, but at the time the wires were routed this was not known. Thus, the detour of wires that are routed later is often unnecessary or unnecessary large. Note that it is extremely unlikely that there exists a wire ordering that avoids this problem altogether.

The conventional way of dealing with this problem is changing the cost function during R&R. During initial routing the focus is on wire length and the cost due to congestion is relatively low. The resulting overflow is resolved during R&R by increasing the cost associated with congestion.

The above method requires a lot of tuning and is not always effective. Manipulating the cost function is a somewhat fuzzy technique without transparency to the user. It is not practical to ask users to manipulate the cost functions used internally in the router. These cost functions are typically highly tuned and also in many cases considered a trade secret. In Graadet a technique called detour bounding is used instead. As the name suggests, this is a bound on the length of wires. This influences the detour distribution more directly and
gives the designer more direct control. The technique is used in FaDGloR (Chapter 6.3.8) and is also effective in Grawet. Both run time and detour distribution can be improved as shown in the results section.

7.7.1 Implementation

The implementation of detour bounding is simple. Besides the cost the router also keeps track of the distance that is traveled. Similar to a cost estimate a path length estimate is known for a node when it is added to the heap. If this path length estimate exceeds the detour bound the node is not added to the heap.

Detour bounding is not equivalent to cost bounding. Users have the ability to change the cost function Grawet uses and little can be assumed about it. As stated in the following theorem, detour bounding does not affect the optimality of the algorithm.

**Theorem 7.16 (Optimality of A* with detour bounding).** Consider a wire \( w = (s, t) \) with \( s \) and \( t \) the source and target nodes of an A* search. Let there be a detour bound \( b \geq 0 \). Then, of all paths between \( s \) and \( t \) with a detour of at most \( b \), A* will return a minimum-cost path.

**Proof.** We need to prove the following.

1. The algorithm returns a path.
   This is evidently true. Because \( b \geq 0 \), some path always exists and A* will find one.

2. The path does not have a detour exceeding \( b \).
   By contradiction. Consider the opposite. If A* terminates by expanding \( t \), \( t \) must have been added to the heap. This contradicts with the fact that the length of the path exceeds \( b \).

3. The path is optimal.
   By contradiction. Consider the opposite: some other path with lower cost and a detour \( \leq b \) exists. Then some node of that path must be in the heap. Since cost estimates are still lower bounds on the actual cost this is a contradiction with the fact that \( t \) must have been expanded from the heap.

Note that contrary to intuition perhaps, detour bounding is not equivalent to the (temporary) removal of a number of edges from the routing graph. A simple idea is to take the wire box of a wire, extend it by half the allowed detour and define a routing problem on the associated sub-graph. This does not necessarily yield a correct path: wires have plenty of room to detour within their wire box. In some cases A* will therefore return with a path exceeding the allowed detour.

7.7.2 Time slack and detour bounds

Although there is not a one-to-one relation between delay due to a wire and its length a first-order estimate can be made [38].

\[
D_{wire} \propto R \cdot C = r l \cdot cl = r c l^2, \tag{7.7}
\]
where $R$ and $C$ are the resistance and capacitance of the wire and $r$ and $c$ are material and technology-dependent constants per unit length, respectively. This model is known as Elmore delay\[^{38}\].

Consider the usual case that required arrival times are known for the pins at the memory elements or I/O circuits when the global router is run. It is possible to estimate wire delay based on the above delay model and the distance between pins. The difference between an estimate and the required arrival times is called time slack. This slack can be used to account for detours. Global routing results can be used to refine arrival times and time slacks.

Slack is usually defined on signal paths rather than on individual nets or wires. Slack budgeting is outside the scope of this thesis but we do note the following.

1. Usually critical paths are known. It is easy to force a detour bound of zero on all nets on these paths.

2. The amount of length slack assigned to a wire may be an interesting criterion for wire ordering. Wires that are routed later are more likely to detour so it is probably a good idea to route the wires with most slack last.

3. Under conventional delay models the same amount of time slack yields more detour slack on short wires than it does on long wires due to non-linear delay.

