Design Rules in VLSI Routing

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Christian Schulte

aus

Bonn

Bonn, Juni 2012

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Jens Vygen
2. Gutachter: Prof. Dr. Dr. h.c. Bernhard Korte

Tag der Promotion: 7.8.2012

Erscheinungsjahr: 2012

Acknowledgments

At this place I want to thank my supervisors Professor Dr. Bernhard Korte and Professor Dr. Jens Vygen for their support over all these years and the perfect working conditions in the Research Institute for Discrete Mathematics at the University of Bonn.

I also feel grateful to all my friendly colleagues at the institute, especially to the former and present members of the routing team: Michael Gester, Dr. Dirk Müller, Jun.-Prof. Dr. Tim Nieberg, Christian Panten, and Dr. Sven Peyer. Without them this thesis and the overall success of BonnRoute would not have been possible. I especially thank them for the many helpful discussions, for proofreading many parts of this thesis, and for covering for me while I was busy writing.

I thank all the people at IBM which I have worked with, especially Karsten Muuss, Dr. Sven Peyer, and Dr. Gustavo Tellez. Together we managed to resolve many tedious problems in practice to get the overall project running.

Finally, I want to express my gratitude to my family and friends, which helped me a lot to recover from the many long working days that were necessary to complete this thesis.

Contents

1 Introduction **1**
- 1.1 Routing . . . 2
- 1.2 BonnRoute . . . 4

2 Handling Design Rules **7**
- 2.1 Basic Definitions . . . 8
- 2.2 Design Rules . . . 11
  - 2.2.1 Background . . . 11
  - 2.2.2 Distance Rules . . . 13
  - 2.2.3 Same Net Rules . . . 16
  - 2.2.4 DPT Design Rules . . . 17
- 2.3 The BonnRouteRules Module . . . 18
  - 2.3.1 BonnRoute Wiring Representation . . . 18
  - 2.3.2 Generating Wire Types . . . 24
  - 2.3.3 Generating Shape Classes . . . 29
  - 2.3.4 Handling Line End Minimum Distance Rules . . . 33
  - 2.3.5 Generating Shape Class Minimum Distance Rules . . . 36
  - 2.3.6 Reducing the Number of Shape Classes . . . 41
  - 2.3.7 Runtime Analysis . . . 42
  - 2.3.8 Further Aspects . . . 43
  - 2.3.9 Implementation . . . 45
  - 2.3.10 Experimental Results . . . 49
  - 2.3.11 Outlook: Handling DPT Design Rules . . . 53
- 2.4 Checking Distance Rules . . . 54
  - 2.4.1 General Concept . . . 55
  - 2.4.2 Shape Data Structures . . . 55
  - 2.4.3 The Checking Module . . . 61
- 2.5 Handling Same Net Rules . . . 61
  - 2.5.1 Pin Access . . . 61
  - 2.5.2 Postprocessing . . . 67

3 BonnRoute in Practice **69**
- 3.1 Combined Routing Flow . . . 69
- 3.2 Experimental Results . . . 70

Bibliography **75**

Summary **80**

1 Introduction

VLSI[1] design is the process of creating the logical and physical representation of highly integrated circuits, which consist of millions of transistors. Because most underlying mathematical problems are extremely hard, and instance sizes occurring in practice are huge, the design of today's chips cannot be done without automated tools using sophisticated algorithms.

VLSI design starts with *logical design*, which first specifies the desired logical function of a chip using a hardware description language like VHDL (IEEE [1994]). This specification then is mapped to a set of *circuits*, which are part of a given *library*, and a *netlist*. A netlist partitions the set of all pins into *nets* such that all pins in the same net have to be connected. The library contains standard circuits realizing elementary boolean functions like $\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}$ etc., as well as *macro circuits* realizing more complex modules like adders.

The second part of VLSI design is *physical design*, which generally is divided into placement, timing optimization, clock network design, and routing. In the *placement* step circuits are placed on the chip area such that they are disjoint and certain objectives are optimized to ensure that the subsequent physical design steps can be realized well. The positions of circuits and their pins for example naturally impose a lower bound on the total wiring length needed to connect all nets (net length). Since placement is done early in the physical design flow, good estimations on the outcome of later steps are needed to optimize these objectives efficiently. Brenner et al. [2008] and Struzyna [2010] describe in detail how placement can be realized well in practice.

The *timing optimization* step deals with optimization of the timing behavior of the chip and ensures that all required signal arrival times are met. Timing can be influenced for example by exchanging circuits with logically equivalent ones having different electrical properties, or by demanding different kinds of wires for certain connections in the routing step. A detailed overview of timing optimization is given by Held [2008].

The *clock network design* step determines how clock signals are propagated from clock generation circuits to different components of the chip which have to be synchronized, e.g. storage elements. Arrival time bounds are considered and objectives such as power consumption optimized. Chu and Pan [2009] give an overview of the basics of clock network design. An extensive discussion of designing clock networks using trees is given by Maßberg [2009].

Finally, in the *routing* step a set of wires connecting the pins of each net is computed. Wires of different nets need to be disjoint and many complex technology dependent design

[1] Very Large Scale Integration

rules have to be satisfied. There are several optimization objectives to consider including total wire length. Since this thesis focuses on routing, we will go into details in section 1.1.

Each of these physical design steps is covered by the *BonnTools* (Korte et al. [2007]), a software package developed at the Research Institute for Discrete Mathematics at the University of Bonn in cooperation with IBM.

In this thesis we present efficient methods to handle design rules in VLSI routing. Due to increased lithographical challenges in the manufacturing process of chips with feature sizes of 32 nm and below, design rules have become more and more complex. Therefore, it has become very difficult for automatic routing tools to produce results with sufficiently low numbers of design rule violations. As any remaining violation basically needs to be fixed manually by the designers, this is, however, a mandatory task for any router used in practice. We describe in detail how this is achieved for BonnRoute, the routing part of the BonnTools. The main result is a new module of BonnRoute, called *BonnRouteRules*, computing a design rule representation that can be used efficiently in the core algorithms and data structures of BonnRoute.

We proceed as follows: After introducing the routing problem and the main components of BonnRoute, we give an introduction into design rules in section 2.2. The main part then is section 2.3, where we describe the BonnRoute wiring and distance rule representation and explain in detail how a given set of design rules can be mapped to this model. We also cover how this representation is used efficiently in data structures of BonnRoute in section 2.4. Finally, in chapter 3 we present experimental results of BonnRoute on current real world designs. We show that BonnRoute is able to route chips of modern technologies very well in practice. The approaches developed in this thesis played a key role in achieving this.

1.1 Routing

Routing is the last major step in the physical design flow. Formally and in its most basic form it can be defined as follows:

SIMPLIFIED VLSI ROUTING PROBLEM

Instance: An undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{N}$, a set N of nets with pins $P(n) \subset V$ for each $n \in N$.

Task: For each $n \in N$, find a Steiner tree $T_n = (V(T_n), E(T_n))$ in G which connects $P(n)$ and is vertex disjoint from all $T_{n'}, n' \in N \setminus \{n\}$ such that $\sum_{n \in N} \sum_{e \in E(T_n)} w(e)$ is minimized.

Even in this simple form the routing problem already contains NP-hard problems like the vertex disjoint paths problem (Kramer and van Leeuwen [1984]). In addition to disjointness, in practice there are many restrictions on the wiring of a net by a given set of design rules, which we will describe in section 2.2. Also note that besides minimizing

total wire length, there are many other (partly conflicting) optimization goals that can be considered. Properties like power consumption, signal delay, and production yield are greatly influenced by routing: For example densely packed wires running in parallel for a long distance increase coupling capacitance and therefore signal delay and power consumption. Also from a production yield point of view wires that are packed less densely often are beneficial, although net length may increase. Some Steiner tree topologies and long detours in critical nets can cause bad timing results and can make an entire routing unusable. Achieving timing closure, i.e. obtaining a routed design satisfying all signal arrival time constraints, often is an iterative process where several physical design steps including routing have to be iterated.

Moreover the instance sizes that occur in practice can be enormous. Often millions of connections in a graph with billions of vertices have to be computed within a few hours. Therefore, the routing problem is typically solved in two steps: *Global routing* and *detailed routing*.

In Global routing Steiner trees are computed on a much coarser grid graph while respecting edge capacity constraints to avoid congestion. Generally it can be solved much faster than detailed routing, e.g. by considering it as a resource sharing problem (Müller [2009]). The result basically is a corridor for each net where the actual connections have to be realized in detailed routing. Since this limits the search area for connections drastically, one obtains a significant speed up of detailed routing. A key point for successful detailed routing is that in global routing the available routing space and its usage was estimated accurately and congestion avoided successfully. As global routing to a large extent already determines the topology of the Steiner tree of each net, it is an important step in optimizing several routing objectives.

Detailed routing determines the actual wiring within the global routing corridors. Instead of directly computing a Steiner tree connecting the pins of each net, most routing tools iteratively connect two different connected components by shortest paths until the whole net is connected. Although this does not necessarily lead to Steiner trees of minimum length, it works very well in practice. Because for some nets there may be large distances to cover, many routing tools use a technique called switch-box routing. The global routing corridor is divided further into cells, and connections are obtained by computing and concatenating multiple point-to-point connections within these cells (Hitchcock [1969]).

Another approach to cover long distances efficiently is to use a *track assignment* step between detailed routing and global routing. In such a step basically a net ordering within the global routing corridors is computed, see e.g. Chang and Cong [2001] and Batterywala et al. [2002] for details. While this offers possibilities to take properties like electrical interference (crosstalk) between long neighboring wires into account, one certainly looses the flexibility that a path search algorithm has.

Generally, one can distinguish between *gridded* and *gridless* detailed routing. Gridded routers restrict themselves to a grid graph and use the shortest path algorithm of Dijkstra [1959] or variants of it. In newer technologies at least for pin access one actually needs

a gridless approach. In gridless routing one considers a set of rectilinear obstacles and solves a shortest obstacle avoiding rectilinear path problem (Lee et al. [1996]).

Finally let us note that it is common practice today only to use wires running parallel to the x- or y-axis (*Manhatten routing*). Throughout this thesis we restrict ourselves to this case. There are, however, some works discussing the benefits of diagonal wires, also called X architecture. See for example Teig [2002], Chen et al. [2003], and Ho et al. [2005]. The downside of gridless routing is that wires not aligned to regular grid-like structures often cannot be packed efficiently and therefore waste routing space.

1.2 BonnRoute

BonnRoute is the routing tool of the BonnTools, a software package for VLSI physical design developed at the Research Institute for Discrete Mathematics at the University of Bonn in cooperation with IBM. It consists of a global routing and a detailed routing part. The global router, mainly developed by Müller [2009], is based on a very general resource sharing approach and is able to optimize various different objectives like wiring length, power consumption, and manufacturing yield. It generates provably near-optimum fractional solutions, applies randomized rounding to obtain integrality, and resolves resulting local congestion with rip-up and reroute techniques. The used algorithms are well parallelized and make the global router extremely fast in practice, even on largest designs.

The detailed router of BonnRoute builds Steiner trees by successively connecting different connected components of each net by a shortest path within the global routing corridors. Most connections are computed by a very fast, interval-based variant of the shortest path algorithm of Dijkstra [1959]. It was originally proposed by Hetzel [1998] and further generalized by Peyer et al. [2009] and Humpola [2009]. Being supported by fast routing space data structures, it is able to cover even very long distances efficiently by labeling whole intervals instead of nodes and using a future cost similar to the A^* heuristic of Hart et al. [1968]. This path search works on a grid-like graph, called track graph, which ensures that wires can be packed well, and therefore is called *on-track path search*. Local conflicts between paths of different nets are resolved by a standard rip-up and reroute approach.

For pin access, the smaller feature sizes and complex design rules of modern technologies make an additional, gridless approach necessary. In BonnRoute pin access paths are precomputed, and their endpoints are used as source and target points for the on-track path search. In particular, design rule violations and local conflicts between pin access paths are avoided by construction. This involves solving a shortest path problem with minimum segment length restrictions, which is done by a variant of Dijkstra's algorithm working on an extended Hanan grid (Nieberg [2011]). We will discuss some aspects of pin access in section 2.5.1.

Note that in contrast to many other routers, BonnRoute does not contain a track assignment step and does not do switch-box routing. Even connections over very long distances

can be found efficiently by the on-track path search and do not require such steps. BonnRoute can optimize objectives like manufacturing yield without using track assignment: Yield can be optimized in global routing, as well as in postprocessing steps in detailed routing (Schulte [2006], Bickford et al. [2006]). A more detailed overview of the main components of BonnRoute is given by Gester et al. [2012].

A recently added part of BonnRoute, the BonnRouteRules module, generates an appropriate model of the complex design rules of modern technologies such that the efficiency of the core algorithms and data structures in BonnRoute is preserved. We will cover this in detail as a main part of this thesis in section 2.3.

2 Handling Design Rules

A solution to the detailed routing problem must fulfill several *design rules* in order to be actually usable in practice. Disjointness of the Steiner trees connecting each net is not sufficient at all. In modern technologies there are increasingly complex spacing requirements that must be obeyed by the wiring of different nets or even parts of the same net. Moreover, there are various restrictions on the geometry of wire shapes. The reason behind most of these design rules is to avoid problems in the lithographic production process. Before a chip can be released to manufacturing it must pass a design rule check (DRC), i.e. it is not allowed to contain any violation of any design rule. Detailed routing tools which leave too many of such *DRC errors* are barely usable in practice, because fixing DRC errors manually can be a huge amount of tedious work.

The increasing complexity of design rules, and their impact on automatic routing tools have been discussed in some works, see e.g. Kahng [2003], Gupta and Kahng [2003], Peyer [2007], and Cho et al. [2009]. An in-depth discussion how to handle such rules, however, does currently not exist to the best of our knowledge. Most related work strongly focuses on general manufacturing aware routing. This comprises routing techniques and post-optimization steps trying to minimize certain types of production errors. A large interest is currently on the new challenges imposed by the upcoming *double patterning technologies* (DPT), see e.g. Tang and Cho [2011] and Ghaida et al. [2011]. We describe the new kinds of design rules occurring in these technologies in section 2.2.4 and give an outlook on how they can be handled in BonnRoute in section 2.3.11.

Our primary goal in this chapter is to show how BonnRoute is able to satisfy the most important design rules of current technologies efficiently such that the resulting routing is clean enough to be usable in practice. We first focus on distance rules, give an overview of the reasoning behind them, and define the most important types needed in later sections formally. We then describe the general concept how wires and minimum distance requirements are represented in BonnRoute. The main part then consist of showing how the given design rules are mapped to this model. This conversion is the task of a new module called *BonnRouteRules* developed by the author to enable BonnRoute for 32 nm technologies and beyond. After that we finish this chapter by describing how this model is actually used in BonnRoute. We propose a new data structure for locating routing shapes efficiently, and discuss how minimum distance requirements are checked.

2.1 Basic Definitions

We start with some basic definitions needed for the later discussions.

Definition 2.1. *We use a three-dimensional cartesian coordinate system as the* base coordinate system *in BonnRoute. The* chip area *is a nonempty rectangular cuboid*

$$\mathcal{A} := [x_{min}, x_{max}] \times [y_{min}, y_{max}] \times [p_{min}, p_{max}]$$

where $x_{min}, x_{max}, y_{min}, y_{max}, p_{min}, p_{max} \in \mathbb{Z}$ *and* p_{min}, p_{max} *even.*

Let $P := \{p_{min}, \dots, p_{max}\}$ *be the set of* planes *and* $P_{wiring} := \{p \in P : p \text{ even}\}$ *and* $P_{via} := \{p \in P : p \text{ odd}\}$ *the set of* wiring *and* via planes*, respectively. For* $p \in P$ *define* $\mathcal{A}_p := \{(x, y, p) \in \mathcal{A}\}$ *as the chip area on plane* p*.*

Each wiring plane $p \in P_{wiring}$ has a *preferred direction* which is — since we restrict ourselves to Manhatten Routing — either parallel to the x- or y-axis, i.e. horizontal or vertical, denoted by $\mathrm{dir}(p) = \mathrm{hor}$ and $\mathrm{dir}(p) = \mathrm{ver}$, respectively. To use routing space efficiently, most wires run in the preferred direction of their plane. The few and usually short wires which are running against this direction are called *jogs*. Preferred directions of adjacent wiring planes typically are orthogonal to each other for several reasons. First, this reduces the risk of electrical interference (*crosstalk*) between close long parallel wires on adjacent planes. Second, it reduces the number of connections between two adjacent wiring planes (*vias*) needed to run in orthogonal direction without using a jog.

Besides the base coordinate system there is the *track coordinate system* in BonnRoute.

Definition 2.2. *For each wiring plane* $p \in P_{wiring}$ *we have a non-empty set of* track coordinates $T_p = \{t_p^1, \dots, t_p^{|T_p|}\}$ *and for convenience define* $T_p := \emptyset$ *for all* $p \notin P$*. Assume that* p *has horizontal preferred direction, the vertical case is defined analogously. We require that* $y_{min} \le t_p^1 < \dots < t_p^{|T_p|} \le y_{max}$ *and call each set in* $\{[x_{min}, x_{max}] \times t : t \in T_p\}$ *a* track *on plane* p*. The set* $Q_p := T_{p-2} \cup T_{p+2} = \{q_p^1, \dots, q_p^{|Q_p|}\}$ *is non-empty if* p *has at least one neighboring wiring plane, and its elements are called* points of interest*. A point* $(i, j, p) \in \{1, \dots, |T_p|\} \times \{1, \dots, |Q_p|\} \times \{p\}$ *in the track coordinate system corresponds to the point* $b(i, j, p) = (q_p^j, t_p^i, p)$ *in the base coordinate system. We also call* $b(i, j, p)$ *an* on-grid *point.*

We say that we have uniform tracks *with* pitch $d_p \in \mathbb{N}$ *on plane* p *if and only if* $t_p^{i+1} - t_p^i = d_p$ *for all* $i = 1, \dots, |T_p| - 1$*.*

An example of the track coordinate system is shown in figure 2.1.

Almost all of the wires generated by BonnRoute will run on tracks. This allows efficient packing of wires and avoids many design rule violations by construction. Generally, off-track wiring will only be used to access certain pins, which we cover in section 2.5.1.

Definition 2.3. *For closed sets* $A, A' \subseteq \mathbb{R}^2$*, we define*

$$\mathrm{dist}(A, A') := \min \{\|a - a'\|_2 : a \in A, a' \in A'\}.$$

Moreover we denote the interior *of* A *by* A°*.*

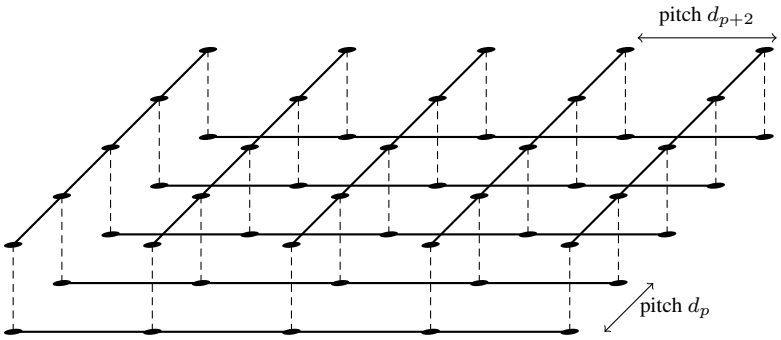


Figure 2.1: Tracks on wiring planes p and $p+2$ (black lines) and the resulting on-grid points (black dots).

The space occupied by objects relevant for detailed routing, i.e. wires, pins and blockages, can be represented by a set of *shapes*.

Definition 2.4. *A* shape *is a 6-tuple* $s = (x_1, y_1, x_2, y_2, p, c)$ *with* $x_1, y_1, x_2, y_2 \in \mathbb{Z}$, $p \in P$, $c \in \mathbb{N}$ *defining an axis parallel rectangle* $A(s) := [x_1, x_2] \times [y_1, y_2] \subset \mathbb{R}^2$. *We call* c *the* shape class *of* s, *and* s *a* shape on plane p.

Remark. As minimum distance rules in BonnRoute will be defined between shape classes, the set of shapes with the same shape class builds an equivalence class in the sense that all of them have the same spacing requirements to other shapes. We will discuss this in detail in section 2.3.

Definition 2.5. *Let* $s = (x_1, x_2, y_1, y_2, c, p)$, s' *be two shapes on plane* p. *Let* S, S' *be sets of shapes, and* $d \in \{\text{north}, \text{east}, \text{south}, \text{west}\}$. *We define:*

(i) $x_1(s) := x_1,\ y_1(s) := y_1,\ x_2(s) := x_2,\ y_2(s) := y_2,\ p(s) := p,\ c(s) := c.$

(ii) $|s|_{\text{hor}} := |x_2(s) - x_1(s)|,\ |s|_{\text{ver}} := |y_2(s) - y_1(s)|$

(iii) $I_{\text{hor}}(s) := [x_1(s), x_2(s)],\ I_{\text{ver}}(s) := [y_1(s), y_2(s)]$

(iv) $$\text{edge}(s, d) := \begin{cases} \{(x, y) \in A(s) : y = y_2(s)\} & \text{if } d = \text{north} \\ \{(x, y) \in A(s) : x = x_2(s)\} & \text{if } d = \text{east} \\ \{(x, y) \in A(s) : y = y_1(s)\} & \text{if } d = \text{south} \\ \{(x, y) \in A(s) : x = x_1(s)\} & \text{if } d = \text{west} \end{cases}$$

(v) $A(S) := \bigcup_{r \in S} A(r)$

(vi) $\text{dist}(s, s') := \text{dist}(A(s), A(s'))$

(vii) $\text{dist}(S, S') := \text{dist}(A(S), A(S'))$

(viii) s, s' intersect *if and only if* $A(s) \cap A(s') \neq \emptyset$.

Definition 2.6. *Given an axis parallel line segment* l *connecting two points* (x_1, y_1, p_1), *and* $(x_2, y_2, p_2) \in \mathbb{Z} \times \mathbb{Z} \times P$, *and a shape* s *with* $p_1 \leq p(s) \leq p_2$, *we define the shape*

$$l + s := (x_1 + x_1(s), y_1 + y_1(s), x_2 + x_2(s), y_2 + y_2(s), p(s), c(s)).$$

We say that l *is* running in direction *x, y, or z if it is parallel to the* x, y, z*-axis, respectively. Analogously we call a line segment* l' *connecting two points in* $\mathbb{R}^2$ horizontal *or* vertical *if* l' *is parallel to the x, or y axis, respectively. We denote the length of such line segments* l, l' *by* $\text{length}(l), \text{length}(l')$, *respectively.*

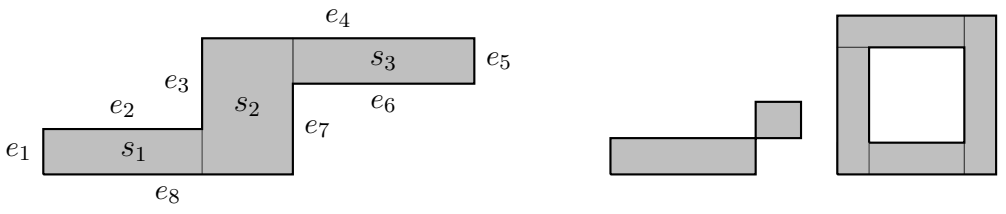
Definition 2.7. *A set of shapes* S *is* connected *if and only if* $A(S)$ *is a connected set.*

Definition 2.8. *We define a* rectilinear polygon *as a finite sequence* $e_1, \ldots, e_n$ *of alternating horizontal and vertical line-segments (*edges*) only intersecting at their endpoints (*vertices*) which bounds a connected set in* $\mathbb{R}^2$. *If the pairs of intersecting edges are exactly the pairs* (e_i, e_{i+1}) *for* $i = 1, \ldots, n-1$ *and* (e_1, e_n), *we have a* simple *rectilinear polygon. A vertex is called* convex *if it is incident to exactly two edges and their inside angle is* $90°$, *otherwise it is called* concave.

If for a set of shapes S *the area* $A(S)$ *is bounded by a simple rectilinear polygon we denote this polygon by* $\text{plg}(S)$ *and say that it is* built *by* S. *For a direction* $d \in \{\text{north}, \text{east}, \text{south}, \text{west}\}$ *we define*

$$\text{plg}(S)_d := \left\{ e \in \text{plg}(S) : \exists S_e \subseteq S \text{ with } e \subseteq \bigcup_{s \in S_e} \text{edge}(s, d) \right\}.$$

Figure 2.2 shows an example.



(a) (b)

Figure 2.2: In (a) the figure shows a set of shapes $S = \{s_1, s_2, s_3\}$ (gray) with the polygon $\text{plg}(S) = \{e_1, \ldots, e_8\}$, where $\text{plg}(S)_{\text{north}} = \{e_2, e_4\}$, $\text{plg}(S)_{\text{east}} = \{e_5, e_7\}$, $\text{plg}(S)_{\text{south}} = \{e_6, e_8\}$, and $\text{plg}(S)_{\text{west}} = \{e_1, e_3\}$. In (b) two polygons are shown which are not simple.

Remark. Rectilinear polygons which are not simple, i.e. where also non-consecutive edges can intersect at their endpoints, and where consecutive edges can be disjoint as in figure 2.2 (b), rarely occur in practice because they often violate certain design rules. We therefore restrict our selves in this work to simple rectilinear polygons built by a set of shapes for the sake of clarity.

We call a shape representing a wire we a *wire shape*. *Vias*, i.e. wires connecting two adjacent wiring planes, consist of one shape per intersected plane, i.e. a *via bottom shape*, *via cut shape*, and *via top shape*.

Definition 2.9. *A* via definition *is a 3-tuple of shapes* $(v_{\text{bot}}, v_{\text{cut}}, v_{\text{top}})$ *on planes* $p-1 \in P_{wiring}, p \in P_{via}$*, and* $p+1 \in P_{wiring}$*, respectively. Given a line segment* v *connecting two points* $(x, y, p-1), (x, y, p+1) \in \mathbb{Z} \times \mathbb{Z} \times \{p-1, p-2\}$*, a via definition defines the* bottom shape $v + v_{\text{bot}}$*, the* cut shape $v + v_{\text{cut}}$*, and the* top shape $v + v_{\text{top}}$.

Pins and blockages may consist of several *pin shapes*, and *blockage shapes*, respectively.

2.2 Design Rules

2.2.1 Background

Most design rules are caused by limitations in the lithographic production process of integrated circuits. This process is basically comprised of the following steps. First a mask is produced consisting of a flat piece of quartz with opaque regions corresponding to the desired layout on some specific plane but several times larger. A projection printer produces a miniaturized image of the mask, using it like a negative in conventional photography. For this process a wafer, typically made of silicon, coated with a photosensitive polymer (photoresist), is precisely aligned to the mask. The wafer is exposed to light through the transparent regions of the mask, and parts of the polymer react and change their solubility. These parts are then removed, and the remaining parts of photoresist correspond to the desired layout. Subsequent production steps like material deposition and etching only are effective on the parts of the waver uncovered from the photoresist. This process then is iterated to map each plane of the desired integrated circuit on the wafer. See e.g. Schellenberg [2009] for a more detailed description. Figure 2.3 gives an optical impression of the result.