4. Slack budgeting is not a simple problem due to the huge number of paths on a chip\[^{17}\]. Buffering strategies can also make time estimation difficult. Methods such as \[^{140}\] optimize the delay of modules under fixed wire delay. The method is based on a timing graph and yields arrival times at the pins of the modules. Similar methods can be used to find the time slacks of wires under fixed standard cell delay.

5. In physical synthesis global routers have an integrated timing engine. Detours can e.g. be solved by logic restructuring or gate sizing during routing. If this kind of timing information is available detour bounds are also useful.

### 7.7.3 Experimental results

Fig.\[^{7.27}\] shows the effectiveness of detour bounding in combination with a little randomization on the detour distribution of ibm03. This is one of the easier benchmarks in the benchmark suite and a detour bound of 8 is used. Due to wavefront expansion odd and negative detours exist. The number of wires with an odd or negative detour illustrates the impact of wavefront expansion.

For this particular example the maximum detour was lowered from 68 to 8. Since wire delay is roughly quadratic with length such a reduction is important for timing closure. The total detour was lowered from 6609 to 3907. Although not a very important criterion, having less detour indicates that timing closure is easier and that there is more room for ECO\[^{18}\] routing. Additionally, the number of excess bends decreased from 3688 to 3547.

\[^{17}\]It is fair to say that the number of paths grows exponentially with the number of standard cells on a chip.

\[^{18}\]Engineering Change Order. Rerouting of small parts of the design to fix problems that were found out at a very late stage, often after feedback from the production facility.
Impact on run time

In FaDGloR detour bounding is primarily used to control run time. In Grawet run time is also significantly lower with detour bounding. For the above example run time decreased by roughly 30% (17s versus 12s). The router spends most of its time finding paths through congested area. When a wire is routed through a congested region the router spends large amounts of time trying to find a way around it. Often a congestion-free path does not exist and lots of run time is spend for nothing. Restricting the search space for these wires therefore has a large positive impact on run time without impacting congestion too much.

Impact on overflow

By experimenting a bit with the detour bound it was also possible to reduce overflow. The most congested design in the benchmark suite is ibm04. By using detour bounding in combination with some randomization congestion could be reduced from 115 to 105. In addition, run time was reduced from 178s to 134s.

7.8 Comparison against other tools

In this chapter many techniques to improve routing results have been discussed. Table 7.18 shows a direct comparison of Grawet with other tools. Labyrinth is a routing tool published by UCLA and described in [72]. The version that was used in our experi-
Comparison against other tools

Table 7.18: Comparisons against other tools.

<table>
<thead>
<tr>
<th></th>
<th>Labyrinth</th>
<th></th>
<th>Chi</th>
<th></th>
<th>Grawet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>overfl</td>
<td>cpu [s]</td>
<td>overfl</td>
<td>cpu [s]</td>
<td>overfl</td>
</tr>
<tr>
<td>ibm01</td>
<td>242</td>
<td>21</td>
<td>154</td>
<td>27</td>
<td>14</td>
</tr>
<tr>
<td>ibm02</td>
<td>214</td>
<td>33</td>
<td>45</td>
<td>105</td>
<td>0</td>
</tr>
<tr>
<td>ibm03</td>
<td>117</td>
<td>49</td>
<td>0</td>
<td>68</td>
<td>0</td>
</tr>
<tr>
<td>ibm04</td>
<td>786</td>
<td>95</td>
<td>369</td>
<td>106</td>
<td>80</td>
</tr>
<tr>
<td>ibm05</td>
<td>0</td>
<td>33</td>
<td>0</td>
<td>252</td>
<td>0</td>
</tr>
<tr>
<td>ibm06</td>
<td>130</td>
<td>84</td>
<td>12</td>
<td>150</td>
<td>0</td>
</tr>
<tr>
<td>ibm07</td>
<td>407</td>
<td>228</td>
<td>137</td>
<td>248</td>
<td>0</td>
</tr>
<tr>
<td>ibm08</td>
<td>352</td>
<td>208</td>
<td>48</td>
<td>206</td>
<td>2</td>
</tr>
<tr>
<td>ibm09</td>
<td>310</td>
<td>333</td>
<td>12</td>
<td>236</td>
<td>0</td>
</tr>
<tr>
<td>ibm10</td>
<td>288</td>
<td>422</td>
<td>26</td>
<td>400</td>
<td>0</td>
</tr>
<tr>
<td>normal</td>
<td>100</td>
<td>100</td>
<td>22</td>
<td>201</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 7.18 shows that for all benchmarks Chi beats Labyrinth on the overflow criterion. For the smaller benchmarks Labyrinth is faster but Chi has better scalability. For these two reasons Grawet is primarily compared to Chi.

According to Table 7.18 Grawet is superior on all benchmarks on the overflow criterion. A little overflow is typically solvable for a detailed router but as much overflow as produced by Chi on e.g. design ibm04 inevitably leads to routability problems. In this table Grawet produces detour-free results for 7 out of 10 benchmarks. By changing the settings slightly

---

19 We did not change the algorithms but only the implementation since this was easy to do and it is fair to compare Grawet to an implementation of decent quality.

20 Labyrinth is mainly faster due to our improvements. In previous publications Chi was faster.
Table 7.19: Results for faster Grawet.

<table>
<thead>
<tr>
<th>design</th>
<th>overflow</th>
<th>run time [s]</th>
<th>design</th>
<th>overflow</th>
<th>run time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm01</td>
<td>67</td>
<td>7</td>
<td>ibm06</td>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>ibm02</td>
<td>0</td>
<td>13</td>
<td>ibm07</td>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>ibm03</td>
<td>0</td>
<td>8</td>
<td>ibm08</td>
<td>18</td>
<td>46</td>
</tr>
<tr>
<td>ibm04</td>
<td>227</td>
<td>20</td>
<td>ibm09</td>
<td>0</td>
<td>60</td>
</tr>
<tr>
<td>ibm05</td>
<td>0</td>
<td>5</td>
<td>ibm10</td>
<td>7</td>
<td>104</td>
</tr>
</tbody>
</table>

it was also possible to remove overflow for ibm08. We speculate that the congestion distribution of Chi is also inferior since Chi appears to produce slightly longer detours (not shown).

Although the overflow numbers are too low to draw reliable conclusions in terms of percentages it is clear that Grawet is by far the most effective tool when it comes to overflow removal.

**Run time and speedup of Grawet**

On all designs Grawet is faster than Chi except for ibm01 and ibm04. These are difficult designs and Grawet invests more time in the removal of overflow leading to a large reduction and increased routability for the detailed router. The proposed router quits after removal of overflow and on most other designs this leads to far lower run times than the other tools.

Grawet has several command line options to speed up the routing, possibly at the cost of solution quality. Table 7.19 shows the results for Grawet with only half the default number of R&R rounds (8) and a fairly arbitrary detour bound of 20. In comparison with the results of Chi the overflow is still lower on all designs. Also on the run time criterion, Grawet is superior to both Labyrinth and Chi. Especially the run time for the difficult designs has been decreased since Grawet invests less run time in overflow removal. It has been our choice however to focus on routability and therefore these settings are not used in the other experiments. This experiment shows that no matter what tradeoff is made, Grawet produces superior results compared to both Chi and Labyrinth.

### 7.9 Conclusions and discussion

In this chapter, the main conclusions from the theory and experiments are summarized. A number of possible extensions and improvements are proposed.

#### 7.9.1 Main conclusions

The developed router Grawet has been shown to be superior to other academic routers on a set of widely used global routing benchmarks. Although previous research indicates that net ordering does not have any effect on the results of sequential global routing, our experiments indicate to the contrary that wire ordering is important. Grawet uses wire
7.9 Conclusions and discussion

ordering based on the true freedom or length of wires. Since overflow numbers are statistically small it is not possible to draw final conclusions based on our experiments. Nonetheless, the experiments show that our wire ordering scheme consistently outperforms other wire ordering schemes on metrics such as run time, congestion and especially bends. This conclusion holds for both difficult designs and relatively easy designs.