Figure 2.3 already shows that the produced structures with their rounded corners differ from the originally intended, purely rectilinear design. In fact the current manufacturing techniques are not able to produce an exact image, there always are several distortions. A significant problem is that, for over ten years now, the feature sizes are much smaller than the actual 193 nm wavelength used for their manufacturing. This increases distortions considerably, and therefore clearly reduces the amount of actually functioning integrated circuits at the end of the production process (yield). Already for feature sizes below

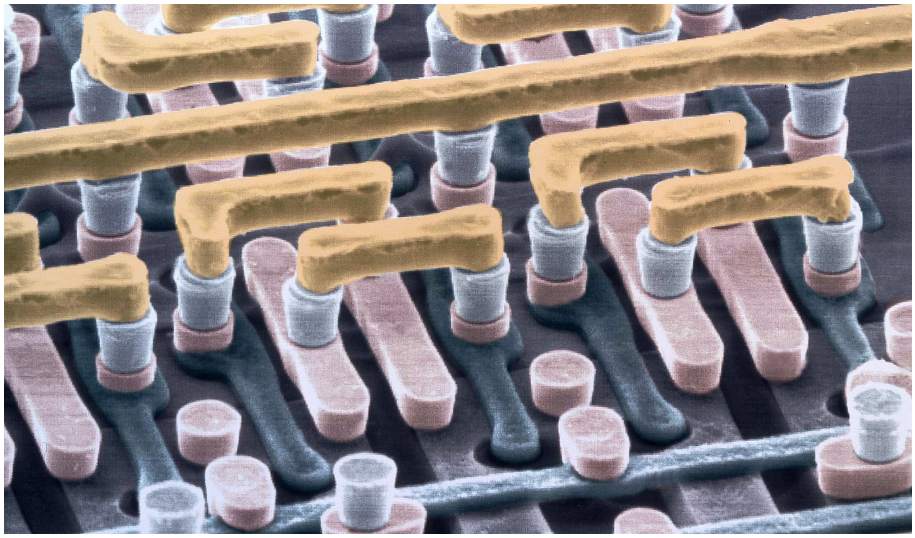


Figure 2.3: Part of a real chip viewed by an electron microscope (with artificial colors). One can see wires and pins on lower planes connected by vias (light blue). (Picture adapted from Peyer [2007])

100 nm, yield would drop to zero without corrective measures. Therefore several resolution enhancement techniques (RET) are used in practice. One technique called "Optical and process correction" (OPC) for example means changing the layout to be produced by anticipating and compensating different kinds of distortions a priori.

As routing is the last major physical design step and determines large portions of the whole layout to be printed, it has to take problems of the production process into account. Patterns that, despite aggressive use of RET, cannot be manufactured properly have to be forbidden. This and the additional space that may be needed for certain OPC operations gives rise to a large set of complex design rules intending to restrict automatic routing tools to lithography friendly routing.

In addition to such *technology design rules* there often are more restrictive *user defined design rules* that only apply to certain nets. For timing critical nets, such as clock nets for example, it is desirable to have wider wires with less resistance and more spacing for long distance connections in order achieve faster signal runtimes. Also nets with high signal switching activities might need larger vias and more spacing to neighboring wires in order to become robust enough and not influence other nets. Therefore in practice each net has its individual set of design rules that apply to it which may differ from the ones of other nets. Note that since all design rules have to be satisfied simultaneously, determining the minimum required distance between the wiring of two different nets for example requires inspecting the rules of *both* nets. Moreover most design rules refer to specific planes, because the manufacturing or electrical properties of these can be different. Typically objects on higher planes are larger and the technology design rules are less complex compared to lower planes.

2.2.2 Distance Rules

We first focus on design rules which specify the minimum distance required between the wiring of two different nets. In the following we describe the most important types of such distance rules occurring in current technologies.

On wiring planes minimum distance requirements depend on *width* and *run length*:

Definition 2.10. *Given a set of shapes S and a point $q \in A(S)$, we define the* width $\mathrm{width}(S,q)$ *of S at q as the maximum edge length of a square contained in $A(S)$ covering q, i.e.* $\max\{|r|_{\mathrm{hor}} : r$ *shape with* $|r|_{\mathrm{hor}} = |r|_{\mathrm{ver}}$ *and* $q \in A(r) \subseteq A(S)\}$.

Note that if $S = \{s\}$ for some shape s we have $\mathrm{width}(S,q) = \min\{|s|_{\mathrm{hor}}, |s|_{\mathrm{ver}}\}$ at every point $q \in A(S)$. We then simply define $\mathrm{width}(s) := \min\{|s|_{\mathrm{hor}}, |s|_{\mathrm{ver}}\}$.

Definition 2.11. *Consider two closed sets $A, A' \in \mathbb{R}^2$ and their projections to the x-axis $A_x, A'_x \in \mathbb{R}$. The* horizontal run length $\mathrm{rl}_x(A,A')$ *of A and A' is the maximum length of an interval in $A_x \cap A'_x$ if this set is non-empty, otherwise we define* $\mathrm{rl}_x(A,A') := -1$.

The vertical run length $\mathrm{rl}_y(A,A')$ *is defined analogously and most of the time we simply speak of the* run length $\mathrm{rl}(A,A') := \max\{\mathrm{rl}_x(A,A'), \mathrm{rl}_y(A,A')\}$ *of A and A'.*

We often only distinguish between non-positive and positive run length which we denote by $l_{\leq 0}$ and l_+, respectively. Figure 2.4 shows an example.

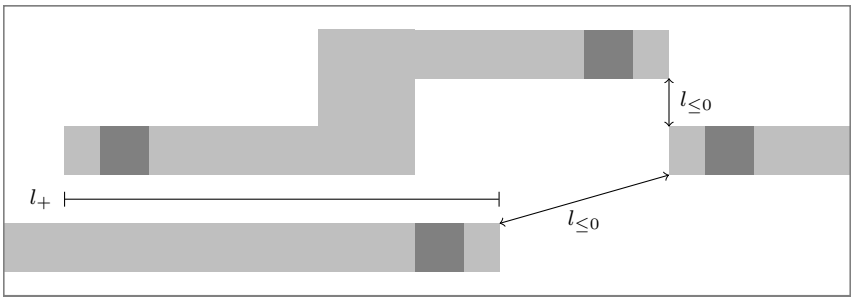


Figure 2.4: Horizontal run length for typical sets of wire and via shapes on some wiring plane (light gray). To indicate via positions the dark gray shapes show the via cut shapes on adjacent via planes.

Instead of a continuous function of widths and run length a minimum distance rule in practice is specified in the following restricted form:

Definition 2.12. *Let $\mathcal{W} = \{W_1, \ldots, W_{|\mathcal{W}|}\}$, $\mathcal{L} = \{L_1, \ldots, L_{|\mathcal{L}|}\}$ be finite sets of disjoint intervals partitioning $\mathbb{N}$ and $\mathbb{Z}_{\geq -1}$, respectively, and*

$$D := \{d_{ijk} \in \mathbb{N} : i,j \in \{1,\ldots,|\mathcal{W}|\}, k \in \{1,\ldots,|\mathcal{L}|\}\}\,.$$

Define the indicator function of $I \in \{\mathcal{W} \cup \mathcal{L}\}$ as

$$X_I(x) := \begin{cases} 1 & \text{if } x \in I \\ 0 & \text{otherwise.} \end{cases}$$

A minimum distance rule *is a step function* $\delta : \mathbb{N} \times \mathbb{N} \times \mathbb{Z}_{\geq -1} \to D$*:*

$$\delta(w_1, w_2, l) := \sum_{i=1}^{|\mathcal{W}|} \sum_{j=1}^{|\mathcal{W}|} \sum_{k=1}^{|\mathcal{L}|} d_{ijk} X_{W_i}(w_1) X_{W_j}(w_2) X_{L_k}(l)$$

with the following properties:

(i) $\delta(w_1, w_2, l) = \delta(w_2, w_1, l)$ *for all* $w_1, w_2 \in \mathbb{N}$

(ii) δ *is nondecreasing in each of its arguments.*

We refer to $\mathcal{W}, \mathcal{L}$ *as the set of* width intervals *and* run length intervals *of* δ*, respectively. In addition we define for convenience*

$$\delta(w_1, w_2, l_{\leq 0}) := \max_{l \in \{-1, 0\}} \delta(w_1, w_2, l),$$
$$\delta(w_1, w_2, l_{+}) := \max_{l \in \mathbb{N}_{>0}} \delta(w_1, w_2, l).$$

Definition 2.13. *Let* S, S' *each be a connected set of shapes and* δ *a minimum distance rule. We say that* S, S' *violate* δ *if and only if there exist* $w, w' \in \mathbb{N}, l \in \mathbb{Z}_{\geq -1}$ *such that for*

$$A := \{p \in A(S) : \text{width}(p, S) \geq w\}$$
$$A' := \{p \in A(S') : \text{width}(p, S') \geq w'\}.$$

we have that A *and* A' *are non-empty,* $\text{rl}(A, A') \geq l$ *and* $\text{dist}(A, A') < \delta(w, w', l)$.

In practice most minimum distance rules only have a simple run length dependency in the sense that only a change from non-positive to positive run length may change the function value:

Definition 2.14. *A minimum distance rule* δ *has* simple run length dependency *if and only if it has run length intervals* $\mathcal{L} = \{[-1, 0], [1, \infty]\}$*, i.e. we have*

$$\delta(w_1, w_2, l) \neq \delta(w_1, w_2, l') \implies l \leq 0 \wedge l' > 0$$

for all $w_1, w_2 \in \mathbb{N}, l, l' \in \mathbb{Z}_{\geq -1}, l \leq l'$.

Considering the polygon built by a set of shapes, there are more restrictive spacing requirements for certain edges:

Definition 2.15. *Given some technology-dependent constant* $l_{max} \in \mathbb{N}$*, we call an edge of a rectilinear polygon between two convex vertices an* end edge *if its length is less or equal to* l_{max}*. All other edges are called* side edges*.*

Definition 2.16. *A* line end minimum distance rule *is a function* $\delta^{le} : \{\text{end}, \text{side}\} \to \mathbb{N}$ *with the property that* $\delta^{le}(\text{side}) \leq \delta^{le}(\text{end})$.

Consider two sets of shapes S, S' *with* $A(S) \cap A(S') = \emptyset$ *building rectilinear polygons* $\text{plg}(S)$, $\text{plg}(S')$, *and let* $t \in \{\text{end}, \text{side}\}$ *and* $d, d' \in \{\text{north}, \text{east}, \text{south}, \text{west}\}$. *An* end *edge* $e \in \text{plg}(S)_d$ *and a* t *edge* $e' \in \text{plg}(S')_{d'}$ satisfy δ^{le} *if and only if*

$$\text{rl}(e, e') > 0 \wedge \{d, d'\} \in \{\{\text{north}, \text{south}\}, \{\text{east}, \text{west}\}\} \implies \text{dist}(e, e') \geq \delta^{le}(t).$$

Figure 2.5 shows an example of line end minimum distance rules.

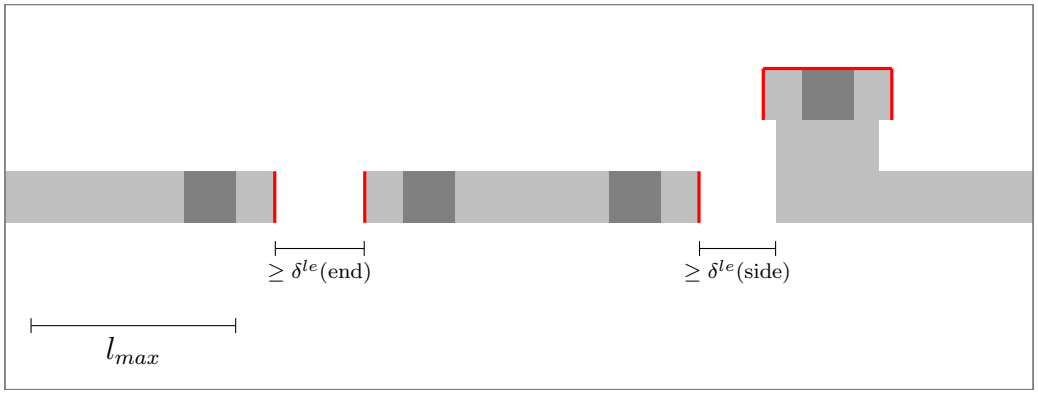


Figure 2.5: Additional minimum distance required by a line end minimum distance rule δ^{le} for end edges which are depicted in red. All other edges are side edges because they are longer than l_{max} or not incident to two convex vertices.

Now we turn to minimum distance rules on via planes. The main difference here is that the required spacing of two via cut shapes on a via plane depends on both diameters of the shapes instead only on their widths.

Definition 2.17. *Given a via cut shape* s *we define its* cut class *by* $c_{\text{cut}}(s) := (|r|_{\text{hor}}, |r|_{\text{ver}})$.

In fact for each via plane p there is only a finite set $C_p^{\text{cut}} \subset \mathbb{N}^2$ of different cut classes allowed, i.e for each via shape s on p we must have $c_{\text{cut}}(s) \in C_p^{\text{cut}}$.

Definition 2.18. *A* via minimum distance rule *on a via plane* p *is a function* $\delta_p^v : C_p^{\text{cut}} \times C_p^{\text{cut}} \times \{l_{\leq 0}, l_+\} \to \mathbb{N}$. *Two via cut shapes* s, s' *on plane* p violate δ_p^v *if and only if*

$$\text{dist}(s, s') < \begin{cases} \delta_p^v(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_{\leq 0}) & \text{if } \text{rl}(A(s), A(s')) \leq 0 \\ \delta_p^v(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_+) & \text{otherwise.} \end{cases}$$

In addition there are distance requirements even between via cut shapes on *adjacent* via planes.

Definition 2.19. *An* inter layer via minimum distance rule *on via planes* $p, p+2$ *is a function* $\delta_p^{iv} : C_p^{\text{cut}} \times C_{p+2}^{\text{cut}} \times \{l_{\leq 0}, l_+\} \to \mathbb{N}$. *Two via cut shapes* s *on plane* p *and* s' *on plane* $p+2$ violate δ_p^{iv} *if and only if*

$$\text{dist}(s, s') < \begin{cases} \delta_p^{iv}(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_{\leq 0}) & \text{if } \text{rl}(A(s), A(s')) \leq 0 \\ \delta_p^{iv}(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_+) & \text{otherwise.} \end{cases}$$

Remark. Note that for the sake of clarity we described the design rules in this section only in their most basic form. Some of them occur in practice in even more complex variants. Minimum distance requirements sometimes do not apply to the shapes directly. Certain line end minimum distance rules for example can apply to end edges being expanded in some fashion. Via minimum distance rules sometimes only apply to the *centers* of cut shapes. For a more detailed overview of the various kinds of distance rules see Silicon Integration Initiative [2007]. All the concepts we discuss in section 2.3, however, can be naturally extended to cover all design rule variants relevant in practice.

2.2.3 Same Net Rules

Besides the minimum distance requirements between the shapes of different nets there are several restrictions on the wiring of each net considered individually. Such *same net rules* generally are difficult to handle in a path search algorithm directly, because they require an analysis of the polygon built by the shapes of the path at a point of time where it is still under construction. Many of them have evolved over several previous technologies. A survey of such is given in Peyer [2007]. In the following discussions we will need only some basic same net rules such as the the minimum width rule restricting the minimum possible with of shapes that can be manufactured.

Definition 2.20. *A* minimum width rule *is a pair* $w = (w_x, w_y) \in \mathbb{N}^2$. *A set of shapes* S *satisfies* w *if and only if for every point* $p \in A(S)$ *there exists a shape* r *with* $p \in A(r) \subseteq A(S)$ *and* $|r|_{\text{hor}} \geq w_x,\ |r|_{\text{ver}} \geq w_y$.

Generally two incident short polygon edges are forbidden:

Definition 2.21. *A* minimum edge rule *is a pair* $r = (l_1, l_2) \in \mathbb{N}^2$. *Let* S *be a set of shapes building the polygon* $\text{plg}(S)$. *Two incident edges* e_1, e_2 *of* $\text{plg}(S)$ *satisfy* r *if and only if* $\text{length}(e_1) < l_1 \implies \text{length}(e_2) \geq l_2$.

For vias there also are several rules restricting their bottom, cut and top shapes and in practice we have a fixed, finite, technology dependent set of via definitions $\mathbb{V}$ to choose from. Moreover for each net we may only be allowed to use a subset of these to satisfy individual timing or robustness requirements.

Definition 2.22. *A* valid via rule *on a via plane* $p \in P_{via}$ *is a set of via definitions* $V_p \subseteq \mathbb{V}$ *for some fixed technology dependent set of via definitions* $\mathbb{V}$.

A set S *of shapes satisfies* V_p *if and only if all via shapes in* S *are defined by some point in* $\mathbb{Z}^2$ *and a via definition in* V_p.

2.2.4 DPT Design Rules

Using *double patterning technology* (DPT) currently seems to be the most practical solution for manufacturing upcoming integrated circuits with smaller feature sizes than 22 nm (Tang and Cho [2011]). The problem is that with the extreme ultra violet (EUV) lithography still having technical and economical problems, one still has to use 193 nm wavelength lithography for these much smaller structures.

The idea of double patterning is to use *two* masks and double exposure to print the desired rectilinear, polygonal layout on each plane. Given a set S of shapes building these rectilinear polygons, shapes with distance below some threshold d_{dp} are assigned to different masks such that the minimum distance between shapes on the same mask (pitch) is increased, which makes production easier. Note that there are cases where such an assignment is not possible: The *conflict graph* of S is an undirected graph G_S, with $V(G_S) := S$ and $E(G_S) := \{\{s, s'\} : s, s' \in S, 0 < \mathrm{dist}(s, s') < d_{dp}\}$. Then this assignment to masks can be seen as the problem of finding a two-coloring in G, i.e. an assignment of two colors (corresponding to the two masks) to the nodes $V(G)$ such that there are no two adjacent nodes having the same color. Of course, the constraint graph may not be bipartite, in which case there is no such coloring.

If two intersecting shapes are assigned to different masks this is called a *stitch*. Note that of course the set of possible stitches depends on the specific shapes in S. There can be a set of shapes $S' \neq S$ with $\mathrm{plg}(S') = \mathrm{plg}(S)$ such that $G_{S'}$ admits a two-coloring with stitches, while G_S does not. Figure 2.6 shows an example of colored layouts.

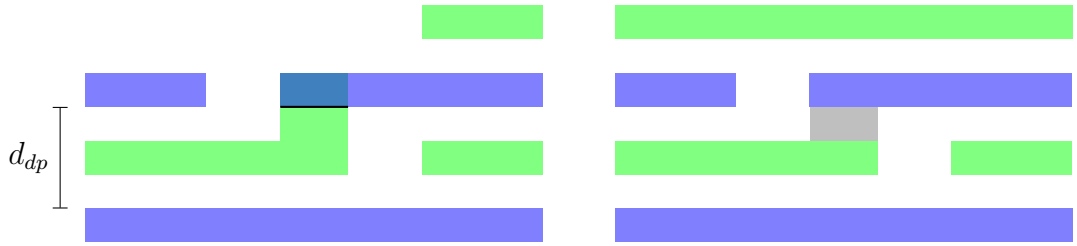


Figure 2.6: Two colored configurations of wire shapes. Shapes of equal color must have a vertical distance of at least d_{dp}. In (a) a complete 2-coloring is shown by using one stitch (black horizontal line). In (b) it is not possible to obtain a 2-coloring of all the shapes. For example the gray shape cannot be feasibly colored if the colors of the other segments are fixed as shown.

Stitches, however, can lead to *overlay errors* in the combined result, i.e. the two intersecting shapes printed with different masks are not sufficiently connected. Therefore it is desirable to compute a feasible coloring using the minimum number of stitches. If the given layout is two-colorable, and an eligible shape decomposition providing all necessary stitching positions is computed as for example in Chen and Chang [2010], this

problem can be solved optimally in polynomial time by reducing it to a min cut problem in a planar graph (Tang and Cho [2011]).

The harder problem, however, is to actually obtain a two-colorable instance at all. In current works this problem is either addressed by locally fixing coloring conflicts heuristically, or by applying linear programming techniques minimizing layout perturbation, see e.g. Ghaida et al. [2011]. Several other works discuss double patterning friendly detailed routing to make the coloring instances easier a priori, see e.g. Cho et al. [2008], Yuan et al. [2009], and Lin and Li [2010].

At the current point of time 14 nm design rules are still under development. It is unclear in which cases stitching will be allowed, or if it must not be used at all. Also the use of jogs, which often are the cause of coloring problems, may become restricted, or even forbidden completely. What is clear, however, is that minimum distance rules, besides width and run length, will first of all depend on the colors of shapes. The required minimum distance between equally colored shapes will generally be several times larger than the one for differently colored shapes. This means that automatic routing tools somehow must determine a feasible coloring, or at least ensure that one exists. In section 2.3.11 we will discuss some basic ideas how this can be done in BonnRoute.

2.3 The BonnRouteRules Module

The design rules of modern technologies, some of which we described in section 2.2, have become very complicated and moreover are specified in a complex environment. Therefore it is not an easy task to represent them in a way that they can be checked and fulfilled by an automatic routing tool efficiently.

In particular the way this was done for BonnRoute in technologies up to 45 nm does not work anymore, because both the design rules themselves and also the way they are specified changed significantly. One key point necessary in order to enable BonnRoute for server and ASIC[1] designs in 32 nm technology and beyond was to rewrite the overall design rule handling. The result of this was a new module designed and implemented by the author, called *BonnRouteRules*. This module serves as an interface for all design rules and creates an appropriate representation in which these rules can be handled in BonnRoute efficiently. The goal of this section is to describe the main aspects of this module. Several technical details, however, are omitted for the sake of clarity.

2.3.1 BonnRoute Wiring Representation

In order to being able to describe the actual task of the BonnRouteRules module, we need to introduce the wiring and distance rule representation of BonnRoute. Generally wires and vias are represented by *stick figures*, i.e. line segments annotated with a *wire type* describing their shape representation.

[1] Application Specific Integrated Circuit

Definition 2.23. *Let a* shape type *be an element of* $T_{shape} := \{\text{pref}, \text{jog}, \text{bot}, \text{cut}, \text{top}, \text{above}\}$. *A* wire type element *is a 3-tuple* (p, t, o) *where* $p \in P, t \in T_{shape}$*, and* o *is a shape on plane* p *called* overhang. *A* wire type *is a finite set of wire type elements* (p, t, o) *with the following properties:*

(i) $\nexists (p, t, o') \in W$ *with* $o' \neq o$

(ii) $p \in P_{wiring} \iff t \in \{\text{pref}, \text{jog}, \text{bot}, \text{top}\}$

(iii) $p \in P_{via} \iff t \in \{\text{cut}, \text{above}\}$

(iv) $t = \text{cut} \iff \exists (p-1, \text{bot}, o') \in W \iff \exists\, (p+1, \text{top}, o'') \in W$

(v) $t = \text{above} \implies \exists (p-2, \text{cut}, o') \in W$

The elements of T_{shape} represent the types of shapes the wire type element will induce together with a stick figure. It may induce the shape of a wire in preferred direction, a jog, or some shape of a via. Property (i) of definition 2.23 ensures that there is at most one overhang shape for each plane and shape type, property (ii) and (iii) define the possible plane and shape type combinations that make sense, and the remaining properties ensure the completeness of via shapes that belong together.

Definition 2.24. *A* stick figure *is a pair* $s = (l, W)$*, where* l *is an axis parallel line segment connecting two points* $(x_1, y_1, p_1), (x_2, y_2, p_2) \in \mathbb{Z} \times \mathbb{Z} \times P$ *and* W *a wire type. We define* $l(s) := l, W(s) := W$*,*

$$A(s) := \{(x_1, y_1) + \lambda((x_2, y_2) - (x_1, y_1)), 0 \leq \lambda \leq 1\} \subset \mathbb{R}^2,$$

and

$$x_1(s) := x_1,\ y_1(s) := y_1,\ x_2(s) := x_2,\ y_2(s) = y_2,\ p_1(s) := p_1,\ p_2(s) := p_2.$$

If l *is running in* z *direction,* s *is a* via stick figure*, otherwise it is a* wire stick figure*. If both endpoints of* l *are contained in tracks, we say that the stick figure is* on-track*, and* off-track *otherwise.*

For convenience we say that s *is running in preferred or non-preferred direction, or is horizontal or vertical, if and only if this is the case for* $l(s)$*.*

The actual shape representation of a stick figure is defined as follows:

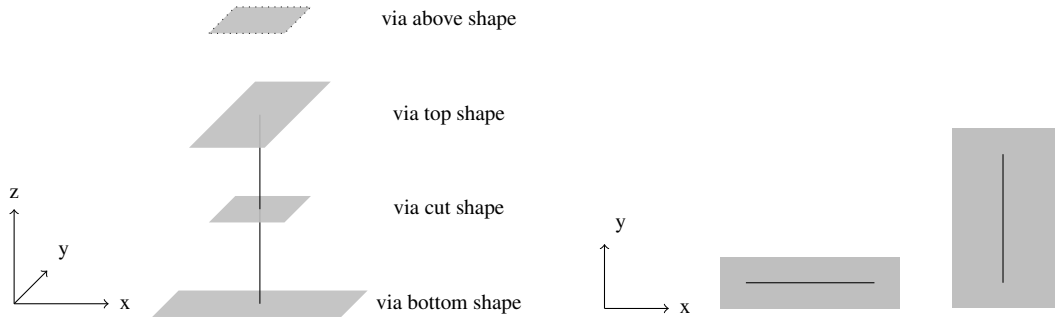
Definition 2.25. *Consider a stick figure* $s = (l, W)$ *on plane* p*. If* s *is a wire stick figure, and* W *contains the elements* $(p, \text{pref}, m_{\text{pref}})$ *and* $(p, \text{jog}, m_{\text{jog}})$*, then* s induces *the shape*

$$l + \begin{cases} o_{\text{jog}} & \text{if } l \text{ is running against the preferred direction of } p \\ o_{\text{pref}} & \text{otherwise.} \end{cases}$$

If s is a via stick figure connecting two points on wiring planes p and $p+2$, and W contains the elements $(p, \mathrm{bot}, o_{\mathrm{bot}})$, $(p+1, \mathrm{cut}, o_{\mathrm{cut}})$, and $(p+2, \mathrm{top}, o_{\mathrm{top}})$, then s induces *the shapes $l+o_{\mathrm{bot}}, l+o_{\mathrm{cut}}$, and $l+o_{\mathrm{top}}$,* called via bottom shape*,* via cut shape *and* via top shape*, respectively. If in this case W in addition contains an element $(p+3, \mathrm{above}, o_{\mathrm{above}})$, then s additionally induces a fourth shape $l+o_{\mathrm{above}}$, called* via above shape*.*

For any shape $r = l+o$ induced by s, where $w = (p,t,o) \in W$ is the wire type element used to define r, we also say that s and w induce r.

The first three via shapes mentioned in definition 2.25 correspond to the ones defined by a via definition (as in definition 2.9). Figure 2.7 shows an example of stick figures and their induced shapes.



(a) via (b) preferred direction wire and jog

Figure 2.7: Stick figures (black line segments) and induced shapes (gray).

Each path in BonnRoute is represented by a set of stick figures only intersecting at their endpoints. We construct wire types such that stick figure connectivity implies sufficient electrical connectivity of the induced shapes. This means we ensure that if we have a set of stick figures connecting all pins of a net, the corresponding shapes are an electrically robust connection of the net.

We define the width of a wire type element on a wiring plane as follows.

Definition 2.26. *The* width *of a wire type element $w = (p,t,o)$ on a wiring plane p with preferred direction $\mathrm{dir}(p) = d$ is*

$$\mathrm{width}(w) := \begin{cases} |o|_{\mathrm{hor}} & \text{if } (d = \mathrm{ver} \wedge t = \mathrm{pref}) \vee (d = \mathrm{hor} \wedge t = \mathrm{jog}) \\ |o|_{\mathrm{ver}} & \text{if } (d = \mathrm{ver} \wedge t = \mathrm{jog}) \vee (d = \mathrm{hor} \wedge t = \mathrm{pref}) \\ \min\{|o|_{\mathrm{hor}}, |o|_{\mathrm{ver}}\} & \text{otherwise .} \end{cases}$$

With very few exceptions, which in practice only occur in off-track routing, the wire stick figures created by BonnRoute will have the following property:

Definition 2.27. *A wire stick figure s has* feasible length *if and only if for the shape r induced by s and $w \in W(s)$ we have* $\mathrm{width}(r) = \mathrm{width}(w)$.

In BonnRoute determining minimum distance requirements needs to be very fast because the question whether a wire shape at a certain position satisfies these has to be answered dozens of times in each path search. We cannot afford a time consuming analysis of all the properties which design rules depend on, e.g. width and run length of sets of shapes. Therefore all minimum distance requirements in BonnRoute will only depend on the shape class of each individual shape. Since minimum distance rules as defined in section 2.2 actually apply to sets of shapes, this approach is only feasible if we have the following property:

Definition 2.28. *A set of shapes S is* width regular *if and only if for all $p \in A(S)$ we have* $\mathrm{width}(S,p) = \max_{s \in S : p \in A(s)} \mathrm{width}(s)$.

This means that the width at any point in the area of a set of shapes is uniquely determined by one of the shapes covering it. Therefore, in this case considering only the width of each shape covering the point individually when evaluating a minimum distance rule is equivalent to considering the exact width with respect to the whole set of shapes.

To maintain width regularity and satisfy same net rules, it is beneficial to avoid unnecessary shape intersections. Generally, we want stick figures only to intersect at their endpoints, and for each pair of stick figures s, s' we want their induced shapes to intersect only if $l(s) \cap l(s') \neq \emptyset$. In most cases this can be satisfied by simply avoiding unnecessary local detours. Therefore, BonnRoute typically only generates sets S of shapes on the same plane which have the following property: There is no point $q \in A(S)$ where a square Q with edge length $\mathrm{width}(q,S)$ and $q \in Q \subseteq A(S)$ intersects more than two shapes of S. Having this property, width regularity of S immediately follows if each subset of two shapes of S is width regular, which is easy to guarantee:

Proposition 2.29. *A set of shapes S with $|S| = 2$ is width regular if and only if for all $s \in S$ at least one of the following conditions holds:*

(i) $|I_l(s) \cap I_l(s')| \leq |s|_w$

(ii) $I_w(s) \subseteq I_w(s')$

(iii) $I_w(s) \supseteq I_w(s')$

where $\{s'\} := S \setminus \{s\}$, $w := \begin{cases} \mathrm{hor} & \text{if } |s|_{\mathrm{hor}} \leq |s|_{\mathrm{ver}} \\ \mathrm{ver} & \text{otherwise,} \end{cases}$ *, and* $\{l\} := \{\mathrm{hor}, \mathrm{ver}\} \setminus \{w\}$.

Proof. Let S be a set of two shapes s and s'. For sufficiency we show that each of the three properties implies that $\mathrm{width}(p,S) = \max\{\mathrm{width}(s), \mathrm{width}(s')\}$ for all $p \in A(s)$. If (i) or (ii) is satisfied, we clearly have $\mathrm{width}(p,S) = \mathrm{width}(s)$ for all $p \in A(s) \setminus A(s')$ and $\mathrm{width}(p,S) = \max\{\mathrm{width}(s), \mathrm{width}(s')\}$ for all $p \in A(s) \cap A(s')$. If (iii) holds, we have $\mathrm{width}(s) \geq \mathrm{width}(s')$ and $\mathrm{width}(p,S) = \mathrm{width}(s)$ for all $p \in A(s)$.

To show necessity assume that all three properties are violated for s. Since (i) is not satisfied, we have $|I_l(s) \cap I_l(s')| > |s|_w$, therefore by violation of (ii) and (iii) there exists

a $p \in A(s) \setminus A(s')$ and a shape t with $|t|_{\text{hor}} = |t|_{\text{ver}} > |s|_w = \text{width}(s)$ and $A(t) \subseteq A(S)$. This means that S is not width regular. □

In section 2.3.2 we will construct each wire type such that all shapes induced by two intersecting wire stick figures with this wire type always satisfy at least one of the properties of proposition 2.29. By these measures, sets of shapes which are not width regular rarely occur in practice. And even if they occur, using the width of individual shapes instead of the actual width to evaluate minimum distance rules does not necessarily imply obtaining a smaller minimum distance value. As long as both widths are contained in the same width interval of the minimum distance rule, the outcome is the same.

Let us now turn to the different kinds of minimum distance rules between shape classes we currently have in BonnRoute:

Definition 2.30. *Let* $T_{dist} := \{\text{north}, \text{east}, \text{south}, \text{west}, \text{hor}, \text{ver}, \text{eucl}\}$. *A* shape class minimum distance rule *is a quadruple* $(c_1, c_2, t, d) \in \mathbb{N} \times \mathbb{N} \times T_{dist} \times \mathbb{N}$ *describing the required minimum distance* d *between a pair of shape classes* c_1, c_2 *with respect to* distance type t.

Definition 2.31. *Let* s *be a shape with shape class* $c(s) = c_1$ *and* $r = (c_1, c_2, t, d)$ *a shape class minimum distance rule. The* violation area $A(s, r)$ *of* s *and* r *is*

$$A(s,r) := \begin{cases} \{p \in \mathbb{R}^2 : \text{dist}(\{p\}, A(s)) \leq d\} & \text{if } t = \text{eucl} \\ \{(x,y) \in \mathbb{R}^2 : x \in [x_1(s), x_2(s) + d], y \in [y_1(s), y_2(s)]\} & \text{if } t = \text{east} \\ \{(x,y) \in \mathbb{R}^2 : x \in [x_1(s) - d, x_2(s)], y \in [y_1(s), y_2(s)]\} & \text{if } t = \text{west} \\ \{(x,y) \in \mathbb{R}^2 : x \in [x_1(s), x_2(s)], y \in [y_1(s), y_2(s) + d]\} & \text{if } t = \text{north} \\ \{(x,y) \in \mathbb{R}^2 : x \in [x_1(s), x_2(s)], y \in [y_1(s) - d, y_2(s)]\} & \text{if } t = \text{south} \\ A(s, (c_1, c_2, \text{east}, d)) \cup A(s, (c_1, c_2, \text{west}, d)) & \text{if } t = \text{hor} \\ A(s, (c_1, c_2, \text{south}, d)) \cup A(s, (c_1, c_2, \text{north}, d)) & \text{if } t = \text{ver} \end{cases}$$

Given another shape s'*, we say that* s, s' satisfy r *if* $(c(s), c(s')) \neq (c_1, c_2)$ *or*

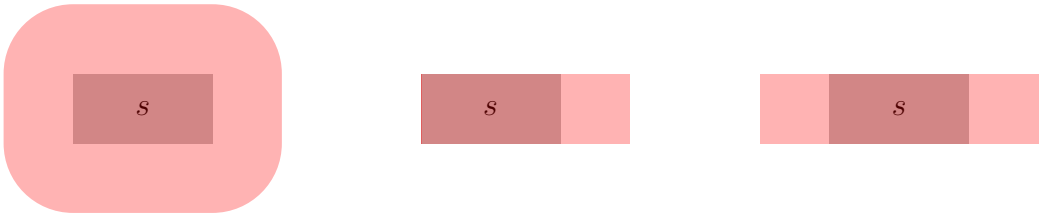
$$(A(s,r)^\circ \cap A(s') = \emptyset) \wedge (c(s') = c_1 \implies A(s',r)^\circ \cap A(s) = \emptyset)$$

Otherwise s, s' violate r.

For two shape class minimum distance rules r, r' *we say that* r dominates r' *if and only if for all shapes* s_1, s_2 *we have that* s_1, s_2 *satisfy* $r \implies s_1, s_2$ *satisfy* r'.

An example of the different types of shape class minimum distance rules is shown in figure 2.8. In section 2.4 we will describe how to identify violations of such shape class minimum distance rules efficiently.

Note that we usually have multiple different shape class minimum distance rules between the same pair of shape classes, for example to model minimum distance requirements for different run lengths. Given a minimum distance rule δ and any two shapes s, s'



$A(s, r_{eucl})$ $A(s, r_{east})$ $A(s, r_{hor})$

Figure 2.8: Area $A(s, r_t)$ (red) of a shape s and shape class minimum distance rules $r_t = (c(s), c_2, t, d)$ for $t \in \{\text{eucl}, \text{east}, \text{hor}\}$.

we will represent the minimum distance required by δ with a set $R(\delta, s, s') := \{r_1, r_2, r_3\}$ of shape class minimum distance rules, where

$$\begin{aligned} r_1 :=& (c(s), c(s'), \text{eucl}, \delta(\text{width}(s), \text{width}(s'), l_{\leq 0})), \\ r_2 :=& (c(s), c(s'), \text{hor}, \delta(\text{width}(s), \text{width}(s'), l_+)), \\ r_3 :=& (c(s), c(s'), \text{ver}, \delta(\text{width}(s), \text{width}(s'), l_+)). \end{aligned}$$

In the case of simple run length dependency this is an exact representation:

Proposition 2.32. *Given a minimum distance rule δ, we have*

$$s, s' \text{ satisfy } \delta \iff s, s' \text{ satisfy all } r \in R(\delta, s, s') \tag{2.1}$$

for any two shapes s, s' if and only if δ has simple run length dependency.

Proof. Assume δ has simple run length dependency, and let s, s' be two shapes. If $\text{rl}(A(s), A(s')) \leq 0$, then by definition of simple run length dependency s, s' violate δ if and only if they violate r_1. If we have $\text{rl}(A(s), A(s')) > 0$, they violate δ if and only if they violate r_2 or r_3. Since in the first case r_2 and r_3 are always satisfied, and in the second case definition 2.12 (ii) ensures that r_1 can only be violated if r_2 or r_3 is violated, we have that s, s' violate δ if and only if they violate an element of $R(\delta, s, s')$.

To show the converse assume that δ does not have simple run length dependency. Then there are shapes s, s' with $\text{rl}(A(s), A(s')) \leq 0$ and

$$d := \delta(\text{width}(s), \text{width}(s'), \text{rl}(A(s), A(s')) \neq \delta(\text{width}(s), \text{width}(s'), l_{\leq 0}),$$

or with $\text{rl}(A(s), A(s')) > 0$ and $d \neq \delta(\text{width}(s), \text{width}(s'), l_+)$, which implies that (2.1) is not satisfied. □

In the following sections we describe in detail how we generate wire types, appropriate shape classes, and shape class minimum distance rules in order to represent a given set of design rules.

2.3.2 Generating Wire Types

The most important task of the BonnRouteRules module is to convert for each net its given set of design rules to an appropriate wire type and a set of shape class minimum distance rules. In this section we describe in detail how the overhangs of all the wire type elements are defined. We postpone the construction of shape classes to section 2.3.3.

We begin by defining the set of design rules we consider to be assigned to each net in the in the input of the module:

Definition 2.33. *A* rule set *is a set* $R := \{R_p : p \in P\}$ *with*

$$R_p := \begin{cases} \{\delta_p, \delta_p^{le}, w_p\} & \text{if } p \in P_{wiring} \\ \{\delta_p^v, \delta_p^{iv}, V_p\} & \text{if } p \in P_{via} \wedge p+2 \leq p_{max} \\ \{\delta_p^v, V_p\} & \text{if } p \in P_{via} \wedge p+2 > p_{max} \end{cases}$$

where δ_p *is a minimum distance rule,* δ_p^{le} *a line end minimum distance rule,* w_p *a minimum width rule,* δ_p^v *and* δ_p^{iv} *via and inter layer via minimum distance rules, respectively, and* V_p *a valid via rule.*

Let N be a set of nets, $R(n) := \{R_p(n) : p \in P\}$ a rule set for each $n \in N$ containing the rules that have to be satisfied by the wiring of n. Let $(\tilde{w}_p^x, \tilde{w}_p^y)$ and $\tilde{d}_p$ be the smallest minimum width and minimum distance value over the rules of all nets on plane p, i.e. $\tilde{w}_p^x := \min\{w_p^x : w_p = (w_p^x, w_p^y) \in \bigcup_{n \in N} R_p(n)\}$, $\tilde{w}_p^y$ analogously, and $\tilde{d}_p := \min\{\delta_p(w, w', l) : \delta_p \in \bigcup_{n \in N} R_p(n), w, w' \in \mathbb{N}, l \in \mathbb{Z}_{\geq -1}\}$. Typically, the design rules of most of the nets allow actually using wires of these minimum widths having this minimum distance to each other. Therefore it is natural to use uniform tracks with pitch $\tilde{w}_p^y + \tilde{d}_p$ on a wiring plane p if p has horizontal preferred direction, and $\tilde{w}_p^x + \tilde{d}_p$ otherwise. If we consider for example two horizontal on-track stick figures s, s' on neighboring horizontal tracks, and the shape $\tilde{o}_p = (0, -\lfloor \frac{\tilde{w}_p^y}{2} \rfloor, 0, \lceil \frac{\tilde{w}_p^y}{2} \rceil)$, then the shapes $l(s) + \tilde{o}_p$ and $l(s') + \tilde{o}_p$ have exactly the necessary distance $\tilde{d}_p$ (if they have positive run length).

Remark. Nonuniform tracks, however, can enable even more efficient routing space usage by better alignment to regular structures like power connection wires. But even in that case it is still valid to assume that almost all neighboring tracks have the same distance. An efficient method to compute routing tracks for optimal routing space usage was proposed by Müller [2009].

Now let us fix some rule set $R := \{R_p : p \in P\} = R(n)$ for some net $n \in N$. We will construct a wire type W_R with respect to this set of rules. In the following let p be a wiring plane, and assume w.l.o.g. that it has horizontal preferred direction. The vertical preferred direction case is analogous.

We create overhang shapes o_{pref} and o_{jog} to add wire type elements $(p, \text{pref}, o_{\text{pref}})$ and $(p, \text{jog}, o_{\text{jog}})$ to the wire type W_R. As noted earlier we define the shape classes of these shapes later in section 2.3.3, and first concentrate on their geometry. These overhangs

define the shapes induced by stick figures with wire type W_R. Therefore, we want to ensure using as little routing space as possible while satisfying the minimum width rule $w_p = (w_p^x, w_p^y) \in R_p$. A good measure of routing space usage is the number of *blocked tracks*.

Definition 2.34. *Assume that we have uniform tracks with track pitch $t_p \in \mathbb{N}_{>0}$ on a plane $p \in P_{wiring}$ with horizontal preferred direction. For a shape o on plane p and minimum distances $d_1, d_2 \in \mathbb{N}$ to the south and north, respectively, we define the* number of blocked tracks *as*

$$\beta_p(o, d_1, d_2) := \begin{cases} u - l + 1 & \text{if } lt_p \le y_2(o) \\ 0 & \text{otherwise} \end{cases}$$

where

$$l := \left\lceil \frac{y_1(o) - d_1 + 1 - |y_2(\tilde{o}_p)|}{t_p} \right\rceil,$$
$$u := \left\lfloor \frac{y_2(o) + d_2 - 1 + |y_1(\tilde{o}_p)|}{t_p} \right\rfloor.$$

Analogously the number of blocked tracks can be defined for wiring planes with vertical preferred direction and vertical tracks.

Given a horizontal on-track stick figure s, this counts the number of tracks where we cannot place another stick figure $\tilde{s}$ (with positive run length) without violating minimum distance d_1 or d_2 between $l(s)+o$ and $l(\tilde{s})+\tilde{o}_p$. Because the overhang $\tilde{o}_p$ typically induces most of the wire shapes on p, this is a valid way to measure the use of routing space. Note that definition 2.34 distinguishes between the southern and northern required minimum distance. This will become useful for determining the blocked tracks of vias, where these requirements actually can differ. For wires, however, we only use the same distance value for both directions.

With $d_p := \delta_p(w_p^y, \tilde{w}_p^y, l_+)$ we now define the overhang o_{pref} by setting

$$\begin{aligned} y_1(o_{\text{pref}}) &:= -\min\left\{w_p^y,\ \min\{y \in \mathbb{N} : y - d_p - |y_2(\tilde{o}_p)| \equiv 0 \mod t_p\}\right\}, \\ y_2(o_{\text{pref}}) &:= w_p^y - |y_1(o_{\text{pref}})|, \end{aligned} \tag{2.2}$$

This actually minimizes the number of blocked tracks in the following sense:

Proposition 2.35. *The shape o_{pref} as defined above minimizes the number of blocked tracks $\beta_p(o, d_p, d_p)$ over all shapes o with $|o|_{\text{ver}} = w_y^p$.*

Proof. By definition of β_p and o_{pref} we clearly have

$$\begin{aligned}
&\beta_p(o_{\text{pref}}, d_p, d_p) \\
&\leq \left\lfloor \frac{y_2(o_{\text{pref}}) + d_p - 1 + |y_1(\tilde{o}_p)|}{t_p} \right\rfloor - \left(\frac{y_1(o_{\text{pref}}) - d_p - |y_2(\tilde{o}_p)|}{t_p} + 1 \right) + 1 \\
&= \left\lfloor \frac{w_p^y + 2d_p - 1 + \tilde{w}_p^y}{t_p} \right\rfloor .
\end{aligned}$$

Furthermore for any shape o we have that $\beta_p(o, d_p, d_p)$ equals the number of integers $t \equiv 0 \mod t_p$ in the interval

$$[\, y_1(o) - d_p + 1 - |y_2(\tilde{o}_p)| \,,\;\; y_2(o) + d_p - 1 + |y_1(\tilde{o}_p)| \,] \,.$$

This interval contains $w_p^y + 2d_p - 1 + \tilde{w}_p^y$ integers if $|o|_{\text{ver}} = w_y^p$. Since for at least $\lfloor \frac{w_p^y + 2d_p - 1 + \tilde{w}_p^y}{t_p} \rfloor$ of these integers t we must have $t \equiv 0 \mod t_p$, the proposition follows. □

For the choice of the overhang shape o_{jog} we do not need to consider the number of blocked tracks because jogs are running orthogonal to tracks anyway. The number of blocked tracks in this case depends on the length of the jog instead of its width. Jogs generally should be used rarely and kept as short as possible to avoid blocking several tracks. We therefore simply set

$$x_1(o_{\text{jog}}) := -\left\lfloor \frac{w_p^x}{2} \right\rfloor ,\; x_2(o_{\text{jog}}) := \left\lceil \frac{w_p^x}{2} \right\rceil \tag{2.3}$$

to satisfy the minimum width rule.

For the remaining parts of o_{pref} and o_{jog}, i.e. their extension in and against preferred direction, respectively, we are not restricted by design rules. We want to ensure, however, width regularity of each pair of a jog shape and preferred direction wire shape in order to avoid minimum distance violations as in proposition 2.29. Therefore, we set

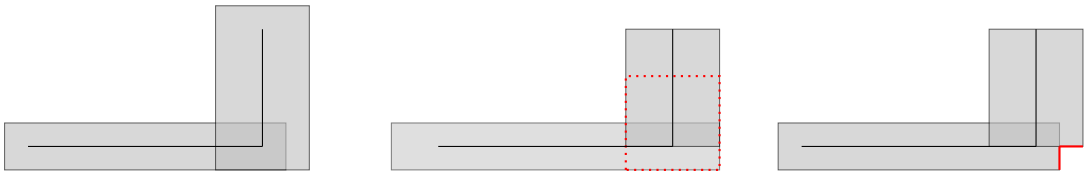
$$\begin{aligned}
x_1(o_{\text{pref}}) &:= y_1(o_{\text{pref}}), \\
x_2(o_{\text{pref}}) &:= y_2(o_{\text{pref}}), \\
y_1(o_{\text{jog}}) &:= y_1(o_{\text{pref}}), \\
y_2(o_{\text{jog}}) &:= y_2(o_{\text{pref}}),
\end{aligned} \tag{2.4}$$

if $w_p^x \geq w_p^y$, and

$$\begin{aligned}
x_1(o_{\text{pref}}) &:= x_1(o_{\text{jog}}), \\
x_2(o_{\text{pref}}) &:= x_2(o_{\text{jog}}), \\
y_1(o_{\text{jog}}) &:= x_1(o_{\text{jog}}), \\
y_2(o_{\text{jog}}) &:= x_2(o_{\text{jog}}),
\end{aligned} \tag{2.5}$$

otherwise.

With this we ensure width regularity of intersecting jog and wire shapes (as we will show in lemma 2.36), and avoid concave vertices of the polygon build by such two shapes. Such vertices generally are bad from a lithographic point of view, and may even cause minimum edge rule violations. Figure 2.9 shows an example.



(a) width regular (b) not width regular (c) concave vertex

Figure 2.9: Using overhang shapes as defined in (2.4) the induced shapes of two intersecting stick figures running in orthogonal directions look like in (a). Width regularity is preserved, and there are no concave vertices. This is not the case for the shapes shown in (b) and (c), which result from differently defined overhang shapes.

Overall we have the following lemma implied by our definition of overhang shapes:

Lemma 2.36. *Let p be a wiring plane, $w = (w_p^x, w_p^y)$ the minimum width rule in $R_p \subseteq R$, and S a set of shapes on p induced by a set of wire stick figures with wire type W_R. Then the following properties hold:*

(i) *S satisfies w if we have $|s|_{\text{hor}} \geq w_p^x$ and $|s'|_{\text{ver}} \geq w_p^y$ for all shapes $s, s' \in S$ induced by stick figures running in horizontal and vertical direction, respectively.*

(ii) *For all $s_{\text{pref}}, s_{\text{jog}} \in S$ induced by stick figures intersecting at one of their endpoints and running in orthogonal directions we have that $\{s_{\text{pref}}, s_{\text{jog}}\}$ is width regular.*

(iii) *If $w_p^x \geq w_p^y$, all stick figures with wire type W_R running in x direction have feasible length.*

If $w_p^x < w_p^y$, all stick figures with wire type W_R running in y direction have feasible length.

Proof. The sufficient length of the shapes in S, and the definition of overhangs in (2.2) and (2.3) immediately implies (i).

To show (ii) assume w.l.o.g. that $\text{width}(s_{\text{pref}}) \leq \text{width}(s_{\text{jog}})$. Then by the definition of overhang shapes in (2.4) and (2.5) we have that s_{pref} satisfies property (ii) of proposition 2.29, and s_{jog} satisfies property (i) of proposition 2.29. This implies that the set of these two shapes is width regular.

Property (iii) immediately follows from our definition of overhang shapes in (2.4) and (2.5), because these imply that already the induced shape of any wire stick figure of length one running in the respective direction specified in (iii) has feasible length. □

We now turn to the case where p is a via plane, and we will add wire type elements to W_R defining the induced shapes of via stick figures. Assume w.l.o.g. that the wiring plane $p-1$ has horizontal and $p+1$ vertical preferred direction. We cannot define the overhang shapes of these wire type elements arbitrarily. Instead we have to choose a via definition from the set specified by the valid via rule $V_p \in R_p$. Each via definition consists of three shapes that we can use as overhang shapes for defining via bottom, via cut, and via top shapes. Of course we again want to use as little routing space as possible with each via.

Assuming uniform track pitches on $p-1$ and $p+1$, and considering an on-track via stick figure s connecting two points on these planes, we compute for each via definition $v_p = (o_{p-1}, o_p, o_{p+1}) \in V_p$ the amount of routing space needed by $l(s) + o_{p-1}$, $l(s) + o_p$, and $l(s) + o_{p+1}$. For this we need to consider several minimum distance requirements:

- First we consider the minimum distance $d(o_{p-1})$ required by the minimum distance rule δ_{p-1} on the lower wiring plane, i.e. $d(o_{p-1}) := \delta_{p-1}(\mathrm{width}(o_{p-1}), \tilde{w}^y_{p-1}, l_+)$. Analogously we set $d(o_{p+1})$.

- In some cases we even consider line end minimum distance rules. Let δ^{le}_{p-1} be the line end minimum distance rule in R_{p-1}, and l_{max} the maximum length of an end edge as in definition 2.15. Assume we already have a wire type element $(p-1, \mathrm{pref}, o^{\mathrm{pref}}_{p-1}) \in W_R$. If $y_1(o_{p-1}) < y_1(o^{\mathrm{pref}}_{p-1})$ and $|o_{p-1}|_{\mathrm{hor}} \leq l_{max}$, it holds that for any wire stick figure s_w intersecting s the edge $\mathrm{edge}(l(s) + o_{p-1}, \mathrm{south})$ is an end edge. In this case we set $d_{\mathrm{south}}(o_{p-1}) := \max\{d(o_{p-1}), \delta^{le}_{p-1}(\mathrm{side})\}$, otherwise we set $d_{\mathrm{south}}(o_{p-1}) := d(o_{p-1})$. Analogously one can determine the required distances $d_{\mathrm{north}}(o_{p-1}), d_{\mathrm{east}}(o_{p+1})$ and $d_{\mathrm{west}}(o_{p+1})$.

- The minimum required distance between a via cut shape $l(s) + o_p$ and any other via cut shape on p is at least

$$d(o_p) := \min\left\{\delta^v_p((|o_p|_{\mathrm{hor}}, |o_p|_{\mathrm{ver}}), c_{\mathrm{cut}}, l) : c_{\mathrm{cut}} \in C^{\mathrm{cut}}_p, l \in \{l_{\leq 0}, l_+\}\right\},$$

  where C^{cut}_p denotes the set of allowed cut classes on p.

With this we can compute the number of blocked tracks of o_{p-1} and o_{p+1} as in the wiring plane case above. We choose a via definition $v^*_p = (o^*_{p-1}, o^*_p, o^*_{p+1}) \in V_p$ lexicographically minimizing

$$\begin{aligned}
&\big(\beta_{p-1}(o^*_{p-1}, d_{\mathrm{south}}(o^*_{p-1}), d_{\mathrm{north}}(o_{p-1})) \;+\; \beta_{p+1}(o^*_{p+1}, d_{\mathrm{east}}(o^*_{p+1}), d_{\mathrm{west}}(o^*_{p+1})),\\
&d(o^*_p),\\
&|o^*_{p-1}|_{\mathrm{hor}} + |o^*_{p+1}|_{\mathrm{ver}},\\
&\left|y_1(o^*_{p-1})\right| + d_{\mathrm{south}}(o^*_{p-1}),\\
&\left|x_1(o^*_{p+1})\right| + d_{\mathrm{east}}(o^*_{p+1})\big).
\end{aligned}$$

This clearly reflects the routing space usage of the induced shapes of on-track via stick figures: The most important aspect is the number of blocked tracks on the affected wiring planes, followed by the minimum distance required on the via plane, and the lengths of the via bottom and top shape. The last two values are just to prefer via definitions that, in case all other values are equal, need less space in south and east direction on $p-1$ and $p+1$, respectively. It can be beneficial having all via shapes spare routing space in the same direction.

We then add the wire type elements

$$\begin{aligned} &(p-1, \mathrm{bot}, o^*_{p-1}),\\ &(p, \mathrm{cut}, o^*_p),\\ &(p+1, \mathrm{top}, o^*_{p+1}),\\ &(p+2, \mathrm{above}, o^*_p) \end{aligned} \tag{2.6}$$

to W_R. Note that in order to be able to handle inter layer via rules in section 2.3.3 correctly, we reuse the overhang o^*_p for the wire type element with shape type above. This means that each via cut shape on p is projected to $p+2$. With this approach we can keep our checking algorithms in BonnRoute simpler, and only check violations between shapes on the same plane.