The second main improvement over existing methods is a tie breaking mechanism during A* expansion. By evaluating the number of bends as a secondary criterion after the cost function the total number of bends after routing is dramatically decreased. It is proven that this method yields the best result for individual wires and experiments show that in practice the total number of bends is dramatically reduced. As a result of the additional consideration additional run time is needed, so this may be a tradeoff that needs to be made. The tiebreaker dist2dest is used to speed up the algorithm. Randomization can also be used as a second or third tiebreaker in order to prevent the algorithm getting stuck in local minima. This can be used to improve routing results especially if run time is available.

Wavefront expansion is a method that allows the router to dynamically insert steiner points depending on the congestion picture. It is very effective in minimizing all important criteria: overflow, run time and bends. A somewhat surprising result is that the combination of bend minimization and wavefront expansion is very effective on both the overflow and bends criteria.

7.9.2 Design-specific tuning

During experimentation with Grawet it was found that the overflow criterion is relatively sensitive to implementation details and settings such as wire sorters and A* tiebreakers. For all designs with overflow better results were found at some point in time than reported in this thesis. Unfortunately this required design-specific tuning of the tool. In practice it may be worthwhile to tweak router settings for a specific design. Possibly, there are ways to deduce such settings from design characteristics, but the obvious things have explored and not found to be successful.

7.9.3 Extension to 3D model with vias

Although Grawet was developed for the 2D global routing model an extension to a 3D model with layer assignment is relatively straightforward. It must be noted however that the number of nodes and edges in the routing graph grows rapidly and the expected run time of the router is expected to grow even faster.

It is easy to replace bends in a 2D model with vias in a 3D model in the tiebreaker scheme. Alternatively, it is possible to assign a cost to via edges. There is an additional consideration however. Sequential routing techniques are essential greedy: they greedily choose the best route for one wire at a time. The cost model is such that other wires are somewhat taken into account. If vias are used as a tiebreaker this means in practice that layers are filled up from the bottom pins are in the bottom layers since they need to connect to the active layers. If the total amount of vias is to be minimized this means the short wires need to be in the bottom layers. Therefore, instead of using a wire ordering based e.g. on true freedom a wire ordering based on length is more appropriate (possibly
7.9.4 Routing after layer assignment

In some design flows (partial) layer assignment takes place before global routing. Wires are assigned to tiers of layers, i.e. one horizontal and one vertical layer. Such an approach yields a 2D routing problem for each tier similar to the routing problems solved by Grawet. The idea is that more time critical wires are assigned to the faster (higher) layers. Also, long wires are usually assigned to higher layers since they have few pins relative to their length.

The most important difference with the Grawet approach is that tier capacities are much lower than the capacities encountered in the benchmarks in this chapter. The statistics are therefore a bit different. There is for example a greater likelihood that relatively few wires block other wires. In these circumstances a freedom-based approach such as Grawet employs is potentially more successful than other approaches.

7.9.5 Detour bounding of individual wires

Detour bounding as presented in this chapter is effective on improving the detour distribution. The same detour bound is imposed on all wires. Obviously, it is possible to impose detour bounds on individual wires. If a relatively small number of paths cause timing problems it is easy to impose very strict detour bounds on the involved wires. This may cause congestion problems during initial routing but during R&R other wires with less strict detour bounds should be rerouted. In modern design flows typically many paths are critical\(^2^1\). In such flows there is often iteration between global routing and logic synthesis techniques. This changes the criticality of paths on the fly and a general detour bound may be the best choice. If a number of wires with slack can be determined these can be given somewhat looser detour bounds.

\(^{21}\) Performance is typically used as a constraint and e.g. power is optimized.
Chapter 8

Concluding remarks

Congestion is a fundamental problem during the design of integrated circuits. It is clearly associated with routing, but since design steps such as placement and even logic synthesis create the input and constraints for routing it is important that congestion is taken into account during such design steps as well. Essentially, congestion-awareness must move up in the design flow. Methods based on wire length estimates already exist for the earlier design stages but such estimates do not capture the local nature of congestion problems.

In this thesis fast congestion estimation is proposed as a way to feed forward congestion information to placement, floorplanning and logic synthesis algorithms. Obviously, such estimators can not be 100% accurate. Firstly because algorithms and designers react to the congestion threat and secondly because accuracy is traded for speed to keep the estimators practical.