This concludes our discussion on how we determine the overhangs of wire type elements. In the next sections we describe the construction of appropriate shape classes and shape class minimum distance rules.

2.3.3 Generating Shape Classes

Assume we have created a wire type $W_{R(n)}$ with respect to rule set $R(n)$ for each net $n \in N$ as described in section 2.3.2. So far we only defined the overhang shapes of all wire type elements properly, and still have to represent their spacing requirements by defining appropriate shape classes and shape class minimum distance rules.

As all distance checking in BonnRoute is done plane-wise, shape classes will also be assigned to wire type elements independently on each plane. First we determine for each wire type element a set of properties that define if we consider two such elements equivalent or not. Shape classes then correspond to equivalence classes of wire type elements with respect to these properties and are assigned to the corresponding overhang shapes.

For each wire type element these properties basically are its minimum distance requirements to other wire type elements. As the design rules describing these requirements depend on properties of *both* of the elements, we would have to inspect for each wire type element each other one. Since this would result in quadratic runtime in the number of wire type elements, which can become quite large, we want to avoid this.

Therefore we proceed as follows. For each wire type element w we inspect the design rules that apply to it and extract the information describing what minimum distance is

required to the different kinds of other wire type elements. Furthermore we collect information about w that, given some other wire type element w', we can determine the exact minimum distance required between w and w'. After building equivalence classes of wire type elements with respect to these properties, we then can explicitly construct shape class minimum distance rules between each pair of these classes. This leads to quadratic runtime only in the number of shape classes, which generally is much smaller than the number of wire type elements. The set of properties of wire type elements we use to determine equivalence is constructed in the rest of this section.

Given two shapes on a wiring plane, by the definition of minimum distance rules it is sufficient (despite of run length) to know the width intervals the widths of the shapes are contained in, to determine their required minimum distance. To get this information with respect to all minimum distance rules we have to consider sufficiently fine intervals. Let p be a wiring plane, Δ_p be the set of all minimum distance rules in $\bigcup_{n\in N} R_p(n)$, and $\mathcal{I}_p^j, j = 1, \ldots, |\Delta_p|$ their sets of width intervals as defined in definition 2.12. Let $\mathcal{I}_p$ be a partition of $\bigcup_{j=1,\ldots,|\Delta_p|} \bigcup_{I\in\mathcal{I}_p^j} I$ such that for each $I \in \mathcal{I}_p$ we have at most one I_j in each $\mathcal{I}_p^j$ with $I \cap I_j \neq \emptyset$. To evaluate any minimum distance rule in Δ_p for two given widths, it then is sufficient to know the intervals in $\mathcal{I}_p$ these widths are contained in:

Proposition 2.37. *Given two shapes s, s' on plane p with run length $\mathrm{rl}(A(s), A(s')) = l$ and $\mathrm{width}(s) \in I_s \in \mathcal{I}_p$, and $\mathrm{width}(s') \in I_{s'} \in \mathcal{I}_p$ we have*

$$\delta(\mathrm{width}(s), \mathrm{width}(s'), l) = \delta(w, w', l) \quad \forall\, w \in I_s, w' \in I_{s'}, \delta \in \Delta_p$$

Proof. Assume there is a minimum distance rule $\delta \in \Delta_p$ with $\delta(\mathrm{width}(s), \mathrm{width}(s'), l) \neq \delta(w, w', l)$ for some $w \in I_s, w' \in I_{s'}$. Then by definition of minimum distance rules the set of width intervals $\mathcal{I}_\delta$ of δ must contain disjoint intervals I_1, I_2 with $\mathrm{width}(s) \in I_1, w \in I_2$, or $\mathrm{width}(s') \in I_1, w' \in I_2$. In the first case we have $\mathrm{width}(s) \in I_s \cap I_1$ and $w \in I_s \cap I_2$, so I_s intersects two intervals in $\mathcal{I}_\delta$, which contradicts the construction of $\mathcal{I}_p$. The second case is analogous. □

Let $w = (p, t_w, o_w)$ be an element of wire type W_R, where $p \in P$, and $R := \{R_p : p \in P\} = R(n)$ for some net $n \in N$.

First we consider the case where p is a wiring plane. We assume that for every shape s induced by any stick figure and w, we have $\mathrm{width}(s) = \mathrm{width}(w)$, and use this value to evaluate minimum distance rules. As almost all stick figures created by BonnRoute have feasible length as defined in definition 2.27, this assumption generally is satisfied. Note that already lemma 2.36 ensures feasible length of all horizontal or all vertical stick figures on p. Stick figures not having feasible length generally only occur in rare cases involving pin access. In these few cases our approach of using $\mathrm{width}(w)$ to evaluate minimum distance rules could theoretically lead to larger minimum distance requirements. In practice, however, this is negligible.

By proposition 2.37 we do not need to consider $\mathrm{width}(w)$ directly as a property for building shape classes, the interval in $\mathcal{I}_p$ containing it is sufficient.

The second property we need is information about the edges of shapes induced by w. To handle line end minimum distance rules in section 2.3.4 we estimate which edges of shapes s induced by w and any stick figure are end edges. We represent this information by a function $\lambda_w : \{\mathrm{north}, \mathrm{east}, \mathrm{south}, \mathrm{west}\} \to \{\mathrm{end}, \mathrm{side}\}$ where $\lambda_w(a) = b$ indicates that we regard the edge $\mathrm{edge}(s, a)$ of every shape s induced by w and any stick figure as a b edge. We define the set $T_{edge} := \{\mathrm{north}, \mathrm{east}, \mathrm{south}, \mathrm{west}\} \times \{\mathrm{side}, \mathrm{end}\}$, and use elements $(a, \lambda_w(a))$ as a further property for defining shape classes.

The remaining properties basically represent the minimum distance requirements of shapes induced by w and any stick figure. We obtain these requirements by evaluating design rules in R_p, and represent them with a set $D(w) \subseteq \mathcal{I}_p \times (T_{edge} \dot\cup \{*\}) \times T_{dist} \times \mathbb{N}$. Each element $(I, t_{edge}, t_{dist}, d) \in D(w)$ will result in a set of shape class minimum distance rules of type t, which require a minimum distance of d between the shape class $c(o_w)$ and other shape classes on p determined by I and t_{edge}. The interval I basically restricts this set of shape classes to the ones having this width interval. The element t_{edge} limits this set to the shape classes which represent shapes with certain types of edges if we have $t_{edge} \in T_{edge}$, and imposes no restriction at all if $t_{edge} = *$. In section 2.3.5 we will describe in detail how the corresponding shape class minimum distance rules are derived.

To construct the set $D(w)$ we consider the minimum distance rule $\delta_p \in R_p$. We evaluate $\delta_p(\mathrm{width}(w), I, l)$ for each width interval $I \in \mathcal{I}_p$ and run length $l \in \{l_{\leq 0}, l_+\}$, and decide which type of shape class minimum distance rules we have to use based on l as in proposition 2.32. We set

$$\begin{aligned} D(w) := &\{([a,b],\, *,\, \mathrm{eucl},\, \delta_p(\mathrm{width}(w), a, l_{\leq 0})) : [a,b] \in \mathcal{I}_p\} \\ &\cup \{([a,b],\, *,\, \mathrm{hor},\, \delta_p(\mathrm{width}(w), a, l_+)) : [a,b] \in \mathcal{I}_p\} \\ &\cup \{([a,b],\, *,\, \mathrm{ver},\, \delta_p(\mathrm{width}(w), a, l_+)) : [a,b] \in \mathcal{I}_p\}\,. \end{aligned} \tag{2.7}$$

Now let us turn to the case where p is a via plane. As the minimum distances required by via minimum distance rules depend on the cut classes of via shapes instead of their widths, our first property will be the cut class $c_{\mathrm{cut}}(o_w) = (|o_w|_{\mathrm{hor}}, |o_w|_{\mathrm{ver}})$ of o_w. Let δ_p^v be the via minimum distance rule in R_p, and δ_{p-2}^{iv} the interlayer via minimum distance rule in R_{p-2}. Let C_p^{cut} the set of all cut classes on p. Similarly to the wiring plane case we represent the minimum distance requirements of w with a set $D(w) \subseteq C_p^{\mathrm{cut}} \times T_{shape} \times T_{dist} \times \mathbb{N}$ by evaluating δ_p^v and δ_{p-2}^{iv} for all cut classes c_{cut} in C_p^{cut} and C_{p-2}^{cut}, respectively. We set

$$\begin{aligned} D(w) := &\left\{(c_{\mathrm{cut}},\, \mathrm{cut},\, \mathrm{eucl},\, \delta_p^v(c_{\mathrm{cut}}(o_w), c_{\mathrm{cut}}, l_{\leq 0})) : c_{\mathrm{cut}} \in C_p^{\mathrm{cut}}\right\} \\ &\cup \left\{(c_{\mathrm{cut}},\, \mathrm{cut},\, \mathrm{hor},\, \delta_p^v(c_{\mathrm{cut}}(o_w), c_{\mathrm{cut}}, l_+)) : c_{\mathrm{cut}} \in C_p^{\mathrm{cut}}\right\} \\ &\cup \left\{(c_{\mathrm{cut}},\, \mathrm{cut},\, \mathrm{ver},\, \delta_p^v(c_{\mathrm{cut}}(o_w), c_{\mathrm{cut}}, l_+)) : c_{\mathrm{cut}} \in C_p^{\mathrm{cut}}\right\} \\ &\cup D_{iv}(w) \end{aligned} \tag{2.8}$$

where

$$\begin{aligned} D_{iv}(w) := &\left\{(c_{\text{cut}},\ \text{above},\ \text{eucl},\ \delta^{iv}_{p-2}(c_{\text{cut}}, c_{\text{cut}}(o_w), l_{\leq 0})) : c_{\text{cut}} \in C^{\text{cut}}_{p-2}\right\} \\ &\cup \left\{(c_{\text{cut}},\ \text{above},\ \text{hor},\ \delta^{iv}_{p-2}(c_{\text{cut}}, c_{\text{cut}}(o_w), l_{+})) : c_{\text{cut}} \in C^{\text{cut}}_{p-2}\right\} \\ &\cup \left\{(c_{\text{cut}},\ \text{above},\ \text{ver},\ \delta^{iv}_{p-2}(c_{\text{cut}}, c_{\text{cut}}(o_w), l_{+})) : c_{\text{cut}} \in C^{\text{cut}}_{p-2}\right\} \end{aligned} \tag{2.9}$$

if $t_w = \text{cut}$, and

$$\begin{aligned} D_{iv}(w) := &\left\{(c_{\text{cut}},\ \text{cut},\ \text{eucl},\ \delta^{iv}_{p-2}(c_{\text{cut}}(o_w), c_{\text{cut}}, l_{\leq 0})) : c_{\text{cut}} \in C^{\text{cut}}_{p}\right\} \\ &\cup \left\{(c_{\text{cut}},\ \text{cut},\ \text{hor},\ \delta^{iv}_{p-2}(c_{\text{cut}}(o_w), c_{\text{cut}}, l_{+})) : c_{\text{cut}} \in C^{\text{cut}}_{p}\right\} \\ &\cup \left\{(c_{\text{cut}},\ \text{cut},\ \text{ver},\ \delta^{iv}_{p-2}(c_{\text{cut}}(o_w), c_{\text{cut}}, l_{+})) : c_{\text{cut}} \in C^{\text{cut}}_{p}\right\} \end{aligned} \tag{2.10}$$

if $t_w = \text{above}$. Note that because δ^{iv}_{p-2} specifies minimum distances between shapes on *adjacent* planes, but not between shapes on the *same* plane, we have to ensure that corresponding shape class minimum distance rules are created exclusively between cut and above shapes, which is reflected by our definition of $D_{iv}(w)$.

This concludes our description of the necessary properties of wire type elements on wiring and via planes, except the representation of line end minimum distance rules, which we postpone to section 2.3.4.

Now we can finally define an equivalence relation $\sim$ on the set of wire type elements on each plane p:

Definition 2.38. *Let $w = (p, t, o)$ and $w' = (p, t', o')$ be two wire type elements. We call w and w'* equivalent *and write $w \sim w'$ if and only if*

(i) $p \in P_{wiring}$

(ii) $\lambda_w = \lambda_{w'}$

(iii) $\text{width}(w) \in I \in \mathcal{I}_p \iff \text{width}(w') \in I$

(iv) $D(w) = D(w')$

or

(i) $p \in P_{via}$

(ii) $t = t'$

(iii) $(|o|_{\text{hor}}, |o|_{\text{ver}}) = (|o'|_{\text{hor}}, |o'|_{\text{ver}})$

(iv) $D(w) = D(w')$.

We proceed as follows to assign a shape class to the overhang shape of each wire type element. Let $\mathcal{W}$ be the set of wire types that we created. First, for each wire type $W \in \mathcal{W}$ and wire type element $(p, t, o) \in W$ we set the shape class $c(o) \in \mathbb{N}$ such that it is a unique number among all shape classes of wire type elements on p of wire types in $\mathcal{W}$. Let E_p be the set of all wire type elements on plane p of wire types in $\mathcal{W}$. We build the set $\mathcal{E}_p = \{E_p^1, \dots, E_p^{|\mathcal{E}_p|}\} \subset 2^{E_p}$ of equivalence classes of E_p by $\sim$. The set of shape classes on plane p then is $C_p := \{1, \dots, |\mathcal{E}_p|\}$, and for each $i \in C_p$ and $(p, t, o) \in E_p^i \in \mathcal{E}_p$ we set $c(o) := i$. This way each overhang shape now has a shape class identifying its minimum distance requirements that will be translated into shape class minimum distance rules in section 2.3.5.

2.3.4 Handling Line End Minimum Distance Rules

Line end minimum distance rules in practice belong to the hardest design rules too obey without being too restrictive. The problem is that in path search algorithms it is hard to determine if the rule actually applies when the complete path has not been constructed yet. An end edge of the polygon of a path found so far may become a side edge depending on how the path is continued.

In order to keep this complexity out of our path search algorithms and preserve their efficiency, we estimate the edge types of shapes induced by wire type elements a priori. Of course this approach, which takes only individual shapes or most common combinations of such into account, cannot be exact in all situations. End edges of an isolated shape s may not be end edges anymore in the polygon with respect to a whole set of shapes containing s. As by definition of line end minimum distance rules we have $\delta_p^{le}(\text{side}) \leq \delta_p^{le}(\text{end})$, this cannot lead to violations of δ_p^{le} but may introduce too restrictive minimum distance requirements. Therefore we will be optimistic and in most cases do not regard edges running in preferred direction as end edges. The increased minimum distance requirement against preferred direction would waste a significant amount of routing space by blocking neighboring tracks.

Given some wire type element $w = (p, t, o)$ of wire type $W_{R(n)}$ for some net n and wiring plane p, we define the function $\lambda_w : \{\text{north}, \text{east}, \text{south}, \text{west}\} \to \{\text{end}, \text{side}\}$ indicating our estimated edge types as follows. W.l.o.g. assume that p has horizontal preferred direction, and recall that the maximum length of an end edge is denoted by l_{max} as in definition 2.15. If $t \in \{\text{pref}, \text{bot}, \text{top}\}$ and $|o|_{\text{ver}} \leq l_{max}$, we generally assume that the two vertical edges of shapes induced by w are end edges:

$$\lambda_w(\text{east}) := \lambda_w(\text{west}) := \begin{cases} \text{end} & \text{if } t \in \{\text{pref}, \text{bot}, \text{top}\} \wedge |o|_{\text{ver}} \leq l_{max} \\ \text{side} & \text{otherwise} \end{cases}$$

For horizontal end edges the rule δ_p^{le} requires vertical distance, which is against preferred direction in our case. In order to avoid blocking neighboring routing tracks, we are optimistic and assume that horizontal edges of shapes induced by w never are end edges

if $t \in \{\text{pref}, \text{jog}\}$. On the other hand, if $t \in \{\text{bot}, \text{top}\}$ and $|o|_{\text{hor}} \leq l_{max}$, the shapes induced by w belong to vias, and their horizontal edges may be end edges most of the time depending on their width compared to other wire shapes of the same wire type. We therefore inspect the width and length of o compared to the overhang shape o_{pref} of the wire type element $(p, \text{pref}, o_{\text{pref}}) \in W_{R(n)}$ like in the computation of blocked tracks in section 2.3.2. We set:

$$\lambda_w(\text{north}) := \begin{cases} \text{end} & \text{if } t \in \{\text{bot}, \text{top}\} \ \wedge \ |o|_{\text{hor}} \leq l_{max} \ \wedge \ y_2(o) > y_2(o_{\text{pref}}) \\ \text{side} & \text{otherwise} \end{cases}$$

$$\lambda_w(\text{south}) := \begin{cases} \text{end} & \text{if } t \in \{\text{bot}, \text{top}\} \ \wedge \ |o|_{\text{hor}} \leq l_{max} \ \wedge \ y_1(o) < y_1(o_{\text{pref}}) \\ \text{side} & \text{otherwise} \end{cases}$$

To ensure creating appropriate shape class minimum distance rules, we add additional elements to the set of distance requirements $D(w)$. Let therefore δ_p^{le} be the line end minimum distance rule in $R_p \in R(n)$, and add the following elements to $D(w)$ if $\lambda_w(east) = \lambda_w(west) = \text{end}$:

$$\begin{aligned} &([0, \infty], (\text{east}, \text{end}), \text{west}, \delta_p^{le}(\text{end})), \\ &([0, \infty], (\text{west}, \text{end}), \text{east}, \delta_p^{le}(\text{end})), \\ &([0, \infty], (\text{east}, \text{side}), \text{west}, \delta_p^{le}(\text{side})), \\ &([0, \infty], (\text{west}, \text{side}), \text{east}, \delta_p^{le}(\text{side})). \end{aligned} \tag{2.11}$$

Additionally we add

$$\begin{aligned} &([0, \infty], (\text{south}, \text{end}), \text{north}, \delta_p^{le}(\text{end})), \\ &([0, \infty], (\text{south}, \text{side}), \text{north}, \delta_p^{le}(\text{side})) \end{aligned} \tag{2.12}$$

if $\lambda_w(\text{north}) = \text{end}$, and

$$\begin{aligned} &([0, \infty], (\text{north}, \text{end}), \text{south}, \delta_p^{le}(\text{end})), \\ &([0, \infty], (\text{north}, \text{side}), \text{south}, \delta_p^{le}(\text{side})) \end{aligned} \tag{2.13}$$

if $\lambda_w(\text{south}) = \text{end}$.

This concludes our representation of the minimum distance requirements of δ_p^{le}. An illustration is shown in figure 2.10.

In section 2.3.5 we will show that if we estimated the edge types of shapes induced by w correctly by λ_w, then with shape class minimum distance rules resulting from $D(w)$ this representation is exact. As already seen in figure 2.10, however, in practice there are in fact situations where we have an end edge not indicated by the functions λ_w, which can lead to violations of line end minimum distance rules. Fortunately, the different types of such situations is limited and their number manageable. Typically in a stick figure path the following types of such situations may occur (see figure 2.11):

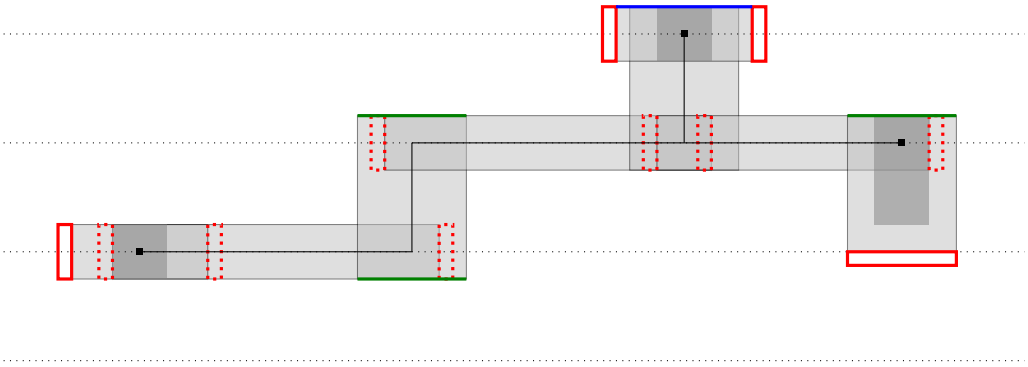


Figure 2.10: Additional minimum distance required by end edges according to the functions λ_w in red. The dotted red regions indicate the cases where we added the minimum distance although we in fact do no have an end edge there. All of these, however, are contained in other shapes anyway such that no routing space is wasted. The blue edge indicates a situation where we are optimistic with λ_w assuming no end edge although there is one. For the green edges, however, this optimism is correct because these are in fact side edges. If we assumed end edges here, the neighboring tracks would be blocked unnecessarily. Finally the large via in the bottom right is a case where we regard an edge running in preferred direction as an end edge because this via is large enough to stick out of any preferred direction wire shape of the same wire type.

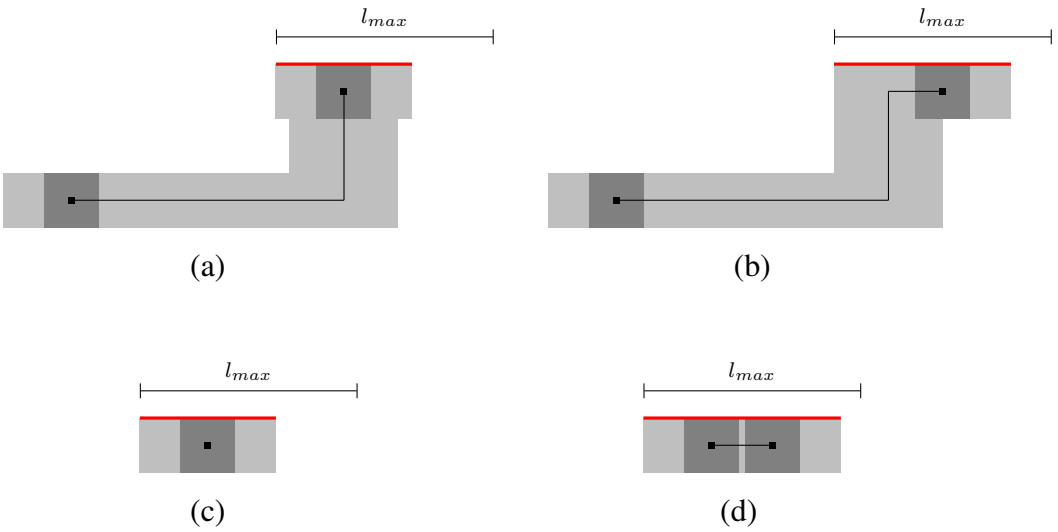


Figure 2.11: Different situations where an edge $\mathrm{edge}(s, \mathrm{north})$ (in red) of a shape s induced by a wire type element w is an end edge but $\lambda_w(\mathrm{north}) = \mathrm{side}$.

(a) A jog followed by a via is the most common situation where violations of line end minimum distance rules can occur because of our optimism.

(b) Even a jog followed by a short wire in preferred direction followed by a via may create an unexpected end edge. Such short wire segments sometimes occur in the pin access part of paths but usually not in the on-track long distance connections.

(c) Two vias on adjacent via layers at the same x and y-coordinate, so called *stacked vias*, also can cause end edges running in preferred direction that we did not reflect in the definition of λ_w.

(d) Again even a short wire in preferred direction may not help to avoid such an unexpected end edge.

Note that most of these situations involve jogs which should be used rarely anyway in order to use routing space efficiently. Moreover shape configurations as shown in figure 2.11 (a) in practice violate several same net rules even if we have no violation of a line end minimum distance rule. Similarly, cases (c) and (d) generally also violate same net rules because they create a polygon with too small area.

Ideally the router should not create such sets of shapes anyway, and there are a couple of approaches to achieve this. An easy measure to eliminate problems as shown in (c) and (d) is to choose via definitions of sufficient length in the wire type definition, but this comes at the price of additional routing space usage, even in cases where it would not be necessary. Generally, we only do this in the case where such a via definition is only slightly less efficient than an optimal one in terms of routing space consumption. In order to avoid also situations as shown in figure 2.11 (a) and (b), we can restrict our path search algorithm to only use vias in combinations with a sufficiently long wire in preferred direction. But since this costs additional runtime, it is probably best to use this only as a postprocessing step when such a violation actually occurred.

The other case where our definition of λ_w indicates an end edge where in the rectilinear polygon defined by several shapes we do not have such an edge, in practice typically does not waste any routing space at all. The resulting additional minimum distance requirement is dominated by other objects most of the time anyway as shown in figure 2.10.

Overall our approach of handling line end minimum distance rules therefore is valid in practice, which we will confirm in the results in section 3.2.

2.3.5 Generating Shape Class Minimum Distance Rules

Let $p \in P$ and $C_p := \{1, \dots, |\mathcal{E}_p|\}$ the set of shape classes on p as created in section 2.3.3. We will now define a set of shape class minimum distance rules D_p^{sc} to represent the minimum distance requirements $D(w)$ of all wire type elements w on p. If p is a wiring plane, by definition 2.38 and the assignment of shape classes described in section 2.3.3 each shape class $c \in C_p$ uniquely corresponds to

- a function $\lambda_c : \{\text{north}, \text{east}, \text{south}, \text{west}\} \to \{\text{end}, \text{side}\}$,
- an interval $I_c \in \mathcal{I}_p$,
- a set $D_c \subseteq \mathcal{I}_p \times (T_{edge} \dot\cup \{*\}) \times T_{dist} \times \mathbb{N}$.

We set

$$D_p^{sc} := \{(c, c', t_{dist}, d) : c, c' \in C_p,\ (I_{c'}, t_{edge}, t_{dist}, d) \in D_c, \\ t_{edge} = (a, b) \in T_{edge} \implies \lambda_{c'}(a) = b\}. \tag{2.14}$$

This means that the set D_p^{sc} basically is built as follows: For each $c \in C_p$ we process the elements $(I, t_{edge}, t_{dist}, d) \in D_c$ one by one. For each shape class $c' \in C_p$ with the property that $I_{c'} = I$, and either $t_{edge} = (a, b) \in T_{edge} \wedge \lambda_{c'}(a) = b$, or $t_{edge} = *$, we add a shape class minimum distance rule (c, c', t_{dist}, d) to D_p^{sc}.

Similarly, if p is a via plane, each shape class $c \in C_p$ uniquely corresponds to

- a model type $t_c \in T_{shape}$,
- a cut class $v_c \in C_p^{\text{cut}}$,
- a set $D_c \subseteq C_p^{\text{cut}} \times T_{shape} \times T_{dist} \times \mathbb{N}$.

We set

$$D_p^{sc} := \{(c, c', t_{dist}, d) : c, c' \in C_p,\ (v_{c'}, t_{c'}, t_{dist}, d) \in D_c\}. \tag{2.15}$$

With this we finally have completed representing a given set of design rules by wire types, shape classes, and minimum distance rules between these. We now show that this representation is correct.

Lemma 2.39. *Let S, S' be width regular sets of shapes on plane p induced by stick figures with wire types W_R and $W_{R'}$, respectively. Furthermore let $E_R \subseteq D_p^{sc}$ be the set of shape class minimum distance rules we created to represent the minimum distance rule $\delta_p \in R_p \in R$, i.e. the ones originating from our construction in* (2.7).

(a) If each pair of shapes in $s \in S, s' \in S'$ satisfies all shape class minimum distance rules in E_R, then S, S' satisfy δ_p.

(b) If all minimum distance rules have simple run length dependency, and all wire stick figures which induced shapes in S, S' have feasible length, then S, S' satisfy E_R if they satisfy the δ_p.

Proof. To show (a), assume that S, S' violate the minimum distance rule $\delta_p \in R_p$ although there is no pair of shapes in $s \in S, s' \in S'$ violating an element of E_R. By definition of minimum distance rules this means that there must exist non-empty sets $A \subseteq A(S), A' \subseteq A(S')$ and $b, b' \in \mathbb{N}, l \in \mathbb{Z}_{\geq 0}$ with $\text{width}(p, S) \geq b$ for all $p \in A$,

$\text{width}(p', S') \geq b'$ for all $p' \in A', \text{rl}(A, A') \geq l$, and $\text{dist}(A, A') < \delta_p(b, b', l)$. Let $p \in A$ and $p' \in A'$ with $\text{dist}(p, p') < \delta_p(b, b', l)$.

Since S and S' are width regular, there are shapes $s \in S, s' \in S'$ with $p \in A(s), p' \in A(s'), \text{width}(s) = \text{width}(p, S)$, and $\text{width}(s') = \text{width}(p', S')$. Let w, w' be the wire type elements which induced s, s', respectively and $c := c(s)$ and $c' := c(s')$ their shape classes. As mentioned earlier, by construction c, c' uniquely correspond to sets $D_c = D(w), D_{c'} = D(w')$ and intervals $I_c, I_{c'} \in \mathcal{I}_p$ with $\text{width}(w) \in I_c$ and $\text{width}(w') \in I_{c'}$. For $I_{c'} = [i_1, i_2]$ let $d_{l_{\leq 0}} := \delta_p(\text{width}(s), i_1, l_{\leq 0})$ and $d_{l_+} := \delta_p(\text{width}(s), i_1, l_+)$. By the construction of $D(w)$ in (2.7) we know that D_c contains the elements

$$(I_{c'}, *, \text{eucl}, d_{l_{\leq 0}}), \quad (I_{c'}, *, \text{hor}, d_{l_+}), \quad (I_{c'}, *, \text{ver}, d_{l_+}).$$

This implies by (2.14) that E_R contains the shape class minimum distance rules

$$(c, c', \text{eucl}, d_{l_{\leq 0}}), \quad (c, c', \text{hor}, d_{l_+}), \quad (c, c', \text{ver}, d_{l_+}).$$

Because $\text{width}(s) \leq \text{width}(w)$, $\text{width}(s') \leq \text{width}(w')$, and δ_p by definition is non-decreasing in each of its arguments, we have that $\delta_p(b, b', l) \leq d_{l_{\leq 0}}$ if $l \leq 0$, and $\delta_p(b, b', l) \leq d_{l_+}$ otherwise. But then one of these shape class minimum distance rules must be violated by s, s', which contradicts our assumption.

To prove (b), assume that all minimum distance rules have simple run length dependency and all wire stick figures which induced shapes in S, S' have feasible length. The feasible length property of wire stick figures by definition ensures that the widths of wire type elements we used to construct shape classes equal the widths of all of their induced shapes. Together with the fact that we defined the shape class minimum distance rules in E_R as in proposition 2.32 and the simple run length dependency of minimum distance rules the claim follows. □

Similarly, on via planes we have:

Lemma 2.40. *Let s, s' be two via cut shapes on a via plane p induced by via stick figures with wire types W_R and $W_{R'}$, respectively. In addition let $E_R \subseteq D_p^{sc}$ be the shape class minimum distance rules we used to represent the via minimum distance rule $\delta_p^v \in R_p \in R$, i.e. the ones implied by our construction in (2.8). Then s, s' satisfy δ_p^v if and only if they satisfy all shape class minimum distance rules in E_R.*

Proof. Let $w \in W_R, w' \in W_{R'}$ be the wire type elements that induced s, s', and let $c := c(s)$ and $c' := c(s')$ be their shape classes, respectively. By definition 2.38 and the assignment of shape classes, we know that c encodes the shape type cut, the cut class $c_{\text{cut}}(s)$, and the set of distance requirements $D(w)$. Analogously c' encodes the shape type cut, cut class $c_{\text{cut}}(s')$, and distance requirements $D(w')$.

By the construction of $D(w)$ in (2.8) and the conversion in (2.15) the set of shape class minimum distance rules in E_R involving c, c' is:

$$\{(c, c', \text{eucl}, \delta_p^v(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_{\leq 0})),$$
$$(c, c', \text{ver}, \delta_p^v(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_+)),$$
$$(c, c', \text{hor}, \delta_p^v(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_+))\}.$$

Because these rules are obviously satisfied by s, s' if and only if δ_p^v is satisfied by s, s', the lemma follows. □

Similarly, for inter layer via minimum distance rules we have the following lemma:

Lemma 2.41. *Let s, s' be via cut shapes on a via plane p and $p+2$, induced by via stick figures with wire types W_R and $W_{R'}$, respectively. Let $\hat{s}$ be the via above shape on $p+2$ corresponding to s that exists by (2.6). In addition let $E_R \subseteq D^{sc}_{p+2}$ be the shape class minimum distance rules we used to model the inter layer via minimum distance rule $\delta_p^{iv} \in R_p \in R$, i.e. the ones implied by our construction in (2.9) and (2.10). Then s, s' satisfy δ_p^{iv} if and only if $\hat{s}, s'$ satisfy all shape class minimum distance rules in E_R.*

Proof. Analogously to the proof of lemma 2.40 the shape classes $c := c(\hat{s})$ and $c' := c(s')$ correspond to cut classes of $\hat{s}$ and s', and to the shape types of the wire type elements inducing them. Since these shape types must be above for c and cut for c' by construction, the elements of E_R involving c, c' are:

$$\{(c, c', \text{eucl}, \delta_p^{iv}(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_{\leq 0})),$$
$$(c, c', \text{hor}, \delta_p^{iv}(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_+)),$$
$$(c, c', \text{ver}, \delta_p^{iv}(c_{\text{cut}}(s), c_{\text{cut}}(s'), l_+))\}$$

By construction of wire type elements with shape type above in (2.6), $\hat{s}$ simply equals s projected to $p+2$ (up to shape class). Therefore these shape class minimum distance rules are satisfied by $\hat{s}, s'$ if and only if δ_p^{iv} is satisfied by s, s'. □

The following lemma shows that we do not violate line end minimum distance rules if we estimated end edges correctly.

Lemma 2.42. *Let S, S' be two sets of shapes on a wiring plane p with $A(S) \cap A(S') = \emptyset$ such that all $s \in S$ are induced by stick figures with wire type W_R, and all $s \in S'$ are induced by stick figures with wire type $W_{R'}$. Let e be a t_e edge in $\text{plg}(S)$ and e' a $t_{e'}$ edge in $\text{plg}(S')$ for $t_e, t_{e'} \in \{\text{end}, \text{side}\}$. Define*

$$S_e := \{s \in S : \exists\, d \in \{\text{north}, \text{east}, \text{south}, \text{west}\} : \text{edge}(s, d) \cap e \neq \emptyset\},$$

and $S'_{e'}$ analogously. Consider the set E_R of shape class minimum distance rules resulting from our construction in (2.11) - (2.13) to represent the line end minimum distance rule $\delta_p^{le} \in R_p \in R$.

Suppose we have the following property:

(i) For all shapes $s \in S, s' \in S', d \in \{\text{north}, \text{east}, \text{south}, \text{west}\}$ *it holds that*

$$\begin{aligned} \text{edge}(s,d) \cap e \neq \emptyset &\implies \lambda_w(d) = t_e, \\ \text{edge}(s',d) \cap e' \neq \emptyset &\implies \lambda_w(d) = t_{e'}. \end{aligned}$$

Then e, e' *satisfy* δ_p^{le} *if and only if* $S_e, S'_{e'}$ *satisfy all shape class minimum distance rules in* E_R*.*

Proof. Let e and e' be edges of $\text{plg}(S)$ and $\text{plg}(S')$, respectively, such that property (i) holds. To show sufficiency, assume that E_R is satisfied by $S_e, S'_{e'}$, but δ_p^{le} is violated by e, e'. We can assume w.l.o.g. that $t_e = \text{end}$ and e, e' both are horizontal: If both were side edges or orthogonal to each other, δ_p^{le} would not be violated. Clearly there must exist $s \in S_e, s' \in S'_{e'}$ and $d, d' \in \{\text{north}, \text{south}\}$ with $\text{edge}(s,d) \cap e \neq \emptyset$ and $\text{edge}(s',d') \cap e' \neq \emptyset$. W.l.o.g. let $d = \text{north}$ which implies that $d' = \text{south}$. By (i) we must have $\lambda_w(d) = t_e = \text{end}$ and $\lambda_{w'}(\text{south}) = t_{e'}$ for the wire type elements w and w' that induced s and s', respectively. By the construction in (2.11) - (2.13) the set $D(w)$ contains the element

$$([0,\infty], (\text{south}, t_{e'}), \text{north}, \delta_p^{le}(t_{e'})).$$

Definition 2.38 (ii) and the construction in (2.14) imply that there is a shape class minimum distance rule

$$r := (c(s), c(s'), \text{north}, \delta_p^{le}(t_{e'})) \in E_R.$$

Since δ_p^{le} is violated, we have $\text{rl}(e,e') > 0$ and $\text{dist}(e,e') < \delta_p^{le}(t')$, which implies that r is violated by s, s', which contradicts our assumption.

To show necessity, assume that we have (i) and δ_p^{le} is satisfied by e, e', but $S_e, S_{e'}$ violate E_R. So there are shapes $s \in S_e, s' \in S_{e'}$ violating a shape class minimum distance rule $r \in E_R$. W.l.o.g. let $r := (c(s), c(s'), \text{north}, \delta_p^{le}(t_r))$, where $t_r \in \{\text{side}, \text{end}\}$. We must have $\lambda_w(\text{north}) = \text{end}$ and $\lambda_{w'}(\text{south}) = t_r$ for the wire type elements w and w' that induced s and s', respectively. Otherwise r would not have been constructed. By (i), $t_e = \lambda_w(\text{north}) = \text{end}$ and $t_{e'} = \lambda_{w'}(\text{south}) = t_r$. Because r is violated, e, e' must have positive run length, $e \in \text{plg}(S)_{\text{north}}$, $e' \in \text{plg}(S)_{\text{south}}$ and $\text{dist}(s,s') < \delta_p^{le}(t_r)$. But this means that e, e' violate δ_p^{le}, which contradicts our assumption.

□

With this we know that in our representation violations of line end minimum distance rules can only happen for a reduced set of edges:

Corollary 2.43. *Let* S, S' *be as in lemma 2.42. If* S, S' *satisfy all shape class minimum distance rules in* D_p^{sc}*, then each* $e \in \text{plg}(S), e' \in \text{plg}(S')$ *violating a line end minimum distance rule* $\delta_p^{le} \in R_p \in R$ *must be running in preferred direction of* p*.*

Proof. The corollary follows directly from our construction of λ_w in 2.3.4 and the proof of sufficiency of lemma 2.42. □

Overall we have the following main theorem stating that with only few exceptions the design rules discussed here are satisfied by construction if we compute a routing satisfying all shape class minimum distance rules.

Theorem 2.44. *Consider two rule sets R, R'. Let S, S' be two sets of shapes induced by stick figures with wire types W_R and $W_{R'}$, respectively, and let S_p and S'_p be their shapes on plane p.*

S, S' satisfy all rules in R, R' with the exceptions regarding line end minimum distance rules stated in corollary 2.43 if S_p, S'_p satisfy all shape class minimum distance rules in D_p^{sc}, and are width regular if $p \in P_{wiring}$.

Proof. The theorem follows directly from lemma 2.39(a), 2.40, 2.41, and corollary 2.43. □

2.3.6 Reducing the Number of Shape Classes

Note that the number of shape classes we created to represent the given minimum distance requirements may not be minimum. For example there could be minimum distance rules requiring the same minimum distance for widths contained in different width intervals. In addition it is possible and common that there are minimum distance requirements where one dominates the other. Therefore there can be different shape classes representing the same minimum distance requirements.

Definition 2.45. *We call two shape classes $c, c' \in C_p$* equivalent *if for each shape class minimum distance rule $r \in D_p^{sc}$ containing c we have that there is a $r' \in D_p^{sc}$ which results from replacing c by c' in r.*

Although such equivalent shape classes and dominated shape class minimum distance rules generally are no problem for BonnRoute, it is still desirable to get rid of them in order to save memory and runtime. Note for example that the number of different wire types also can be decreased by ensuring that there are no equivalent shape classes anymore. In section 2.4 we discuss how shape class minimum distance rules are checked in BonnRoute. Since this checking involves a data structure called *fast grid* (Müller [2009]) which maintains precomputed information for a restricted set of wire types, it is important to keep the total number of different wire types as small as possible.

We first have to identify which shape class minimum distance rules are dominated within D_p^{sc}.

Proposition 2.46. *Let $r := (c_1, c_2, t, d)$ and $r' := (c_1, c_2, t', d')$ be shape class minimum distance rules. Then r dominates r' if and only if $d \geq d'$ and at least one of the following conditions hold:*

(i) $t = t'$

(ii) $t \in \{\text{hor}, \text{eucl}\} \wedge t' \in \{\text{east}, \text{west}, \text{hor}\}$

(iii) $t \in \{\text{ver}, \text{eucl}\} \wedge t' \in \{\text{north}, \text{south}, \text{ver}\}$

Proof. Sufficiency is clear because any of the three properties together with $d \geq d'$ implies that $A(s, r) \supseteq A(s, r')$ for every shape s.

To show necessity assume r dominates r' but (i)-(iii) do not hold or we have $d < d'$. The latter clearly would be a contradiction to r dominating r', and if (i)-(iii) do not hold, then we must have one of the following cases:

- $t \in \{\text{hor}\} \wedge t' \in \{\text{north}, \text{south}, \text{ver}, \text{eucl}\}$
- $t \in \{\text{ver}\} \wedge t' \in \{\text{east}, \text{west}, \text{hor}, \text{eucl}\}$
- $t \in \{\text{north}, \text{east}, \text{south}, \text{west}\} \wedge t' \in T_{dist} \setminus \{t\}$

But each case implies that for every shape s we have $A(s, r') \setminus A(s, r) \neq \emptyset$, which contradicts r dominating r'. □

This means we can easily check for each $r \in D_p^{sc}$ if it is dominated by a any other element in D_p^{sc} and remove r from D_p^{sc} if this is the case. Then for each two equivalent shape classes $c, c' \in C_p$ we can replace c by c' in each overhang shape with shape class c of a wire type element on p. We remove c from C_p and all shape class minimum distance rules from D_p^{sc} containing c.

After these operations the resulting set of shape classes C_p and minimum distance rules D_p^{sc} do not contain equivalent or dominated elements anymore, but they still represent the same minimum distance requirements as before by definition of equivalence and dominance.

2.3.7 Runtime Analysis

The main routine of the BonnRouteRules module is building the representation of the given design rules with wire types and shape class minimum distance rules as described in the previous sections. It can be summarized as in algorithm 1.

Theorem 2.47. *Algorithm 1 computes $\mathcal{W}$, and $\{D_p^{sc} : p \in P\}$, such that theorem 2.44 holds for all $R, R' \in R$ and $W_R, W_{R'} \in \mathcal{W}$. Assuming that each design rule in the given rule sets has constant size, its runtime is $\mathcal{O}\left(|P|\,|\mathcal{R}| \log |\mathcal{R}| + \sum_{p \in P} |C_p|^2\right)$.*

Proof. As the algorithm proceeds exactly as in sections 2.3.2 to 2.3.6, the correctness is clear. To prove the runtime, observe that each wire type created in line 3 contains $\mathcal{O}(|P|)$ elements, and each of these elements can be constructed in constant time. For the entire loop we therefore have a runtime of $\mathcal{O}(|P|\,|\mathcal{R}|)$. The time needed for generating shape classes in lines 5 and 6 is $\mathcal{O}(|P|\,|\mathcal{R}| \log |\mathcal{R}|)$, and creating all shape class minimum distance rules in lines 7 to 9 takes $\mathcal{O}(\sum_{p \in P} |C_p|^2)$ time. Since also the elimination of equivalent shape classes in line 10 can be done in this time, summing up yields the desired runtime. □

Algorithm 1: BUILD WIRING RULE REPRESENTATION

Input : A set $\mathcal{R}$ of rule sets.
Output: A set of wire types $\mathcal{W} = \{W_R : R \in \mathcal{R}\}$, and a set D_p^{sc} of shape class minimum distance rules on shape classes $C_p \subset \mathbb{N}$ for each $p \in P$.

1 Set $\mathcal{W} := \emptyset$ and $D_p^{sc} := \emptyset$ for all $p \in P$.
2 **for** $R \in \mathcal{R}$ **do**
3 Create wire type W_R from R as in section 2.3.2.
4 $\mathcal{W} := \mathcal{W} \cup W_R$
5 Build equivalence classes $W_p^1, \dots, W_p^{l_p}$ of $\{(p,t,o) \in \bigcup_{W \in \mathcal{W}} W\}$ by $\sim$ as in section 2.3.3 for each $p \in P$.
6 Set $c(o) := i$ for each $(p,t,o) \in W_p^i$ for $i \in C_p := \{1, \dots, l_p\}$.
7 **for** $p \in P$ *and* $i, j \in C_p$ **do**
8 Set $D_{p,i,j}^{sc}$ to the set of shape class minimum distance rules based on $D(w_i), D(w_j)$ for arbitrary $w_i \in W_p^i$, $w_j \in W_p^j$ as in section 2.3.5.
9 Set $D_p^{sc} := D_p^{sc} \cup D_{p,i,j}^{sc}$.
10 Remove dominated elements from D_p^{sc} and eliminate equivalent shape classes as in section 2.3.6 for all $p \in P$.

Experimental results in section 2.3.10 will show that with this approach the runtime of the BonnRouteRules module in practice is insignificantly small. In section 2.3.9 we will describe how the set $\mathcal{R}$, which is part of the input of the module, is generated.

2.3.8 Further Aspects

Blockages

So far we only discussed how wires are represented in BonnRoute. Pins and blockages are handled similarly, therefore we only describe the main differences in this section.

In terms of their representation we do not distinguish between pins and blockages. For both the important thing is that they are part of the input of BonnRoute, cannot be changed, and have no kind of stick figure representation in contrast to wires. In order to represent their minimum distance requirements defined in the technology design rules we create a set of *blockage models* for each plane.

Let p be a wiring plane. A blockage model on p is a 3-tuple (I, λ, c) where I is a width interval in $\mathcal{I}_p$ as defined in section 2.3.2, $\lambda : \{\text{north}, \text{east}, \text{south}, \text{west}\} \to \{\text{end}, \text{side}\}$ a function indicating end edges as in section 2.3.4, and $c \in \mathbb{N}$ a shape class. We define c and appropriate shape class minimum distance rules such that they represent the minimum distance requirements imposed by the technology design rules for blockages whose width is contained in I and edge types determined by λ. Basically this can be done similarly as in the previous sections. The only difference is that we can model the distance requirements

more precisely by creating a fitting blockage model for each specific blockage situation. We simply create one blockage model $m_p^{I,\lambda} = (I, \lambda, c_{I,\lambda,p})$ for each I, λ, and p to be sure to cover all possible cases.

The more difficult task then, however, is to correctly assign blockage models to each given connected set of blockage shapes B_p for $p \in P_{wiring}$. We can easily determine all end edges in $\mathrm{plg}(B_p)$, the more difficult part is to determine widths exactly. Generally, we cannot assume that B_p is width regular in the sense of definition 2.28, so just inspecting $\mathrm{width}(b)$ for each $b \in B$ may not be correct in some cases. We have to solve the following decomposition problem: Given a set of shapes S, and a set $\mathcal{I}$ of disjoint intervals in $\mathbb{N}$, find for each $I \in \mathcal{I}$ a set of shapes S_I such that $\bigcup_{I \in \mathcal{I}} A(S_I) \cap \mathbb{Z}^2 = A(S) \cap \mathbb{Z}^2$, and $\mathrm{width}(S, q) \in I$ for all $q \in A(S_I)$.

We solve this problem for $S = B_p$ and $\mathcal{I} = \mathcal{I}_p$ by a sweep-line algorithm to obtain a set B_p^I of blockage shapes for each $I \in \mathcal{I}_p$, and assign the blockage model $m_p^{I,\lambda}$ to all $b \in B_p^I$, where λ represents the edge types of edges of $\mathrm{plg}(B_p)$ intersected by b.

Complex Design Rule Variants

The complex design rule variants remarked at the end of section 2.2.2 are also handled by the BonnRouteRules module. For legacy reasons, however, BonnRoute currently only supports the types of shape class minimum distance rules defined in section 2.3.1, and for each of them an additional type only applying to so called *expanded shapes*. Wire type elements and stick figures actually can for each plane define not only a (real) shape describing the actual metal area, but also an additional expanded version of that shape. This concept originates from earlier BonnRoute versions, where minimum distance rules where checked differently. Instead of considering euclidean distances directly, only horizontal and vertical distances of expanded shapes were checked, using a pattern based approach. In the current implementation expanded shapes are still used to model design rules which actually hold for shapes or edges expanded in some fashion. While this currently works reasonably well in practice with some minor pessimism involved, it is not safe to rely on that. The fundamental problem with this approach is that in BonnRoute these expansions are part of a wire type although they actually belong to individual design rules. Handling various different expansions of multiple design rules with only one expanded shape is a difficult task, and may not work well anymore if design rules change. Moreover, expanding edges or shrinking shapes is not supported accurately by this approach.

Therefore, we propose to naturally extend shape class minimum distance rules such that it is possible to specify for each of the two shape classes some kind of shape modification. This modification then is applied to each shape with this shape class during the checking procedure performed by the checking module, described in section 2.4.3. At least simple modifications such as restricting a shape to some edge, and performing certain expand or shrink operations should be possible in practice without a significant runtime impact.

Input Wiring

Often there are some nets which are already routed at the time when BonnRoute is used. One typical example are some types of clock nets which have to be routed in a specific way ensuring certain timing properties. Such nets are routed already before the normal signal nets in a separate step before BonnRoute. The BonnRouteRules module then of course has to provide appropriate wire types for each existing wire and via and somehow ensure that these get assigned correctly. Generating the appropriate wire types can basically be done as described in the previous sections after determining the rule set containing the design rules holding for each wire and via. The only thing that is different is that stick figure overhangs and via definitions are predefined and do not have to be computed. In section 2.3.9 we will describe how the assignment of the resulting wire types to existing wires and vias works.

2.3.9 Implementation

The BonnRouteRules module was implemented by the author in the C++ programming language (Stroustrup [2000]), and is currently used in practice for server and ASIC designs in 32 nm and 22 nm technologies. To make this possible the module must be able to deal with several different data models and complex design rule specifications. Moreover it has to be robust against incomplete or inconsistent data, which can happen easily in this complex environment. Note that detailed routing with BonnRoute is just one of many steps in the overall automatic design flow, which involves several different tools, and changes consistently. In this section we first describe how to obtain all necessary input data needed by the BonnRouteRules module in the environment where it is currently used. We then continue by discussing the basic structure of the module itself.

Collecting Input Data

In the design flow in which BonnRoute is used today design data is stored in the OpenAccess data model (Silicon Integration Initiative [2012]). Similar to BonnRoute, OpenAccess represents wires by axis parallel line segments (*route segments*). Each route segment s is annotated with non-negative numbers defining a shape $o(s)$, which represents overhangs such that the actual shape representation of s is $s + o(s)$. A via s consists of a point shaped line segment s and a reference $v(s)$ to a via definition, which defines its shapes on all three affected planes.

In terms of design rules, in OpenAccess we are not directly given a set of design rules for each net. The assignment of design rules to objects is actually more complex: Design rules are first grouped in so called *constraint groups*, which then can be assigned to various objects, e.g. nets and route segments or groups of these, respectively. In addition there is one distinguished constraint group, often called foundry constraint group, containing all the design rules originating from the manufacturing process. The design rules

contained in this constraint group hold for all nets, the requirements can only be tightened, but not relaxed, for individual nets. The design rules that hold for a given object can be distributed over the constraint group of the object itself, the constraint group of its containing objects, and the foundry constraint group. In case of multiple occurrences of the same kind of design rule, a specific, *hierarchical order* of the objects determines which design rule actually applies.

The main part of the input of the BonnRouteRules module is a set of rule sets $\mathcal{R}$, which then is converted to wire types as in algorithm 1 in section 2.3.7. We build each of these rule sets using the given OpenAccess data as follows: From the object we want to construct the rule set for, and from its parent objects within the object hierarchy, we obtain a finite sequence c of constraint groups. For each kind of design rule separately, we query the constraint groups in c one by one, and in each case add the first found design rule to the rule set. In addition we need information about the kinds of existing route segments and vias for which these rules hold, so that we can create appropriate wire types for them. We therefore collect for each rule set R a set of shapes O_R containing the overhangs of existing route segments, and a set V_R of via definitions of existing vias. Finally, to be able to assign wire types correctly, we need to remember for each rule set the constraint group sequence that is was build from. Therefore we build an injective mapping $\phi : \mathcal{R} \to \mathcal{C}$, where $\mathcal{C}$ is the set of finite sequences of constraint groups. All this input data collection is done by algorithm 2.

Of course we can implement algorithm 2 efficiently:

Proposition 2.48. *Algorithm 2 can be implemented in $\mathcal{O}(|N| + |W| \log |\mathcal{R}|)$ time.*

Proof. The procedure `FindOrCreateRuleSet` can be implemented using a balanced search tree in $\mathcal{O}(\log |\mathcal{R}|)$ time. Since it is called $1 + |W_n|$ times for each $n \in N$ we have the desired runtime. □

Our implementation is written in the scripting language Tcl (Ousterhout and Jones [2009]), which fits well in the overall flow environment, keeps the code simple, and makes this part easily adaptable. The downside of this is higher runtime compared to a compiled programming language.

Basic Structure of the Module

The basic structure of the BonnRouteRules module is depicted in the UML® class diagram in figure 2.12. See (Object Management Group [2012]) for the specification of the Unified Modeling Language (UML®).

The module provides three interfaces. The *DesignRuleInterface* class provides methods to input all necessary kinds of design rules into the module and is implemented by the *DesignRuleManager* class. As mentioned in the previous sections, the design rule data comes from OpenAccess, but the interface does not depend on that. In particular the sets $\mathcal{R}$, and O_R, V_R for $R \in \mathcal{R}$, and the mapping ϕ, as defined in the previous section, are

Algorithm 2: BUILD RULE SETS

Input : A set of nets N, and a set $\{W_n : n \in N\}$ of sets of route segments and vias. A set of constraint groups C containing the foundry constraint group c_{foundry}. Constraint group assignments $\gamma : N \cup W \to C \cup \{\varnothing\}$, where $W := \bigcup_{n \in N} W_n$, and $\varnothing$ denotes that no constraint group is assigned.