Two congestion estimation algorithms are developed in this thesis. The first method is a probabilistic congestion estimation algorithm that models the design and routers using probabilities. Congestion maps are created by stamping patterns in the map. These stamps are based on the observed behavior of an industrial router. The method is very fast and reasonably accurate but tends to be overly pessimistic.

The second estimation algorithm is a degenerate global router. Compared to a real global router accuracy is sacrificed for speed using a number of heuristics. This method is much more accurate than the probabilistic method, especially in and around the most congested areas. Routability does not average out over a chip but instead the routability of the chip as a whole is determined by the routability of every single wire. Since routability is most questionable in congested areas accuracy is needed in exactly those areas. The improved accuracy only costs relatively little additional run time.

Global routing is the last design step with a global overview over congestion. In this thesis a global router is developed. This router is evaluated on a number of different criteria associated with congestion. The first set of contributions is made on the steiner tree problem. The popular BOI algorithm is improved in time complexity, and the impact it has on the following stage of global routing is evaluated. A number of theoretic results is obtained. Secondly, a large number of heuristics is used within the core routing algorithm. Most of those are based on tiebreaking. Especially the results on bend minimization are noteworthy since this important criterion is largely ignored in the literature. In compari-
son with other academic tools the performance of the router is very good on all relevant metrics.

8.1 Outlook to the future

As manufacturing process technology continues to scale down feature sizes, congestion problems are projected to get worse. Transistors continue to show reduced switching times but delay associated with wires does not decrease. This delay is related to wire cross-resistance and capacitance. In order to improve wire delay wires are typically not scaled in the same way as the transistors. Specifically, the wire cross section is increased by changing the aspect ratio. This decreases resistance. Wire spacing is increased in order to decrease capacitive coupling to ground and surrounding wires. Obviously all this has a negative effect on the amount of available routing resources.

The industry is keeping up with Moore's law. Effectively this means that although the number of transistors on a chip grows exponentially, chip area has remained approximately constant. Even though the length of global interconnect has not changed much in absolute terms, relative to wire dimensions much longer wires exist in more advanced technologies. Additionally, the number of such wires is growing rapidly. An even bigger impact is on the power and clock distribution networks. Due to both increased switching speeds and larger number of standard cells that need to be serviced, the requirements on these networks are increasingly hard to meet. This has resulted in the use of more and more buffers, fat wires and new network topologies that require more wiring. The demand for routing resources is increasing as a result.

In order to address the increasing demand for routing resources additional routing layers are added in new processes. Unfortunately, supply does not grow as quickly as demand. Additionally, due to the fact that the active layers are below the routing layers, via stacks are necessary to be able to use the additional layers. These via stacks cause routing problems in the lower routing layers since they are effectively blockages there. Increasingly more such via stacks are necessary because of buffering strategies. These buffering strategies do not only address super-linear wire delay, but also signal integrity related issues.

8.1.1 Incorporating congestion in the design flow

As a result of the aforementioned trends there is an increasing interest in routing congestion. Recent efforts have lifted congestion-awareness to the level of logic synthesis. In the literature there is an impressive amount of algorithms and methodologies for dealing with the problem, but in practice these methods are still very much a research topic.

What is needed is a comprehensive tool set and methodology including a consistent way of dealing with and representing congestion. Imagine an analogy with timing closure. Until a few years ago users dealt with point tools with inconsistent views on delay and time constraints. Modern physical synthesis tool sets have timing engines that at each stage of the flow can give estimates of delays and slacks. These estimates are available to the algorithms that are run in the flow. This enables the integration of e.g. global routing and logic restructuring with the purpose of delay optimization. During physical synthe-
sis increasingly accurate delay (and area/power) estimates are available to the tools and designer.

Now consider a physical synthesis tool set with a congestion map available to all algorithms. Tile size can be decreased during the flow for better accuracy. Congestion estimation algorithms such as proposed in this thesis update these maps with increasing accuracy based on user parameters and the stage of the flow. Such an approach enables adding congestion as another dimension to for instance physical synthesis. Both timing and congestion are used as constraints while performing logic restructuring and technology mapping. Timing is analyzed based on integrated global routing, extraction and timing engine, while congestion is analyzed using incremental estimation techniques or the router when applicable. Research such as carried out in this thesis is necessary to enable the development of real congestion-aware flows.
References


Summary

Congestion Analysis and Management

Since the first integrated circuits in the late 1950s, the semiconductor industry has enjoyed exponential growth. This observation is illustrated by Moore’s law which states that the number of transistors that can be placed on an integrated circuit (chip) doubles roughly every two years\(^1\). Other examples of approximately exponential behavior are the increase in clock speed at which digital circuits run and the decrease in size of features that can be printed by manufacturing equipment.

Due to improvements in manufacturing technology, the performance of chips is increasingly determined by the lengths of the wires that connect the functional elements of a chip. These wires are realized (routed) very late in the design flow. Mismatch between routing resources and routing demand is known as congestion and can lead to detoured wires and too many vias\(^2\). Such surprises usually have detrimental effect on performance and yield and should be avoided.

In order to prevent routing problems later on, congestion needs to be considered at earlier stages of the design flow than is traditionally the case. Congestion maps display congestion hotspots, and are a means to communicate (potential) resource conflicts. In this thesis, two methods for fast estimation of such maps are proposed. The first uses a probabilistic approach, and the second is essentially a global router with focus on speed rather than overflow minimization. Using such methods, algorithms earlier in the design flow can take routing considerations into account without the need to actually invoke the routers, allowing for more effective and accurate decisions and optimizations.

Many algorithms require that multi-pin connections are broken down into two-pin connections. This thesis introduces the concept of routing freedom for measuring the amount of flexibility in such connections and decompositions. In the face of uncertainty about what will happen later on in the design flow, it can be beneficial to create as much flexibility as possible. Unfortunately, more freedom also corresponds to more vias under certain conditions. This thesis demonstrates how freedom analysis can be used to optimize net decomposition using Steiner tree algorithms and improve global routing.

During chip implementation there are typically multiple considerations. Important examples are congestion, vias and run time. Equivalently, (heuristic) algorithms may base

\(^1\)In his original work, Moore stated that the size at which integrated circuits could be produced at lowest price per transistor doubles every two years. In recent years, Moore’s law has been used to refer to the size of the most complex designs.

\(^2\)Vias connect metal wires on two layers through the electrical insulator separating them. They are undesirable amongst others because they are difficult to manufacture reliably.
decisions on several criteria. In this thesis, *tiebreakers* are used to deal with such situations instead of more traditional approaches based on weights. This approach is used extensively in a net decomposition algorithm and during global routing.

The problems encountered in chip implementation are huge in size and complexity. Often, approaches without guarantee of optimality are therefore used, as is the case for many of the algorithms presented in this thesis. Such approaches need to be validated and compared by benchmarking. Therefore, an extensive software package implementing the ideas presented in this thesis has been developed. A large number of experiments have been conducted demonstrating the effectiveness and limitations of the proposed methods and the tradeoffs that need to be made.
Samenvatting

Congestion Analysis and Management

Sinds de eerste geïntegreerde schakelingen eind jaren vijftig heeft de halfgeleider industrie exponentiële groei gekend. Deze observatie wordt geïllustreerd door de *wet van Moore* die zegt dat het aantal transistoren dat geplaatst kan worden op een geïntegreerde schakeling (chip) ongeveer iedere twee jaar verdubbeld.\(^1\) Andere voorbeelden van bij benadering exponentieel gedrag zijn de toename van de kloksnelheid waarop digitale schakelingen open en de daling van de detailgrootte die nog gedrukt kan worden.


Om latere bedradingproblemen te voorkomen moet tijdens eerdere ontwerpfases dan traditioneel het geval is rekening gehouden worden met congestie. *Congestiekaarten* tonen congestiepunten en vormen een middel om over (potentiële) conflicten tussen vraag en aanbod in bedrading capaciteit te communiceren. In dit proefschrift worden twee methodes voor het snel schatten van dergelijke kaarten voorgesteld. De eerste gebruikt een *probabilistische* aanpak, en de tweede is in wezen een globale bedrader met de focus op *snelheid* in plaats van minimalisering van capaciteit overschrijding. Dankzij dergelijke methoden kunnen algoritmen tijdens eerdere ontwerpfases rekening houden met bedrading overwegingen zonder de bedraders daarop te hoeven roepen. Dit maakt effectievere en accuratere beslissingen en optimalisaties mogelijk.