Output: A set of rule sets $\mathcal{R}$, and for each $R \in \mathcal{R}$ a set of shapes O_R and via definitions V_R. A function $\phi : \mathcal{R} \to \mathcal{C}$, where $\mathcal{C}$ is the set of sequences $(c_1, \ldots, c_k)$ with $c_i \in C$, $i \in \{1, \ldots, k\}$ and $1 \leq k \leq 3$.

```
Set R := ∅
for n ∈ N do
    Set k := 1, c_1 := c_foundry.
    if γ(n) ≠ ∅ then
        Set k := 2, c_2 := γ(n).
    Set R := FindOrCreateRuleSet((c_1, ..., c_k), R).
    for w ∈ W_n do
        if γ(w) ≠ ∅ then
            Set R := FindOrCreateRuleSet((c_1, ..., c_k, γ(w)), R).
        if w is a route segment then
            Set O_R := O_R ∪ {o(w)}
        else if w is a via then
            Set V_R := V_R ∪ {v(w)}
```

Procedure `FindOrCreateRuleSet` $((c_1, \ldots, c_l), \mathcal{R})$

```
if ∃R ∈ R with φ(R) = (c_1, ..., c_l) then
    Set R := φ^{-1}((c_1, ..., c_l)).
else
    Build a rule set R from (c_1, ..., c_l).
    Set φ(R) := (c_1, ..., c_l).
    Set R := R ∪ {R}
return R
```

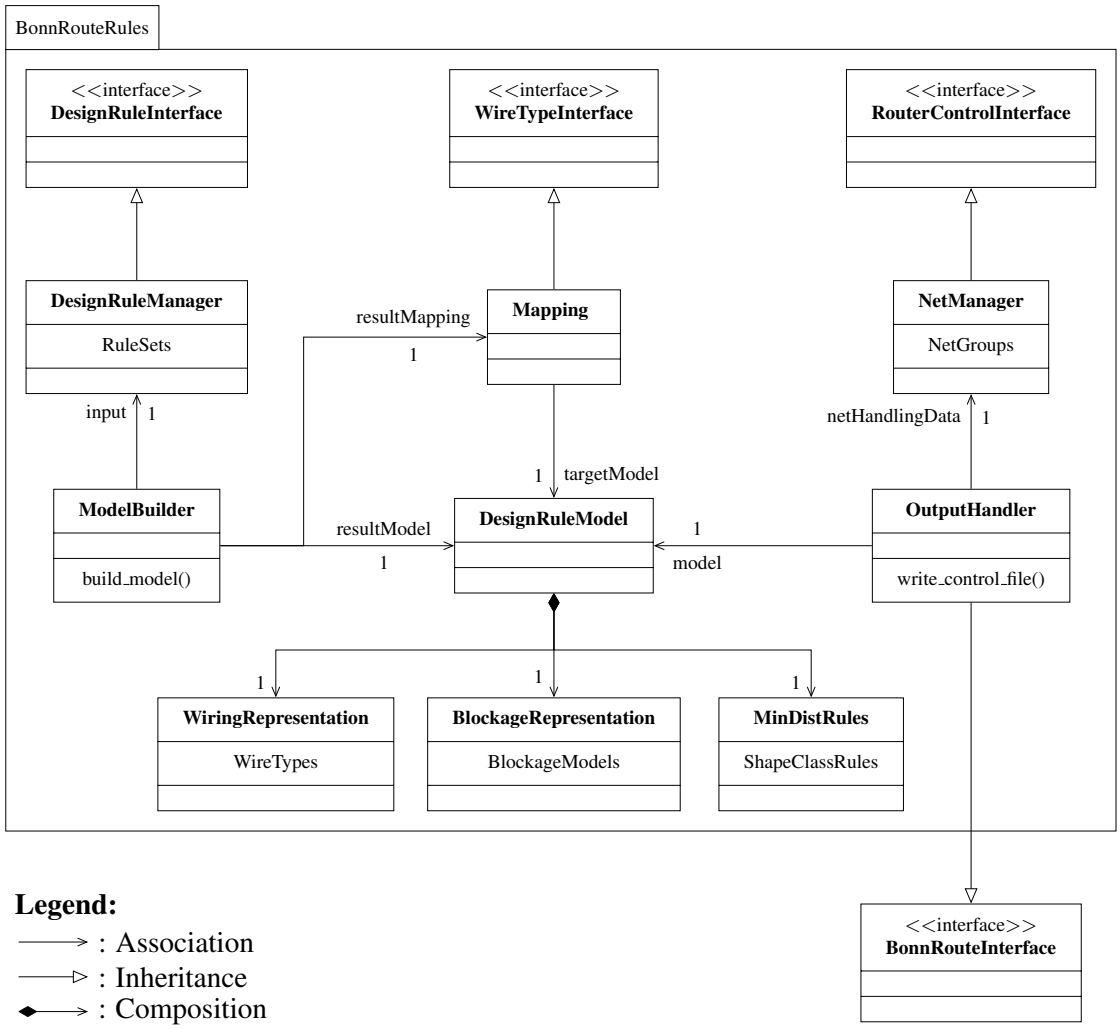


Figure 2.12: UML® class diagram of the BonnRouteRules module. Some arrows are annotated with a multiplicity number and a role name.

passed to the module via this interface. The purpose of the *RouterControlInterface* class is to provide methods to the user to specify the set of nets to be routed, their priorities, and preferred wiring planes. Finally, the *WireTypeInterface* is able to manage requests in terms of mapping a sequence of constraint groups to an appropriate wire type, and vice versa.

The modeling task as described in the previous sections, including algorithm 1, is realized by the *ModelBuilder* class. This class contains methods to build a *DesignRuleModel* consisting of classes describing the BonnRoute representation of wires, blockages and their minimum distance requirements. It also builds a *Mapping* object, which implements the RouterControlInterface by using the function ϕ and internal structures mapping wire types to the rule sets they were created from. After all modeling work has been done, the resulting representations and user defined net handling data is passed to BonnRoute by an *OutputHandler* object. Currently this still is done via a file interface for legacy reasons and convenient debugging possibilities, but a direct data handover could also be realized.

The design of the module ensures that data structures and algorithms are well separated, and have clearly defined interfaces. Changes in the modeling process can be done by reimplementing the ModelBuilder class without affecting many other parts of the module. Furthermore there are no strong dependencies to the outside environment. A change from OpenAccess to any other data model with similar concepts would not be difficult. Generally, making changes as easy as possible is very important in our case because the design rule handling is one of the first things that have to be adapted for each new technology. Even during the lifetime of an existing technology the design rules often change in order to react to experiences gained in the production process.

2.3.10 Experimental Results

To give an impression on the result and runtime of the BonnRouteRules module in practice we present some statistics on eight 22 nm server chips and eight 32 nm ASICs. Table 2.1 reports for each of the chips the number of wire type elements (i.e. the sum of the cardinalities of all wire types), the number of wire types, shape classes, and shape class minimum distance rules. The table also shows the runtime needed for algorithm 1, which creates all these structures, as well as the total runtime including the generation of the rule sets, which is done in algorithm 2.

One can see that on most chips the module created several thousand wire type elements building several hundred wire types. The high number of wire types is due to the fact that many of them just model one via and no wires at all. Such wire types are only used for pin access and postprocessing in combination with other wire types.

The total number of shape classes on each chip is below two hundred, and the minimum distance requirements are modeled by around two thousand shape class minimum distance rules on each instance. One can also see that some of the ASIC instances are parts belonging to the same chip and therefore have very similar or even equal design rules which is reflected in the BonnRouteRules output. Also interesting is that on the

Chip	Tech. (nm)	Nets	WTE	WT	SC	SCR	Runtime (sec)	
							Alg. 1	Total
S1	22	116,257	1,799	353	169	2,449	0.015	59
S2	22	136,573	2,226	366	160	2,014	0.016	71
S3	22	155,092	3,096	593	168	2,087	0.025	79
S4	22	438,328	3,177	723	156	1,630	0.022	152
S5	22	466,157	2,364	460	156	1,630	0.019	159
S6	22	501,875	3,414	682	163	1,751	0.028	171
S7	22	527,465	4,710	844	181	2,630	0.032	194
S8	22	604,213	6,130	1,499	148	1,358	0.043	214
A1	32	215,272	459	115	157	2,231	0.006	123
A2	32	648,023	1,080	284	157	2,231	0.012	195
A3	32	909,922	1,080	284	157	2,231	0.010	323
A4	32	985,565	1,080	284	157	2,231	0.016	324
A5	32	989,834	1,080	284	157	2,231	0.012	317
A6	32	1,252,364	1,080	284	157	2,231	0.011	390
A7	32	1,283,905	1,200	324	157	2,231	0.013	392
A8	32	1,650,584	459	115	157	2,231	0.006	455

Table 2.1: Total number of wire type elements (WTE), wire types (WT), shape classes (SC), and shape class minimum distance rules (SCR). The runtime column shows the runtime of algorithm 1 and the total runtime including algorithm 2.

p	WTE	SC	$\frac{\text{SC}}{\text{WTE}}(\%)$	SCR
0	94	6	6.38	36
1	64	5	7.81	92
2	178	10	5.62	216
3	147	11	7.48	163
4	189	13	6.88	322
5	156	12	7.69	175
6	173	13	7.51	322
7	143	12	8.39	175
8	235	13	5.53	322
9	205	9	4.39	39
10	264	4	1.52	16

Table 2.2: Number of wire type elements (WTE), shape classes SC, and shape class minimum distance rules (SCR) on some planes $p \in P$ of chip S3. Note that if p is even, we have a wiring plane, and if p is odd, we have a via plane.

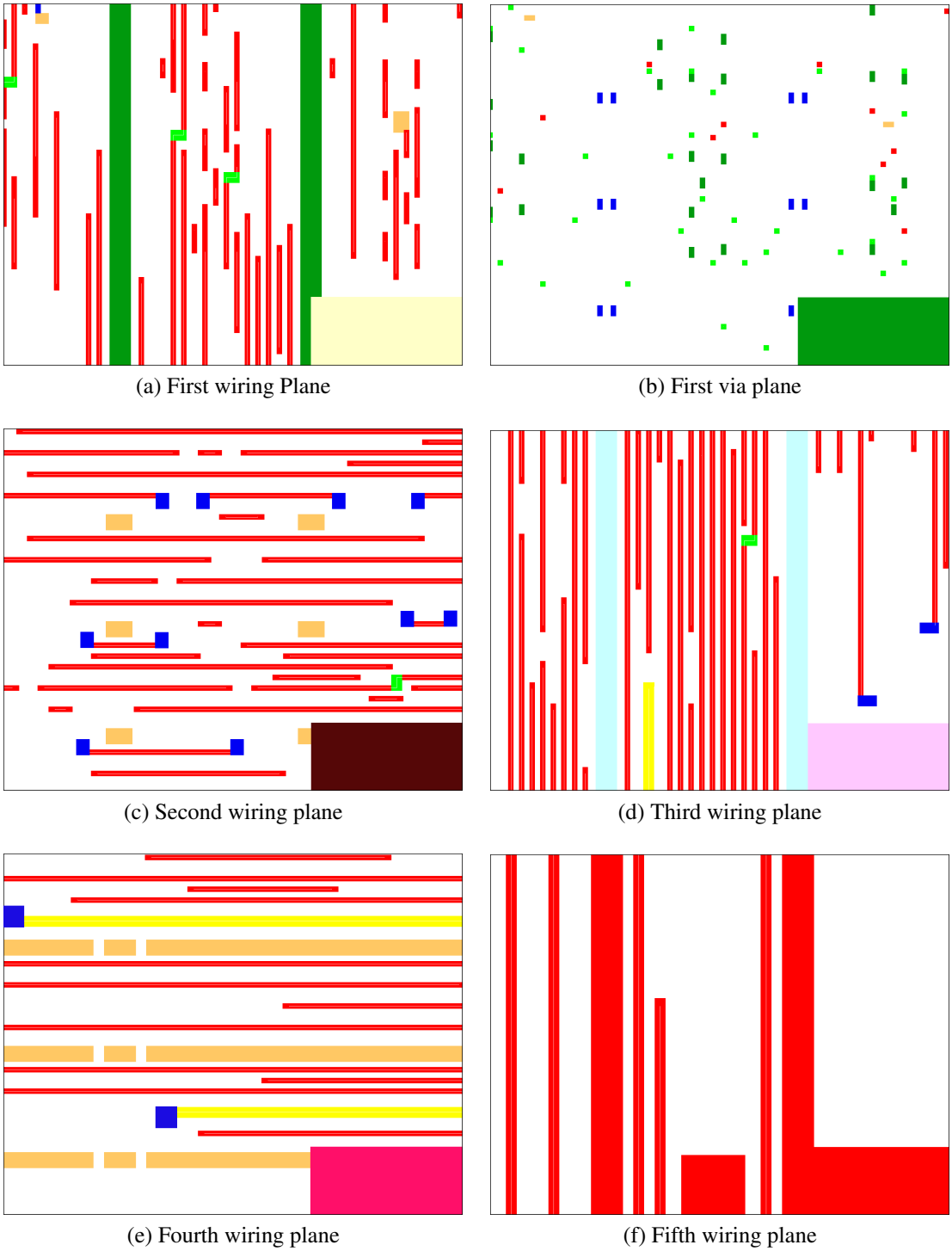
22 nm server designs the number of wire types is considerably higher than on the ASIC instances because of more user defined design rules in addition to the ones originating from manufacturing.

The runtime columns show that the runtime of algorithm 1 is insignificantly small in practice since it always stays way below one second. The total runtime including algorithm 2 is much larger, mainly because it is implemented in Tcl. This runtime can be significantly reduced by identifying the few parts that cost most of the runtime and reimplementing those using a compiled programming language. But compared to the total runtime of the whole routing flow, even the current total runtime of the BonnRouteRules module is very small. We will see this in chapter 3, where we present experimental results of the complete routing flow on the same 16 test instances.

In table 2.2 one can see how the numbers are distributed over the different planes considering the chip S3 as an example. While the number of wire type elements per plane is between 60 and 270, we have only between 4 and 13 shape classes which is below 9% on each plane. On the highest planes these numbers typically decrease because there the design rules are simpler. On via planes the number of shape class minimum distance rules is considerably smaller compared to wiring planes. The reason is that we only have via minimum distance rules on such planes and no other kinds of rules such as line end minimum distance rules. In addition the number of different cut classes that can be used is quite small, so there are not many different kinds of shapes on these planes.

Note that all of these statistics include the representation of some design rule variants not discussed in this thesis. Overall, one can summarize that the methods presented here provide a quite compact and efficient representation of the design rules of current technologies.

Finally, in figure 2.13 we give an optical impression of the shape classes of the shapes within a small area of a 22 nm design. The figure shows that most of the wiring consists of minimum width on-track wires, which can be packed densely and all have the same shape class. Other shape classes are assigned to wider wires, larger via shapes, and big blockages which all need more space. The shapes on the highest wiring plane shown in the figure all have the same shape class. This again confirms that design rules on such planes are considerably simpler compared to the lower planes. Generally, wires also have to be much wider on higher planes.



(a) First wiring Plane

(b) First via plane

(c) Second wiring plane

(d) Third wiring plane

(e) Fourth wiring plane

(f) Fifth wiring plane

Figure 2.13: A small area of a 22 nm server design over several planes containing shapes colored by shape class. Most shapes represent wires of minimum width running in preferred direction (red shape class), and there are only very few jogs (light green shape class). One can see several larger vias (blue shape class) as well as wider wires requiring more spacing (yellow shape class). Figure (b) shows a via plane with several different via cut shapes, each having one of the few allowed cut classes.

2.3.11 Outlook: Handling DPT Design Rules

In this section we give an outlook on how the new kinds of distance rules occurring in future double patterning technologies (DPT) can be incorporated in our distance rule model, and handled in BonnRoute. As described in section 2.2.4, the DPT distance rules also depend on the assigned masks of shapes, which can be regarded as a color assignment. This means that routing tools must assign a color to each wire shape such that all distance rules are satisfied, or at least it must be ensured that such an assignment exists. It is still unclear how blockage and pin shapes are handled in a DPT routing flow, i.e. if the coloring of these shapes is predetermined, or can also be chosen by the routing tool.

In BonnRoute the assignment of colors to wire shapes can be naturally encoded by wire types generated by the BonnRouteRules module as follows. We extend the definition of wire type elements such that each wire type element in addition to the overhang and shape class also contains a color. Then for each rule set we create two instead of one wire type, where the first one contains only wire type elements with the first color, and the second only wire type elements with the second color. Of course when generating shape classes as in section 2.3.3, we have to take the color of each wire type element w into account for creating the set $D(w)$ of distance requirements, and also use it as a further property for defining equivalence in definition 2.38.

Having such two wire types corresponding to the two colors, we can let our path search algorithms choose between these at any time. I.e. we label a node in our Dijkstra based algorithms if at least one of these wire types can be used without shape class minimum distance rule violations at the corresponding location. Of course it would not be a good approach to let these algorithms choose wire types (e.g. colors) arbitrarily: Although each path search will ensure that the found path together with the already existing shapes admits a feasible coloring, it does not care about making the coloring of subsequent paths hard, or even impossible. An interesting question is how to guide the router to use these wire types efficiently such that in the end we obtain a complete coloring without many time consuming rip-up and reroute sequences.

One simple, but probably most practical, way to achieve this, is to fix for each track in an alternating manner one of the two colored wire types to be used for all stick figures on that track. Of course this only makes sense if we define tracks such that neighboring tracks admit placing stick figures running in preferred direction with differently colored wire types without creating a minimum distance rule violation. Analogously we need that tracks with a common neighboring track admit placing stick figures with equally colored wire types. To achieve this track-wise coloring one can actually route with just one artificial, third wire type with appropriate minimum distance requirements, which guarantee the existence of the desired track based coloring. These distance requirements basically have to ensure two things (illustrated in figure 2.14):

- The induced shapes of on-track stick figures running in preferred direction on the same track have to keep enough distance such that they can be colored equally.
- For each jog stick figure connecting two points on neighboring tracks there must

be enough space next to at least one end of the induced shape. Then by using an appropriate stitch one can maintain the track based coloring.

With these two properties the two colored wire types then can be assigned to stick figures afterwards in a post processing step accordingly.

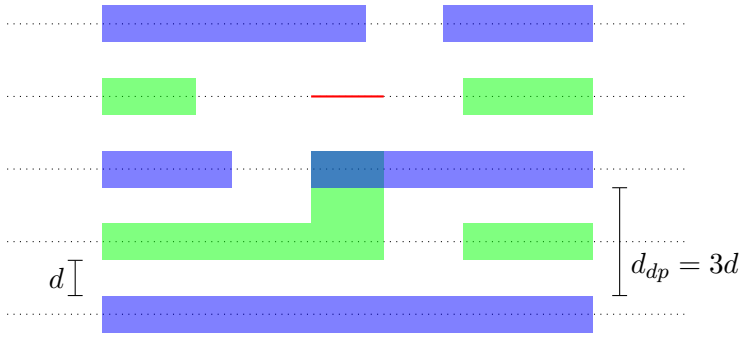


Figure 2.14: Example of a track based coloring where we have minimum vertical distance d for differently colored shapes of minimum width (which also equals d), and minimum vertical distance $3d$ for the case with equal colors. The uniform tracks with pitch $2d$ admit alternatingly using only one of the two colored wire types on each track. Note that the minimum distance between shapes on the same track (which have equal color), and the additional blocked track at jogs (red) is needed to guarantee this kind of coloring while actually routing only with one artificial wire type and assigning colors later.

While this simple approach with its efficient, dense packing of wires seems promising, we still have to verify that in practice on the upcoming 14 nm DPT designs. The success of this of course depends on whether stitching becomes restricted, and on how well pin and blockage shapes in the input can be colored. Also wire types with shapes of larger width may be problematic in this track based coloring approach. Probably, more complex additional methods will be necessary, for example a generalized version of the on-track path search algorithm in BonnRoute, which supports more sophisticated restrictions on wire type usage by a multi-labeling approach.

2.4 Checking Distance Rules

In this section we describe how the shape class minimum distance rules generated in section 2.3 are checked efficiently in BonnRoute. The question if a certain wire shape can be placed at some position without violating any of these rules with respect to the existing shapes has to be answered dozens of times in every routing run. Therefore efficient handling of such queries is essential for achieving good overall runtime.

2.4.1 General Concept

In BonnRoute we use a path search algorithm even if long distances have to be covered. To be able to check for minimum distance rule violations efficiently, we need a *shape data structure* for determining the subset of all wire, pin and blockage shapes S within the whole chip area which intersect some given query area.

Given a shape q, such *range queries* consist of determining the set $S_q := \{s \in S : A(s) \cap A(q) \neq \emptyset\}$. We discuss such data structures in section 2.4.2. With this the legality of some given shape s on a plane p can be checked as follows: The *checking module* of BonnRoute determines a set Q_s of shapes such that for all shapes $s' \in S$ which violate a shape class minimum distance rule in D_p^{sc} together with s we have $A(s') \cap A(Q_s) \neq \emptyset$. It then performs a range query for all $q \in Q_s$ and checks for each of the resulting shapes if it violates any rule in D_p^{sc} together with s. We describe the checking module in section 2.4.3.

Since by far most legality queries are issued by the on-track path search algorithm which only uses on-track stick figures, we have a second, very fast data structure for this special case. The so called *fast grid* developed by Müller [2009] efficiently stores precomputed, continuously updated data, generated by the checking module for a restricted set of wire types. For each track coordinate t and some wire type elements e of these wire types it maintains the information whether e and a point-shaped stick figure placed at t induce a legal shape or not. This data then is stored efficiently as intervals of track coordinates where this information is equal.

2.4.2 Shape Data Structures

For processing range queries in BonnRoute we currently use the following data structure called *shape grid* (Müller [2009]). The following section summarizes the basic structure of the shape grid.

The Shape Grid

For storing a set of shapes S, the shape grid partitions the chip area on each plane into shapes, called *cells*, such that each of them has at most one neighboring cell on each of its four sides. For a cell c the set of shapes resulting from intersecting each shape in S with c builds the *cell configuration* of c. The shapes of a cell configuration are stored with coordinates relative to its center as anchor point. Since typically many cells have identical cell configurations, only a *cell configuration number* is stored in each cell, identifying the actual cell configuration, which is stored in a lookup table. When adding a shape to the shape grid, a balanced search tree is used (e.g. Adelson-Velskii and Landis [1962]) in order to determine if an existing cell configuration can be reused or a new one has to be created. Because it is common that several neighboring cells have the same cell configuration, the shape grid only stores intervals of equal cell configuration numbers of

horizontally or vertically neighboring cells. Each row or column of cell configurations is stored by again using a balanced search tree. Figure 2.15 shows an example.

1	2	3	4	5	6	7	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	8	8	8	8	8	8	8	8
0	0	9	10	11	12	12	12	12	12	12	12
0	0	0	0	0	0	0	0	0	1	2	3

Figure 2.15: Cells of the shape grid with their cell configuration numbers. While having five times seven cells, there are only fifteen intervals of equal neighboring cells. Intervals with empty cell configurations (number zero) are not stored explicitly. (Figure adapted from Müller [2009] and Gester et al. [2012].)

Finally at each interval a mapping of cell configuration shapes to corresponding nets is maintained. This is needed in the rip-up and reroute approach in BonnRoute for evaluating a cost function on the nets of shapes that need to be removed to make room for some new wire shape. By respecting this cost function in rip-up path searches, one can for example make it more expensive to rip-up nets having a high user defined priority such that these are less likely to be rerouted with detours.

The shape grid, however, gradually has shown some weaknesses and room for improvement. First, since shapes are removed and added one by one, many intermediate cell configurations can occur which are not used anymore at the end. Identifying and deleting such unused cell configurations therefore is necessary from time to time in practice, which costs additional runtime. Second, despite of building intervals of same cell configurations, sometimes there are regular, strongly repeating structures in practice that are not represented memory efficiently in the shape grid. An example are the large, grid like patterns of vias occurring in power supply nets that can be aligned badly to cell boundaries leading to many short intervals. The shape grid also does not take advantage of the fact that there are many standard circuits at different places of the chip area consisting of the same shapes up to translation. Third, the result of a range query obtained from the shape grid usually consists of the intersections of the original shapes with several cells. For most routines using range queries it is inconvenient and unnecessary to work with this larger set of cut shapes. In particular this can be problematic in routines which necessarily depend on working on the original shapes. We therefore propose a new alternative data structure addressing all these problems in the following section.

The Shape Tree

In this section we present a new data structure for processing range queries, the *shape tree*, which is much simpler than the shape grid and better addresses the needs of the current BonnRoute implementation.

Most shapes in BonnRoute represent wires and therefore are induced by stick figures. For each net the wiring generated by BonnRoute is represented as a *normalized* set of stick figures, which means that stick figures only intersect at their endpoints and two intersecting stick figures running in the same direction are merged if possible.

Moreover, most stick figures are on-track, because they are generated by the on-track path search in BonnRoute. We have only a few percent of off-track stick figures, which are needed for accessing pins. By subdividing the chip area appropriately, we therefore can efficiently store stick figures using only one-dimensional search trees. Also the blockage and pin shapes in BonnRoute can be stored in this structure. Important for efficiency, however, is to avoid explicitly representing all shapes of repeating patterns, as we will describe later in this section.

The shape tree data structure works as follows: W.l.o.g. let p be a wiring plane with horizontal preferred direction, the vertical case is analogous. We determine coordinates $y_0 < \ldots < y_n \in \mathbb{Z}$ with $y_0 = y_{min}, y_n = y_{max}$ and partition the chip area $\mathcal{A}_p$ into disjoint stripes

$$\begin{aligned} A_p^i &:= [x_{min}, x_{max}] \times [y_{i-1}, y_i[\times \{p\} \text{ for } i \in \{1, \ldots, n-1\}, \text{ and} \\ A_p^n &:= [x_{min}, x_{max}] \times [y_{n-1}, y_n] \times \{p\}. \end{aligned}$$

For efficiency, the height of the stripes should be chosen small enough, e.g. containing at most one track. Let S be a set consisting of stick figures intersecting p and shapes on plane p to be stored in the shape tree. For each stripe $A_p^i, i \in \{1, \ldots, n\}$ we maintain a balanced search tree T_p^i, which contains elements $e = (x(e), S(e)) \in \mathbb{Z} \times 2^{S(A_p^i)}$ sorted by $x(e)$ (as key), where $S(A_p^i) := \{s \in S : A(s) \cap A_p^i \neq \emptyset\}$. Note that the tree may contain different elements with equal keys.

At all times we maintain the following invariant for $i \in \{1, \ldots, n\}$:

$$\begin{aligned} T_p^i &:= \{e_s : s \in S(A_p^i)\}, \text{ where} \\ x(e_s) &:= x_1(s) \\ S(e_s) &:= \{s\} \cup \{s' \in S(A_p^i) : x_1(s') < x(e_s) < x_2(s')\} \end{aligned} \tag{2.16}$$

Let s be a shape on plane p or a stick figure intersecting p. We realize adding s to (or removing s from) the shape tree by performing the following operations for each $i \in \{1, \ldots, n\}$ where A_p^i is intersected by $A(s)$:

- `Add(`s, i`)`: First, we set $S(e) := S(e) \cup \{s\}$ for all $e \in T_p^i$ with $x_1(s) < x(e) < x_2(s)$. Second, we check if there is an $e \in T_p^i$ with $x(e) = x_1(s)$ and $s \in S(e)$. If not, we create such an e with $x(e) := x_1(s)$,

$$S(e) := \left\{s\right\} \cup \left\{s' \in S(A_p^i) : x_1(s') < x(e) < x_2(s')\right\},$$

and set $T_p^i := T_p^i \cup \{e\}$.

- `Remove(`s, i`)`: We set $S(e) := S(e) \setminus \{s\}$ for all $e \in T_p^i$ with $x_1(s) \leq x(e) \leq x_2(s)$, and remove e from T_p^i if $\{s' \in S(e) : x_1(s') = x(e)\} = \emptyset$.