Voor veel algoritmen is het nodig dat connecties bestaande uit meerdere pinnen in twee-pins connecties opgesplitst worden. Dit proefschrift introduceert het concept *bedrading vrijheid* voor het kwantificeren van hoeveelheid flexibiliteit in dergelijke connecties en splitsingen. In het licht van de onzekerheid over wat er tijdens latere ontwerpfases zal gebeuren, kan het nuttig zijn zoveel mogelijk flexibiliteit te creëren. Helaas is meer vrijheid in sommige omstandigheden gerelateerd aan meer vias. Dit proefschrift toont hoe

---

\(^1\) In zijn oorspronkelijke werk verklaarde Moore dat de grootte waarbij geïntegreerde schakelingen voor de *laagste prijs per transistor* geproduceerde konden worden elke twee jaar verdubbeld. Recentere wordt de *wet van Moore* gebruikt om te refereren aan de grootte van de meest complexe onterpen.

\(^2\) Vias verbinden draden van twee lagen door de elektrische isolator die hen scheidt. Zij zijn ongewenst onder andere omdat het moeilijk is hen betrouwbaar te fabriceren.
vrijheidsanalyse gebruikt kan worden om het splitsen van netten door middel van *Steiner tree* algoritmes te optimaliseren en de globale bedrading te verbeteren.

Gedurende het ontwikkelen van chips zijn er meestal meerdere overwegingen. Belangrijke voorbeelden hiervan zijn congestie, vias en snelheid van de algoritmen. Op vergelijkbare wijze kunnen (heuristische) algoritmen hun beslissingen baseren op meerdere criteria. In dit proefschrift worden *gelijkspelbeslissers* gebruikt om om te gaan met dergelijke situaties in plaats van een meer traditionele aanpak op basis van gewichten. Deze aanpak wordt extensief gebruikt in een netsplitsingsalgoritme en gedurende globale bedrading.

De problemen die men tegenkomt tijdens chipontwikkeling zijn gigantisch in grootte en complexiteit. Daarom moet vaak een aanpak die geen optimaliteit garandeert gekozen worden, zoals ook het geval is bij veel van de algoritmen die in dit proefschrift worden gepresenteerd. Een dergelijke aanpak moet gevalideerd en vergeleken worden op basis van testontwerpen. Daarom is een uitgebreid softwarepakket geschreven op basis van de ideeën in dit proefschrift. Een groot aantal experimenten is gedaan en de resultaten tonen de effectiviteit en beperkingen van de voorgestelde methodes en de afruil tussen verschillende criteria die soms gedaan moet worden.
About the author

Jurjen Westra was born on January 9, 1978 in Voorschoten, the Netherlands. From 1996 he studied Electrical Engineering at Delft University of Technology. There, he developed an interest for both the theoretical and practical sides of algorithms and computation. His final work in the Circuits and Systems group of professor Otten was developing algorithms for placing macros of a chip in a way suitable for the wireplanning flow. He graduated with a Master’s degree early 2001.

After graduation, Jurjen worked for three months as a summer intern at Magma Design Automation, Cupertino, CA on pin assignment, resulting in a successful demo at the Design Automation Conference 2001.

During the summer of 2001 Jurjen joined the Electronic Systems group at the Electrical Engineering faculty of Eindhoven University of Technology as a PhD student under professor Groeneveld, working on various topics in electronic design automation. Initially, he continued working on wireplanning, but gradually his interests shifted towards routing and specifically congestion. He has also coached a number of students and served on several graduation committees.

In late 2005 Jurjen joined Takumi Technology, Eindhoven to work in the field of Design for Manufacturability—analyzing chip designs for potential problems during manufacturing and fixing those. Specifically, he contributed to Takumi’s mask inspection analysis and layout rating software.

Since the beginning of 2007 Jurjen works at Fugro-Jason in the oil and gas industry. He works on software for modeling the sub-surface based on geostatistical inversion techniques. This software is used to delineate and develop oil and gas fields.

List of publications