It is easy to see that the operations `Add` and `Remove` maintain invariant (2.16). Given a shape q on plane p, we realize a range query by performing the following operation for each $i \in \{1, \ldots, n\}$ where A_p^i is intersected by $A(q)$:

- `RangeQuery(`q, i`)`: We determine $l := \max\{x(e) : e \in T_p^i, x(e) < x_1(q)\}$ and return all $s \in S_q(T_p^i) := \{s \in S(e) : e \in T_p^i, l \leq x(e) \leq x_2(q)\}$ with $A(s) \cap A(q) \neq \emptyset$. The existence of l can be guaranteed by adding a dummy element $e_{min} = (x_{min} - 1, \emptyset)$ to all T_p^i.

This procedure works as intended:

Proposition 2.49. *Let S be a set consisting of stick figures and shapes stored in the trees $T_p^i, i \in \{1, \ldots, n\}$ as described above such that invariant (2.16) holds, and let q be a shape on plane p. Then calling the above procedure* `RangeQuery(`q, i`)` *for each $i \in \{1, \ldots, n\}$ with $A_p^i \cap A(q) \neq \emptyset$ correctly answers the range query with respect to q, i.e. we have*

$$S_q := \{s \in S : A(s) \cap A(q) \neq \emptyset\} \subseteq \bigcup_{i \in \{1,\ldots,n\}} S_q(T_p^i).$$

Proof. Assume there is a $s \in S_q \setminus \bigcup_{i \in \{1,\ldots,n\}} S_q(T_p^i)$. By construction there is a $A_p^j, j \in \{1, \ldots, n\}$ with $A_p^j \cap A(s) \neq \emptyset$. Then by the invariant, T_p^j contains an element e_s with $x(e_s) = x_1(s)$ and $s \in S(e_s)$. Since $s \in S_q$ we must have $x_1(s) \leq x_2(q)$ and $x_2(s) \geq x_1(q)$. If also $x_1(s) \geq x_1(q)$, then we clearly have $s \in S_q(T_p^j)$, which contradicts our assumption. Otherwise, suppose $x_1(s) < x_1(q)$ and s is not contained in $S_q(T_p^j)$. Then we must have $x_1(s) < l := \max\{x(e) : e \in T_p^j, x(e) < x_1(q)\}$. But because of $x_1(s) < l < x_1(q) \leq x_2(s)$ and the invariant, this implies that there is a $e^* \in T_p^j$ with $x(e^*) = l$ and $s \in S(e^*) \subseteq S_q(T_p^j)$, which again contradicts our assumption. □

Regarding memory and runtime of the shape tree data structure, note that for storing a set S we have in the worst case $\left|T_p^i\right| = |S|$ and $\sum_{e \in T_p^i} |S(e)| \in \mathcal{O}(|S|^2)$ for each $i \in \{1, \ldots, n\}$. In the common situations occurring in practice, as described in the beginning of this section, this looks much better. In the standard case where each stripe contains at most one track, and S only contains normalized on-track stick figures running in preferred direction, we have $\sum_{i \in \{1,\ldots n\}} \left|T_p^i\right| = |S|$ and $|S(e)| = 1$ for all $e \in T_p^i, i \in \{1, \ldots, n\}$.

Remark. Note that since the shape tree data structure stores wires only in their stick figure representation, some additional work has to be done if one wants to obtain the wire *shapes* intersecting a query shape q. We then have to determine an appropriate query shape q' covering a larger area $A(q') \supseteq A(q)$ such that for each shape s induced by a stick figure (l, W) we have $A(s) \cap A(q) \neq \emptyset \implies l \cap A(q') \neq \emptyset$. The area $A(q')$ of course depends

on q and the overhang shapes of wire type elements on plane $p(q)$. If it becomes much larger than $A(q)$, runtime of such range queries can increase. In practice, however, it is only insignificantly larger than $A(q)$ because the width of most overhang shapes is small.

Regarding pin and blockage shapes, we also cannot have many of them covering the same x-coordinate within one stripe because of minimum distance rules. Moreover, we build an overlap free representation of these in the initialization part of BonnRoute, stick figures are disjoint from blockage shapes, and typically abut with, or are contained in pin shapes in the case of pin access. We discuss how to deal with repeating, regular patterns of such shapes efficiently in the next section.

Overall, by choosing an appropriate balanced search tree, e.g. as in (Adelson-Velskii and Landis [1962]) or (Bentley [1979]), the shape tree data structure can be implemented efficiently. An example of the shape tree is shown in figure 2.16.

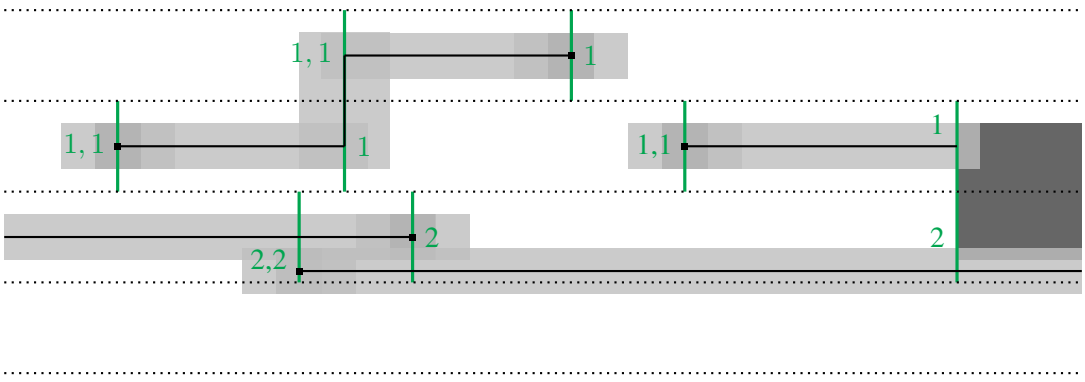


Figure 2.16: Example of the shape tree data structure showing wire stick figures (black lines), via stick figures (black dots) and a pin shape (dark gray) in an area divided into four stripes. The vertical green line segments within the stripes depict the x-coordinates $x(e)$ of tree elements e of the tree corresponding to the respective stripe. Each number next to such a line segment denotes $|S(e)|$ for a tree element e at that x-coordinate. Note that for some x-coordinates we have multiple tree elements. While the top two stripes contain standard situations, which occur dozens of times in practice, the two bottom stripes contain shapes causing a minimum distance violation, which generally should not happen.

Regular Shape Patterns

In practice there often are sets of shapes S which consist of placing a certain pattern of shapes at multiple different locations. Instead of storing all shapes in S explicitly we compute a much smaller set S' approximating S in the sense that $A(S) \subseteq A(S')$ and only store this reduced set of shapes in the shape tree together with a reference to the pattern. If the query area of a range query intersects an element of S', then only the shapes of the corresponding pattern which intersect this area are instantiated on the fly. A simple example of such patterns are *power vias*, i.e. vias of power supply nets, aligned in a grid like fashion as follows: A *power via pattern* v consists of a shape s, counts $n_x, n_y \in \mathbb{N}$, and distances $d_x, d_y \in \mathbb{N}$. The corresponding set of power via shapes on p is:

$$S(v) := \{(id_x, jd_y, p) + s : \ i \in \{1, \ldots, n_x\}, \ j \in \{1, \ldots, n_y\}\}$$

Note that in practice n_x or n_y can be very large, e.g. $\geq 10^5$ on current large designs. If d_x, d_y are small enough, we simply can choose the approximating set $S'(v)$ as the shapes with the smallest area containing a row or column of power vias, respectively. Having stored only $S'(v)$ and references from each $s \in S'(v)$ to v, for a given query shape q the set of shapes $S(v)_q = \{s \in S(v) : A(s) \cap A(q) \neq \emptyset\}$ can be found in $\mathcal{O}(|S'(v)_q| + |S(v)_q|)$ time, where $S'(v)_q := \{s' \in S'(v) : A(s) \cap A(q) \neq \emptyset\}$, plus the time of the range query to obtain $S'(v)_q$.

Of course if S' contains few shapes, but approximates S only roughly, we have few tree elements, but often check if the pattern intersects a query area in vain. Conversely, if S' contains many shapes to be more accurate, we may not reduce the number of tree elements significantly.

A second example of repeating patterns are the multiple occurrences of circuits having the same internal structure. Most types of standard circuits occur several times on a design, and have the same shapes up to translation and certain types of rotation and mirroring. Instead of explicitly storing the shapes of each circuit in the shape tree, we only store its *circuit area*, a given set of shapes whose area covers all shapes of the circuit, together with a reference. To retrieve the shapes within the circuit area that actually intersect a query area efficiently, one might need a data structure like a quadtree (Finkel and Bentley [1974]), which can be build once in advance for each type of circuit. This approach naturally can be extended to more general patterns as long as they can be identified and approximated efficiently.

Using this relatively simple data structure instead of the shape grid also makes ignoring certain shapes of a net while it is being routed much easier. Attached to each shape one can store a *net identifier* such that ignoring every shape of a given net is possible in constant time, which is not the case for the shape grid. Ignoring a specific set of shapes could be achieved by storing this set in a hash table, and not adding any shape contained in that table to any range query result.

2.4.3 The Checking Module

The task of the checking module of BonnRoute is to decide if a given shape s on plane p is legal with respect to the set of shape class minimum distance rules D_p^{sc} and all other shapes S on p in the current routing situation. Moreover, it has to return an interval in which this property does not change. To determine the area in which shapes in S can cause a violation with s, it computes a set of shapes Q such that $A(Q) \supseteq A(s,r)$ for all $A(s,r)$ in

$$\left\{A(s,r) : r = (c,c',t,d) \in D_p^{sc}, c(s) \in \{c,c'\} \subseteq C_p\right\},$$

where C_p denotes the set of shape classes on p. After determining the set of shapes $S_Q := \{s' \in S : A(s') \cap A(Q) \neq \emptyset\}$ by using one of the data structures described in section 2.4.2, it checks for each $s' \in S_Q$ if s, s' violate any rule $r \in D_p^{sc}$.

Checking the different kinds of shape class minimum distance rules for violations according to definition 2.31 is straight forward, and can be done in constant time. The implementation of euclidean distance checks in practice can be accelerated by precomputing the function $\delta_d : \{0,\dots,d\} \to \{0,\dots,d\}$, $\delta_d(x) := \left\lceil \sqrt{d^2 - x^2} \right\rceil$ for each distance $d \in \mathbb{N}$ required by any shape class minimum distance rule $(c,c',\mathrm{eucl},d) \in D^{sc}$. Then it clearly holds for any $q = (x,y), q' = (x',y') \in \mathbb{Z}^2$ that $\mathrm{dist}(q,q') < d \Leftrightarrow |x - x'| < d \wedge |y - y'| < \delta_d(|x - x'|)$.

If we use the shape tree data structure presented in 2.4.2, it would be also easily possible to realize the shape modifications of generalized shape class minimum distance rules proposed in section 2.3.8. Note that with this data structure we directly obtain the original shapes (instead of cut pieces as in the shape grid), which can be necessary to apply such modifications efficiently.

2.5 Handling Same Net Rules

In the previous sections we mainly considered design rules involving shapes of *different* nets. A further difficult task is satisfying the various kinds of same net rules, i.e. design rules that apply to shapes belonging the *same* net. In BonnRoute we try to avoid same net errors by using a same net rule aware pin access algorithm, and by restricting ourselves to on-track routing for longer distances. In combination with several postprocessing methods and an external DRC error fixing step, this yields sufficiently low same net error counts in practice. In this section we describe the main aspects of this approach.

2.5.1 Pin Access

In older technologies with 90 nm feature sizes and above all routing shapes had to be aligned to a given, layer dependent grid, where typically the distance between each two adjacent nodes (grid pitch) was the sum of the minimum width and minimum spacing required by the technology design rules. Each pin was guaranteed to cover and align to a

set of on-grid points. Typically simply accessing each pin at any of these points also did avoid same net rule violations. Therefore, clean and efficient pin access was relatively easy. In BonnRoute there was a single path search algorithm for covering long distances and accessing pins, both of course on the predefined grid.

Starting with the 65 nm technology, all shapes can be placed on a much finer grid with a pitch of only a few nanometers. For most connections it still is feasible, and good for efficient packing of wires, to use equidistant tracks similar to the predefined grid in older technologies. Pin access, however, has become much more difficult because densely packed pin shapes may require to actually use the finer grid. Off-track wires generally are necessary for pin access, but can easily cause dozens of violations of same net rules, which have also become much more restrictive. In addition the much denser pin configurations require a pin access approach which also avoids creating access paths blocking other nearby pins.

With all these problems that have to be taken into account, pin access has become a very difficult part of detailed routing. In BonnRoute we have developed specialized algorithms and data structures for pin access, keeping this complexity out of our on-track path search algorithm, and therefore maintaining its efficiency. A special path search algorithm supporting off-track routing computes for each pin a set of paths connecting the pin to near on-grid points. Then the on-track path search algorithm of BonnRoute only accesses these on-grid points instead of the pin itself. By concatenating the resulting *on-track path* with these *pin access paths* we obtain a complete connection.

Access Areas of Pins

The first step in our pin access approach is to determine eligible *access areas* for each pin. An access area of a pin is basically an area that can be used for access without creating same net errors. In this section we define access areas formally, and describe how to compute them efficiently. Access areas are part of the input of the path search algorithm of BonnRoute which computes pin access paths.

The intention behind access areas is to get rid of the parts of the pin's area where a via or a wire stick figure (regardless of its length) cannot access without causing some kind of same net rule violation. Violations, however, depending on the length of a wire stick figure, we do not want to remove. These are avoided directly in the pin access path search algorithm itself. Some examples of same net errors caused by stick figures accessing a pin are shown in figure 2.17.

In the following we formally define access areas based on the example of minimum edge rules. Of course, in practice there are many other same net rules that have to be taken into account, but these can be handled similarly. Let $\mathcal{P}$ be a pin with a set $S_{\mathcal{P}}$ of shapes contained in plane p and building a rectilinear polygon $\mathrm{plg}(S_{\mathcal{P}})$. W.l.o.g. assume that p has horizontal preferred direction, and there is a unique minimum edge rule r_{edge} specified by the technology design rules on p. We first need to define what accessing $\mathcal{P}$ actually means:

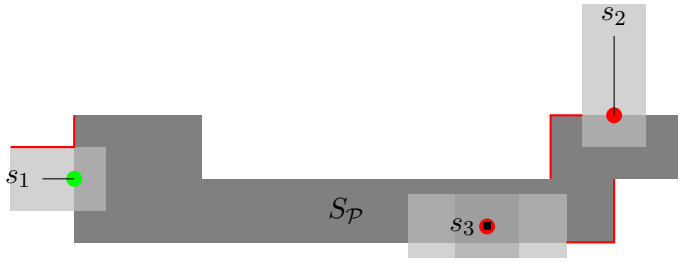


Figure 2.17: A pin $\mathcal{P}$ with a set $S_\mathcal{P}$ of shapes (dark gray), two wire stick figures s_1, s_2, and one via stick figure s_3 accessing $\mathcal{P}$, each creating a minimum edge rule violation. The violations caused by s_2 and s_3 we want to avoid by computing access areas which do not contain the corresponding access points (red dots). The violation caused by s_1, however, depends on the length of $l(s_1)$, therefore the green access point will remain in our access areas unless other kinds of errors can occur.

Definition 2.50. *Let $\mathcal{P}$ be a pin with a set $S_\mathcal{P}$ of shapes on plane p, and s a stick figure with $p \in \{p_1(s), p_2(s)\}$. If s is a via stick figure and $A(s) \cap A(S_\mathcal{P}) = \{a\}$, we say that s* accesses *$\mathcal{P}$ at a. Otherwise, if s is a wire stick figure and there exist*

$$(x, y) \in \{(x_1(s), y_1(s)), (x_2(s), y_2(s))\} \cap A(S_\mathcal{P}),$$
$$(x', y') \in \{(x_1(s), y_1(s)), (x_2(s), y_2(s))\} \setminus A(S_\mathcal{P}),$$

we say that s accesses *$\mathcal{P}$* at (x, y) in direction $d \in \{\text{north}, \text{east}, \text{south}, \text{west}\}$ *where*

$$d := \begin{cases} \text{north} & \text{if } y < y' \wedge x = x' \\ \text{east} & \text{if } x < x' \wedge y = y' \\ \text{south} & \text{if } y > y' \wedge x = x' \\ \text{west} & \text{if } x > x' \wedge y = y'. \end{cases}$$

In figure 2.17 the via stick figure s_3 accesses p, the wire stick figure s_1 accesses $\mathcal{P}$ in direction west, and s_2 accesses $\mathcal{P}$ in direction north.

While for via stick figures the accessed point of $S_\mathcal{P}$ is sufficient to determine the violations caused by the induced shapes, for wire stick figures we additionally need to consider the direction in which $\mathcal{P}$ is accessed. Regarding the length of wire stick figures we always assume it to be sufficiently large such that the thereby influenced edges are long enough to satisfy r_{edge}.

More precisely, we will construct the following access areas. As defined in section 2.3.1 let $w = (p, t, o)$ be a wire type element with shape type t and overhang shape o. If $t \in \{\text{bot}, \text{top}\}$, the wire type element is used for vias, and we define the access area of $\mathcal{P}$ with respect to w as

$$A_w(S_\mathcal{P}) := \{(x, y) \in A(S_\mathcal{P}) \cap \mathbb{Z} : \text{plg}(S_\mathcal{P} \cup \{(x, y, p) + o\}) \text{ satisfies } r_{edge}\}.$$

Similarly, if $t = \text{pref}$, it is used for wires in preferred direction, and we define for each $d \in \{\text{east}, \text{west}\}$ the access area

$$A_{w,d}(S_\mathcal{P}) := \{(x,y) \in A(S_\mathcal{P}) \cap \mathbb{Z} : \exists l \in \mathbb{N} : \text{plg}(S_\mathcal{P} \cup \{s_{x,y,d,l} + o\}) \text{ satisfies } r_{edge}\},$$

where $s_{x,y,d,l}$ is the line segment connecting the point (x,y,p) with $(x+l,y,p)$ if $d = \text{east}$, or with $(x-l,y,p)$ if $d = \text{west}$.

Analogously we define $A_{w,d}(S_\mathcal{P})$ for $t = \text{jog}$ and $d \in \{\text{north}, \text{south}\}$. We then have the property that any shape induced by w and a via stick figure accessing $\mathcal{P}$ at $a \in A(S_\mathcal{P})$ does not cause a violation of r_{edge} if and only if $a \in A_w(S_\mathcal{P})$. Similarly any shape induced by w and a wire stick figure of sufficient length accessing $\mathcal{P}$ in direction d at $a \in A(S_\mathcal{P})$ does not cause a violation of r_{edge} if and only if $a \in A_{w,d}(S_\mathcal{P})$.

Let us now describe the basic idea how we compute such access areas. Consider the via case, i.e. we have a wire type element $w = (p,t,o)$ with shape type $t \in \{\text{bot}, \text{top}\}$. We inspect each edge $e \in \text{plg}(S_\mathcal{P})$ and derive, using o, a set of shapes C_e with the property that for each $(x,y) \in A(C_e) \cap \mathbb{Z}$ we have that $(x,y,p) + o$ intersects e and causes a violation of r_{edge} by the corresponding edges in $\text{plg}(S_\mathcal{P} \cup \{(x,y,p) + o\})$. We then compute $A(S_\mathcal{P}) \setminus A(\bigcup_{e \in \text{plg}(S_\mathcal{P})} C_e)$ to obtain the desired access area $A_w(S_\mathcal{P})$. Figure 2.18 (a) shows an example of vertices of the polygon build by the shapes of a pin and the induced shape of a via stick figure accessing it.

Considering an edge e of this polygon intersected by this via shape we can construct the set C_e by adding an appropriate shape for some specific vertices of this polygon. For vertex v in figure 2.18 (a) for example we add the following shape c_e^v to C_e:

$$\begin{aligned}
x_1(c_e^v) &:= x_1(e) + |x_1(o)| + 1 \\
x_2(c_e^v) &:= \begin{cases} x_1(e) + l_2 - 1 & \text{if } \text{length}(e_1) < l_1 \\ x_1(e) + l_1 - 1 & \text{if } l_1 \leq \text{length}(e_1) < l_2 \\ x_1(c_e^v) - 1 & \text{otherwise} \end{cases} \\
y_1(c_e^v) &:= y_1(e) - y_2(o) + 1 \\
y_2(c_e^v) &:= y_2(e) + y_1(o)
\end{aligned}$$

Note that in the third case of setting $x_1(c_e^v)$ we have $A(c_e^v) = \emptyset$, meaning that no point of $A(S_\mathcal{P})$ needs to be removed with respect to v and e. It can be easily verified that $A(c_e^v) \cap \mathbb{Z}$ is the set of points $(x,y) \in A(S_\mathcal{P})$ where v is a vertex in $\text{plg}(S_\mathcal{P} \cup \{(x,y,p) + o\})$ with incident edges e_1 and $e' \subseteq e$ with $\text{length}(e_1) < l_1 \implies \text{length}(e') \geq l_2$. Figure 2.18 (b) shows the complete set of shapes C_e.

Note that figure 2.18 only shows a standard case. There might be other edges of $\text{plg}(S_\mathcal{P})$ in addition to e which intersect the wire/via shape, and therefore also need to be considered. Moreover in practice in addition to minimum edge rules there are several other same net rules to be considered when computing access areas. For each of these rules we compute access areas one by one, and build their intersection to avoid violations

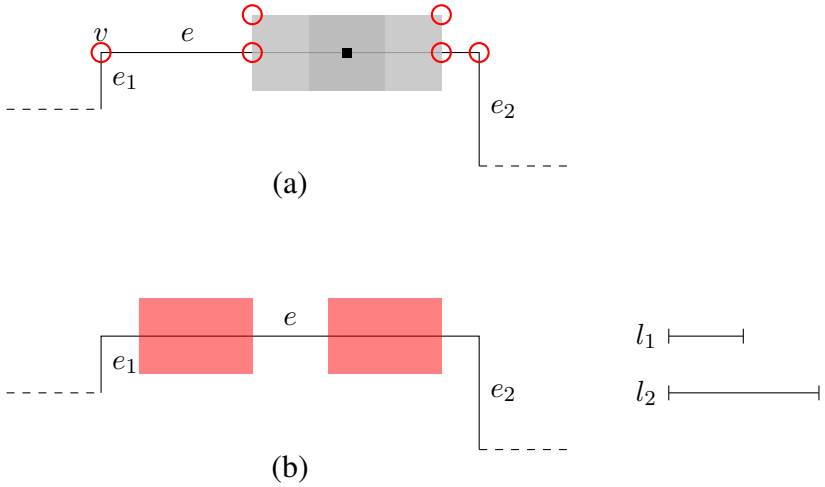


Figure 2.18: Illustration of computing the set of shapes C_e for an edge $e \in \mathrm{plg}(S_\mathcal{P})_{\mathrm{north}}$ for some pin $\mathcal{P}$ with shapes $S_\mathcal{P}$. Vertices like the ones indicated by the red circles in (a) have to be considered when constructing the shapes of C_e, which are shown in red in (b). In these red areas via stick figures inducing shapes like the gray shape shown in (a) create a violation of the minimum edge rule $r_{edge} = (l_1, l_2)$.

of all of these rules simultaneously. The case where we are given a direction and a wire type element used for wires is handled similarly.

We do not compute access areas each time we want to access a pin. Instead we build classes of pins having the same shapes up to translation and some types of rotation and mirroring. For each representative pin of such a class we construct and store access areas for each wire type element (and direction if required) plane by plane. Since we only have a few thousand of such classes even on largest designs with tens of millions of pins, this precomputation is very fast in practice. To illustrate the result, figure 2.19 shows three access areas of a pin of a real-world 22 nm design taking several different same net rules into account, including minimum edge rules. This figure nicely demonstrates the strong pin access restrictions imposed by the design rules of current technologies.

Computing Pin Access Paths

Having determined access areas of pins, we need to find paths connecting these to near on-grid points, which then can be used as source and target for the on-track path search of BonnRoute. To obtain such *pin access paths*, off-track routing may be required because access areas of pins do not necessarily intersect an on-grid point or track. This and the fact that pin access paths only have small distances to cover can lead to short path segments, which can easily violate same net rules. To avoid this, we search for shortest paths under minimum segment length restrictions, which is done in BonnRoute as described

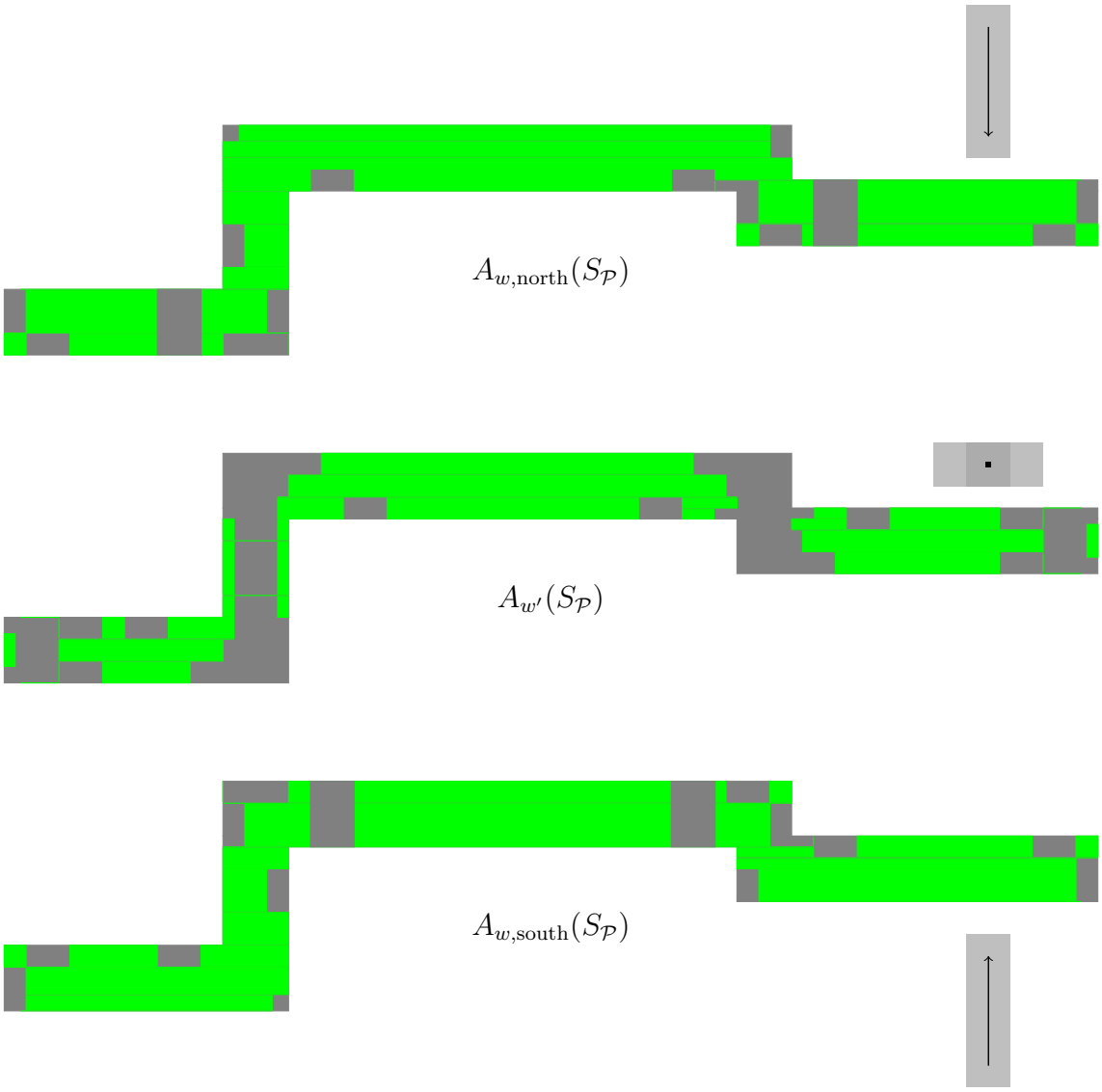


Figure 2.19: Example of access areas (green) of a pin $\mathcal{P}$ with shapes $S_\mathcal{P}$ (dark gray) of a real-world 22 nm instance for different wire type elements and directions. The northern and southern figures show the access areas for a wire type element w defining the overhang shape of vertical wires for direction $\text{north}, \text{south}$, respectively. The middle figure shows the access area for a wire type element w' defining the overhang shape used for via bottom shapes. Example shapes induced by these wire type elements and some stick figure are depicted next to the access areas. Each of these can be placed with the stick figure inside the corresponding access area without creating a violation of same net rules.

by Maßberg and Nieberg [2012]: Given obstacles represented by n shapes, one can solve this problem by a modified version of Dijkstra's algorithm working on an extended Hanan grid in $O(n^4 \log n)$ time.

In order to save runtime in practice, we do not compute pin access paths each time we want to access a pin. Note that pins are parts of circuits, and large designs can contain millions of these. All circuits, however, are placed instances of a much smaller number of circuit definitions called books. This means that equal configurations of pin and blockage shapes typically occur many times at different locations on each design.

Therefore we build classes of circuits with the property that all circuits in such a *circuit class* contain equal pin and blockage shapes within some area up to translation and some kinds of rotation and mirroring (Schulte and Nieberg [2008]). By also ensuring that the relative positions of on-grid points within these areas are equal for all members of a circuit class, it is sufficient to precompute and store pin access paths only for one representative circuit of each class. The access paths for any other pin then can be obtained simply by inspecting its corresponding representative circuit and translating the paths computed for the corresponding pin appropriately.

In addition it is important to avoid conflicts between pin access paths of near pins. In dense pin configurations, which occur frequently on designs in modern technologies, it can easily happen that a pin access path of one pin blocks another pin belonging to the same circuit. In BonnRoute this is also taken into account by actually computing a *conflict free set* of access paths for each circuit class. This basically results in solving a COLORED INDEPENDENT SET PROBLEM (Maßberg and Nieberg [2009]), which is done in BonnRoute by a branch and bound method. See (Nieberg [2011]) for more details.

2.5.2 Postprocessing

Despite of the same net error avoiding pin access approach described in section 2.5.1 and our restriction to on-track routing for longer distances, without further measures same net errors still occur in significant numbers in practice.

There are two main reasons for this: First, even if a pin access path and an on-track path are free of same net errors, this does not necessarily hold for the concatenation of these. In fact on-track paths may be needed which access the on-grid endpoint of a pin access path from a specific direction in order to remain same net error free. Computing such directions is easy, guiding the on-track path search to do this, however, is not trivial. This currently is not realized in BonnRoute, but would be a worthwhile improvement for future versions.

Second, on-track routing satisfies many same net rules by construction because most edges of the resulting rectilinear polygons of shapes generally have sufficient length, but combinations of different kinds of shapes still can cause violations. For example, intersecting jog and via shapes, as noted already in section 2.3.4, often lead to end edges requiring more space, and in addition typically create minimum edge rule violations. Another example where this can happen are shapes induced by stick figures with different

wire types.

Such errors are not easy to avoid. Generally considering these directly within the on-track path search, e.g. by a multi-labeling approach, may cost too much runtime. Always using larger vias/wires which are wider than minimum width or require larger minimum distances is not an option because of high additional routing space usage. We can, however, apply such measures selectively in a postprocessing step where needed.

In BonnRoute we have two postprocessing phases where several routines try to apply local modifications to the wiring in order to fix same net rule violations. In the first phase each individual path is postprocessed directly after it has been created. Especially modifications that need additional routing space are applied in this step, as later the wiring of other nets may block this space. Examples of such modifications are changing the wire type of certain stick figures or changing their geometry slightly. The second phase is done after all nets have been routed. Here same net errors not only involving a single path are addressed, and the whole wiring of each net is taken into account.

Note, however, that it is a tedious task to avoid or fix *all* kinds of same net rule violations, therefore we currently only handle the most important ones in BonnRoute. To clean up remaining violations in practice, BonnRoute is combined with an industrial standard router with strong local DRC error fixing capabilities. This combination gives very good results in practice, which we present in chapter 3.2.

3 BonnRoute in Practice

BonnRoute is the routing part of the BonnTools software package developed at the Research Institute for Discrete Mathematics at the University of Bonn in cooperation with IBM. We have given an overview over its main components in section 1.2. BonnRoute, formerly known as XRouter, is used in practice by IBM for over 20 years now. Over thousand different ASIC chips have been routed with BonnRoute, some of which had enormous sizes up to 11 million nets. Since 2011 BonnRoute is also able to route server chips. One key part to support this new environment and the complex design rules of 32 nm and 22 nm technologies was the BonnRouteRules module that we presented in section 2.3.

In this chapter we first give an overview of the combined routing flow in which BonnRoute is currently used at IBM. We then present detailed experimental results showing that this flow gives excellent results in practice on current real-world ASIC and server chips. The low number of remaining design rule violations in these results also confirms that our design rule model described in chapter 2 works well in practice.

3.1 Combined Routing Flow

In practice BonnRoute is used together with another routing tool, an industrial standard router (*ISR*) that originally has been used without BonnRoute at IBM, mainly for server chip routing. Using a track assignment and switch-box routing step (see section 1.1), it is fundamentally different from BonnRoute. Generally, one can say that while the strength of BonnRoute lies in the fast and efficient packing of wires without many detours, ISR is rather focused on obtaining an exceptionally clean result in terms of remaining design rule violations. ISR also is far more modularized than BonnRoute, meaning that many individual functions can easily be controlled from outside. Since these properties of the two routing tools complement one another, first experiments using a combined flow with both tools started several years ago in close cooperation with IBM.

This flow basically works as follows: First, BonnRoute routes all nets except a small set of special nets that in the current BonnRoute implementation are not supported. The resulting wiring generally satisfies almost all distance rules that apply to shapes of different nets. Exceptions are for example some cases of line end minimum distance rule violations that can occur due to our optimistic model as discussed in section 2.3.4. In addition BonnRoute avoids most same net rule violations by the measures we described in section 2.5. The second step of the combined routing flow then is to use ISR to realize

the remaining connections and fix design rule violations by a sequence of clean up steps. This basically involves local rip-up and reroute of wires that violate minimum distance rules, and several local postprocessing steps to fix same net errors.

Important for the overall success of this flow is that BonnRoute especially avoids errors that are difficult to fix later. These basically are errors where the fixing requires a significant amount of additional routing space, which might not exist anymore in the second step of the flow. Otherwise ISR struggles to fix such errors and needs a huge amount of runtime such that the combined flow has no runtime benefit anymore. Identifying the parts of the BonnRoute result that need to be improved to achieve better runtime of the overall flow is not easy and still under development.

3.2 Experimental Results

In this section we present experimental results of BonnRoute and the combined routing flow on several real world chips. Let us first describe the different criteria that we consider for evaluating the routing results.

- One traditional important criterion of course is the total *wiring length* over all nets. Shorter connections may give better results in terms of signal delay and power consumption and leave more space for subsequent physical design steps. Note that there are several postoptimization steps after routing in order to address timing issues, that may need additional space. For example the insertion of additional buffer circuits to amplify signals that have to cover long distances.

- From a timing point of view already one single net which is routed with a large detour may cause problems. Therefore in addition to total wiring length we consider the amount of nets with large detours. We call a net *scenic* if its total wiring length is at least 100 μm and at least 25% (or 50%) larger than the length of a Steiner tree having (approximatively) minimum length. For nets up to nine terminals, minimum Steiner trees can be obtained for example as proposed by Chu and Wong [2008]. For larger nets heuristic Steiner tree algorithms are used.

- Especially for production yield a low *number of vias* is important, because vias have a relatively high error probability in the manufacturing process. It is even common practice to add so called redundant vias in a postprocessing step to achieve better via robustness.

- A necessary condition for a routing result to be usable in practice is a sufficiently low number of *design rule violations* (DRC errors). Any such remaining error basically has to be fixed manually, which can be very difficult and time consuming.

- Finally, *runtime* is an important criterion. Routing typically is not just done once, but has to be iterated together with other physical design steps to achieve a successful result. Since a main part of the runtime of the overall physical design process

is spent in routing, it is very important to make routing tools as runtime efficient as possible.

Our testbed is shown in table 3.1. It consists of eight 22 nm server chips and eight 32 nm ASICs. The instance sizes range from about one hundred thousand nets up to 1.6 million nets. On the largest instance A8 there are about four million connections needed to connect all pins of each net. We call such missing connections *opens*. Typically ASIC instances reach larger sizes compared to server instances. But even ASICs today generally do not reach the enormous sizes that occurred in former technologies. They are designed more hierarchically, meaning that the design is split into several large blocks which are considered separately.

Chip	Tech. (nm)	Image Size (mm × mm)	Wiring Planes	Nets	Opens
S1	22	0.80 × 0.24	7	116,257	253,462
S2	22	0.95 × 0.33	7	136,573	252,120
S3	22	0.26 × 1.11	9	155,092	288,162
S4	22	0.48 × 2.13	13	438,328	757,548
S5	22	0.96 × 1.06	9	466,157	816,773
S6	22	0.96 × 1.06	9	501,875	877,224
S7	22	0.96 × 0.89	7	527,465	1,042,216
S8	22	1.24 × 1.52	13	604,213	1,108,832
A1	32	0.64 × 0.64	8	215,272	583,135
A2	32	2.73 × 0.77	9	648,023	1,494,413
A3	32	2.37 × 1.56	9	909,922	2,249,512
A4	32	2.90 × 5.34	9	985,565	2,248,659
A5	32	3.05 × 1.17	9	989,834	2,398,713
A6	32	2.77 × 1.95	9	1,252,364	3,064,564
A7	32	5.63 × 1.41	9	1,283,905	2,950,513
A8	32	2.90 × 1.28	8	1,650,584	3,939,558
$\sum$				10,881,429	24,325,404

Table 3.1: Our testbed consisting of eight 22 nm server and eight 32 nm ASIC chips.

All of the results we show in the following were produced on a machine with 192 GB memory and two Intel Xeon X5690 CPUs, each having six cores running at 3.47 GHz. Both tools, BonnRoute and ISR, were run using 12 threads.

Table 3.2 shows the BonnRoute results on our testbed in terms of wiring length, number of vias, runtime, as well as the number of remaining opens and spacing errors. The runtime columns show the runtime of the initialization, the global routing part, the detailed routing part, and the total runtime of BonnRoute. One can see that detailed routing by far dominates total runtime and global routing is extremely fast, needing less than two

hours for all 16 chips. The detailed routing part runs in about 18 hours, and in total BonnRoute needs about 30 hours to close most of the over 24 million opens of the chips in our testbed. The largest amount of the initialization runtime, which amounts to a total of over four hours, is spent in the precomputation of pin access paths that we described in section 2.5.1, and in initializing the shape grid and fast grid data structures described in section 2.4.1. The total runtime is larger than the sum of initialization, global, and detailed routing runtime because it additionally contains the runtime of postprocessing steps, data I/O, and data conversion.

The relatively small number of remaining opens compared to the initial number of opens shows that BonnRoute connected almost all nets. Note that there are some special nets currently not supported in BonnRoute, whose missing connections are also counted as opens here. Also some spacing errors are created on each instance, but almost all of them occur at line ends and are locally fixable.

Chip	Wires	Vias	Runtime (hh:mm:ss)				Opens	Spacing
	(m)	($\times 10^3$)	Init	Global	Detailed	Total		Errors
S1	1.97	863	0:02:44	0:00:42	0:16:27	0:25:51	1,638	3,103
S2	2.54	881	0:03:33	0:00:58	0:07:20	0:18:26	147	879
S3	2.69	1,061	0:04:28	0:01:19	1:39:26	1:53:32	74	1,960
S4	12.47	3,198	0:08:58	0:05:05	0:19:49	0:53:21	3,231	2,032
S5	11.91	3,274	0:09:41	0:04:03	0:23:22	0:57:17	2,753	2,033
S6	13.31	3,543	0:11:43	0:04:16	0:29:08	1:05:49	2,902	1,904
S7	11.39	3,952	0:12:43	0:04:44	0:51:22	1:33:14	1,060	4,831
S8	14.91	4,163	0:14:43	0:04:43	1:19:14	2:06:36	1,548	3,944
A1	4.11	2,059	0:04:04	0:01:24	0:20:50	0:31:46	1,147	1,692
A2	22.33	5,674	0:15:36	0:06:14	1:12:12	1:48:57	3,736	2,596
A3	32.87	9,005	0:30:17	0:08:28	1:47:18	2:46:32	4,187	6,683
A4	59.06	9,476	0:30:26	0:16:19	1:55:19	3:03:56	6,162	4,673
A5	35.23	9,022	0:23:52	0:09:31	1:16:02	2:16:21	4,742	4,320
A6	46.27	12,054	0:36:51	0:11:32	2:27:26	3:45:45	5,568	7,866
A7	61.59	12,067	0:38:21	0:15:42	2:29:37	3:52:20	5,808	7,115
A8	49.54	15,233	0:23:37	0:12:34	1:39:11	2:47:51	3,536	6,897
$\sum$	382.19	95,525	4:31:37	1:47:34	18:34:03	30:07:34	48,239	62,528

Table 3.2: The results of BonnRoute in terms of runtime, opens, and spacing errors. The runtime columns show the runtime of the initialization, global, and detailed routing part of BonnRoute as well as the the total runtime including postprocessing, data I/O, and data conversion.

In table 3.3 we compare the results of the combined flow we described in section 3.1 to the results of a plain ISR run without BonnRoute.

Chip	Tool	Runtime (hh:mm:ss)		Wires	Vias	Scenic Nets		DRC
		BR	Total	(m)	($\times 10^3$)	20%	50%	Errors
S1	ISR		0:52:39	2.05	1,196	47	7	4
	BR+ISR	0:25:51	1:31:36	1.99	914	6	0	5
S2	ISR		1:07:33	2.6	1,135	163	38	95
	BR+ISR	0:18:26	1:01:28	2.55	907	21	2	0
S3	ISR		2:07:32	2.9	1,481	1,812	1,079	21
	BR+ISR	1:53:32	2:40:47	2.7	1,080	582	301	12
S4	ISR		7:29:14	13.85	4,120	15,128	10,880	634
	BR+ISR	0:53:21	3:04:40	12.5	3,238	1,785	619	244
S5	ISR		5:53:25	12.29	3,980	3,868	2,421	269
	BR+ISR	0:57:17	3:33:25	11.93	3,322	428	260	74
S6	ISR		9:40:39	14.19	4,330	6,314	3,665	53
	BR+ISR	1:05:49	5:08:18	13.33	3,589	335	45	78
S7	ISR		7:51:13	11.97	5,279	3,101	1,422	290
	BR+ISR	1:33:14	7:46:43	11.46	4,119	224	33	46
S8	ISR		15:27:13	15.94	5,276	4,832	2,185	77
	BR+ISR	2:06:36	13:44:13	14.97	4,224	176	21	78
A1	ISR		2:09:07	4.20	2,360	352	92	12
	BR+ISR	0:31:46	1:11:43	4.13	2,072	4	1	11
A2	ISR		4:26:10	24.01	6,616	6258	3593	23
	BR+ISR	1:48:57	3:32:44	22.45	5,825	409	184	10
A3	ISR		7:52:17	34.59	9,910	7850	3932	21
	BR+ISR	2:46:32	5:40:32	32.96	9,090	465	67	15
A4	ISR		9:22:09	60.24	10,481	14563	6371	63
	BR+ISR	3:03:56	6:07:33	59.19	9,570	3,225	764	38
A5	ISR		8:33:55	37.00	10,358	8340	4789	24
	BR+ISR	2:16:21	7:42:45	35.41	9,306	438	134	30
A6	ISR		9:39:12	48.96	13,530	11309	5997	32
	BR+ISR	3:45:45	7:44:05	46.47	12,211	96	5	17
A7	ISR		12:10:51	65.94	13,797	19892	10874	50
	BR+ISR	3:52:20	8:21:34	61.73	12,198	1,593	773	27
A8	ISR		8:28:07	52.43	17,209	13165	6430	49
	BR+ISR	2:47:51	6:15:03	49.66	15,335	62	5	14
$\sum$	ISR		113:11:16	403.17	111,058	116,994	63,775	1,717
	BR+ISR	30:07:34	85:07:09	383.41	97,000	9,849	3,214	699
			-24.80%	-4.90%	-12.66%	-91.58%	-94.96%	-59.29%

Table 3.3: Comparison of our combined routing flow BR+ISR and ISR alone. All runtimes measure the total runtime of the respective step.

For each chip we have two rows, where the first row contains the result of the ISR run and the second row the result of the combined flow (BR+ISR). One can clearly see that the combined flow is far superior to ISR alone. It has 24% less runtime, 5% less wiring length, 12% less vias, and over 90% less scenic nets. Both, the combined flow and ISR alone, manage to route most chips with very few remaining design rule violations.

Surprisingly, the BonnRoute runtime only amounts to about 35% of the total runtime of the combined flow, although BonnRoute left only few opens and spacing errors as seen in table 3.2. Furthermore, most of these errors as well as the same net errors left by BonnRoute are locally fixable. This shows that the combined flow probably still can be significantly improved in terms of runtime. One the one hand by reducing DRC error counts already in the BonnRoute result, and on the other hand by improving the error fixing steps within the ISR part of the combined flow.

But already with the current state of the combined flow the results are excellent and clearly demonstrate that this is a very good approach to route current 32 nm and 22 nm chips.

Bibliography

Adelson-Velskii, G. M. and Landis, E. M. [1962]. An algorithm for the organization of information. *Soviet Mathematics Doklady*, 3:1259–1263.

Batterywala, S., Shenoy, N., Nicholls, W., and Zhou, H. [2002]. Track assignment: a desirable intermediate step between global routing and detailed routing. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '02, pages 59–66.

Bentley, J. L. [1979]. Decomposable Searching Problems. *Information Processing Letters*, 8(5):244–251.

Bickford, J., Hibbeler, J., Bühler, M., Koehl, J., Müller, D., Peyer, S., and Schulte, C. [2006]. Yield improvement by local wiring redundancy. In *ISQED*, pages 473–478.

Brenner, U., Struzyna, M., and Vygen, J. [2008]. BonnPlace: Placement of leading-edge chips by advanced combinatorial algorithms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(9):1607–1620.

Chang, C. and Cong, J. [2001]. Pseudopin assignment with crosstalk noise control. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 20(5):598–611.

Chen, H., Cheng, C.-K., Kahng, A., Mandoiu, I., Wang, Q., and Yao, B. [2003]. The Y-architecture for on-chip interconnect: Analysis and methodology. In *Proc. DAC*, pages 13–19.

Chen, S.-Y. and Chang, Y.-W. [2010]. Native-conflict-aware wire perturbation for double patterning technology. In *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, pages 556–561.

Cho, M., Ban, Y., and Pan, D. [2008]. Double patterning technology friendly detailed routing. In *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on*, pages 506–511.

Cho, M., Mitra, J., and Pan, D. Z. [2009]. Manufacturability-aware routing. In Alpert, C. J., Mehta, D. P., and Sapatnekar, S. S., editors, *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications.

Chu, C. and Pan, M. [2009]. Clock Network Design: Basics. In Alpert, C. J., Mehta, D. P., and Sapatnekar, S. S., editors, *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications.

Chu, C. and Wong, Y.-C. [2008]. FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design. *IEEE Trans. on CAD of ICs and Systems*, 27:70–83.

Dijkstra, E. [1959]. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271.

Finkel, R. A. and Bentley, J. L. [1974]. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9.

Gester, M., Müller, D., Nieberg, T., Panten, C., Schulte, C., and Vygen, J. [2012]. Algorithms and data structures for fast and good VLSI routing. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 459–464.

Ghaida, R., Agarwal, K., Nassif, S., Yuan, X., Liebmann, L., and Gupta, P. [2011]. A framework for double patterning-enabled design. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 14–20.

Gupta, P. and Kahng, A. B. [2003]. Manufacturing-aware physical design. In *Proceedings of the 2003 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '03.

Hart, P. E., Nilsson, N. J., and Raphael, B. [1968]. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics SSC4*, 2:100–107.

Held, S. [2008]. *Timing closure in chip design*. PhD thesis, University of Bonn.

Hetzel, A. [1998]. A sequential detailed router for huge grid graphs. In *Proc. DATE*, pages 332–339.

Hitchcock, R. B. [1969]. Cellular wiring and the cellular modeling technique. In *Proceedings of the 6th annual Design Automation Conference*, DAC '69, pages 25–41.

Ho, T.-Y., Chang, C.-F., Cheang, Y.-W., and Chen, S.-J. [2005]. Multilevel full-chip routing for the X-based architecture. In *Proc. DAC*, pages 597–602.

Humpola, J. [2009]. Schneller Algorithmus für kürzeste Wege in irregulären Gittergraphen. *Diploma Thesis, University of Bonn*.

IEEE [1994]. IEEE standard VHDL language reference manual. *ANSI/IEEE Std 1076-1993*.

Kahng, A. B. [2003]. Research directions for coevolution of rules and routers. In *Proceedings of the 2003 International Symposium on Physical Design*, ISPD '03.

Korte, B., Rautenbach, D., and Vygen, J. [2007]. BonnTools: Mathematical innovation for layout and timing closure of systems on a chip. *Proc. of the IEEE*, 95:555–572.

Kramer, M. and van Leeuwen, J. [1984]. The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits. *Advances in computing research*, 2:129–146.

Lee, D. T., Yang, C. D., and Wong, C. K. [1996]. Rectilinear paths among rectilinear obstacles. *Discrete Applied Mathematics*, 70:185–215.

Lin, Y.-H. and Li, Y.-L. [2010]. Double patterning lithography aware gridless detailed routing with innovative conflict graph. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 398–403.

Maßberg, J. [2009]. *Facility Location and Clock Tree Synthesis*. PhD thesis, University of Bonn.

Maßberg, J. and Nieberg, T. [2009]. Colored independent sets. In *8 th Cologne-Twente Workshop on Graphs and Combinatorial Optimization CTW09*, page 35.

Maßberg, J. and Nieberg, T. [2012]. Rectilinear paths with minimum segment lengths. *Discrete Applied Mathematics, to appear*.

Müller, D. [2009]. *Fast Resource Sharing in VLSI Routing*. PhD thesis, University of Bonn.

Nieberg, T. [2011]. Gridless pin access in detailed routing. In *Proc. DAC*, pages 170–175.

Object Management Group [2012]. UML® resource page. `http://uml.org`.

Ousterhout, J. K. and Jones, K. [2009]. *Tcl and the Tk Toolkit*. Addison-Wesley, 2. edition.

Peyer, S. [2007]. *Shortest Paths and Steiner Trees in VLSI Routing*. PhD thesis, University of Bonn.

Peyer, S., Rautenbach, D., and Vygen, J. [2009]. A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing. *Journal of Discrete Algorithms*, 7:377–390.

Schellenberg, F. M. [2009]. Modeling and computational lithography. In Alpert, C. J., Mehta, D. P., and Sapatnekar, S. S., editors, *Handbook of Algorithms for Physical Design Automation*. Auerbach Publications.

Schulte, C. [2006]. Yield-optimierung im Detailed Routing. *Diploma Thesis, University of Bonn*.

Schulte, C. and Nieberg, T. [2008]. Classbased detailed routing in VLSI design. In *Proceedings of the 7th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, pages 22–25.

Silicon Integration Initiative [2007]. LEF/DEF format specification. `http://www.si2.org`.

Silicon Integration Initiative [2012]. OpenAccess. `http://si2.org`.

Stroustrup, B. [2000]. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., 3rd edition.

Struzyna, M. [2010]. *Flow-based Partitioning and Fast Global Placement in Chip Design*. PhD thesis, University of Bonn.

Tang, X. and Cho, M. [2011]. Optimal layout decomposition for double patterning technology. In *Computer-Aided Design (ICCAD), 2011 IEEE/ACM International Conference on*, pages 9 –13.

Teig, S. [2002]. The X architecture: Not your father's diagonal wiring. In *Proc. International Workshop on System-Level Interconnect Prediction*, pages 33–37.

Yuan, K., Lu, K., and Pan, D. [2009]. Double patterning lithography friendly detailed routing with redundant via consideration. In *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pages 63–66.

Summary

One of the last major steps in the design of highly integrated circuits (VLSI design) is *routing*. The task of routing is to compute disjoint sets of wires connecting different parts of a chip in order to realize the desired electrical connectivity. There are several different optimization goals that are considered including total wiring length, power consumption or production yield. While this problem even in its most basic form is already NP-hard, in practice it becomes even harder because all resulting metal shapes have to respect a large set of technology dependent *design rules*. Moreover instance sizes in practice are huge, such that routing, as most other steps in VLSI design, cannot be done without sophisticated automated tools.

Design rules define restrictions on the minimum distance and geometry of metal shapes. The intent of most design rules is to forbid patterns that cannot be manufactured well in the lithographic production process. This process has become extremely difficult with the current small feature sizes of 32 nm and below, which are still being manufactured using 193 nm wavelength technology. Because of this, the design rules of modern technologies have become very complex, and computing a routing with a sufficiently low number of design rule violations is a difficult task for automated routing tools. This is, however, a necessary requirement in practice because every remaining violation basically has to be fixed manually by the designers.

In this thesis we present in detail how design rules can be handled efficiently in an automated routing tool. In chapter 2 we develop an appropriate design rule model which considerably reduces complexity while not being too restrictive. This involves mapping complex polygon-based rules to simpler rectangle-based rules and building equivalence classes of shapes with respect to their minimum distance requirements. Our model enables efficient checking of minimum distance rules, which has to be done dozens of times in each routing run. We also discuss efficient data structures that are necessary to achieve this.

We implemented our design rule model within *BonnRoute*, the routing tool of the *BonnTools*, a software package for VLSI physical design developed at the Research Institute for Discrete Mathematics at the University of Bonn in cooperation with IBM. The result is a new module of BonnRoute, called *BonnRoutRules*, which computes this design rule model and embeds BonnRoute in the complex routing environment of current technologies. Chapter 2 also describes the internal structure of this module and some implementation aspects. At the end of the chapter we discuss the handling of design rules which restrict the geometry of shapes instead of defining minimum distance requirements. Vi-

olations of such rules have to be avoided especially in *pin access*, which is an important part of routing and has become considerable more difficult in recent technologies.

The BonnRouteRules module was a key part in enabling BonnRoute to route current 32 nm and 22 nm chips. In chapter 3 we first describe the *combined routing flow* used by IBM in practice, in which BonnRoute solves the main routing task and an industrial standard router is used for postprocessing. We then present detailed experimental results of this flow on real-world designs. The results show that this combined flow produces routings with almost no remaining design rule violations, which proves that our design rule model works well in practice. Furthermore, compared to the industrial standard router alone, the combination with BonnRoute provides several significant benefits: It has 24% less runtime, 5% less wiring length, and over 90% less detours, which shows that with this flow we have an excellent routing tool in practice.