Cell Layout Routing

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Vorgelegt von

Benjamin Klotz

aus

Bonn

Bonn, April 2023

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

Erstgutachter:	Herr Professor Dr. Stefan Hougardy
Zweitgutachter:	Herr Professor Dr. Stephan Held
Tag der Promotion:	07.07.2023
Erscheinungsjahr:	2023

Acknowledgments

I want to thank a number of people who enabled me to write this thesis. First and foremost, I want to thank Prof. Dr. Stefan Hougardy for his excellent guidance, great support as a supervisor, and helpful feedback on early versions of this thesis. I also thank the other professors at the institute for their insights and ideas.

My thanks go out to my colleagues at IBM, who provided the essential integration of BonnCell into the IBM flow and steered the development of BonnCell through continued feedback, new ideas, and expert insights. I am especially grateful for the heartwarming and kind working conditions that Tobias Werner, Dr. Iris Leefken, and Gerhard Hellner provided.

The BonnCell project is a team effort, and I want to thank the many students that participated in its development, including Silas Rathke, Lars Friederichs, Andreas Gwilt, Marena Richter, Iris Hebbeker, Jakob Gierschmann, and Armin Settels. My special thanks go out to Malte Schürks, who, while still a student, already took incredibly good care of countless aspects of the development process and advanced BonnCell in many areas. I am grateful to him for seamlessly taking over this project, which allowed me to finish my thesis, and for proofreading parts of it.

Last but not least, I want to thank my colleagues at the institute for the productive and friendly working environment they provided. I especially want to thank Benjamin Rockel and Dr. Jannik Silvanus for helping out in the development of BonnCell when it was most needed.

Contents

Chapter 1

Introduction

1.1 The cell layout problem

The design of modern computer chips is an immensely complex task that is typically broken down into many subproblems. One of these is the cell layout problem, which is the task of generating a layout for a given input schematic. A schematic, as displayed in fig. 1.1.1, describes a set of transistors and their interconnections, which are called nets. The goal now is to generate a corresponding layout, which is an instruction of how to physically implement the circuit on the chip. A cell layout is given by a set of rectilinear shapes, all associated to a specific layer (see fig. 1.1.2). These shapes indicate either the positions and properties of the transistors or describe the small metal wires that are to be created during the lithographic manufacturing process to physically realize the nets. During the further steps of the chip design process, this layout will be instantiated many times throughout the chip.

Cell layouts are created for a certain manufacturing process, also called a technology node. Due to physical limitations and the extreme complexity of the manufacturing process, the shapes that make up a cell layout must adhere to a wealth of restrictions referred to as design rules. For example, design rules govern spacing requirements between shapes on the same layer, which prevent shorts in the resulting physical circuit. Modern technology nodes typically come with a design rule manual that defines dozens to hundreds of design rules for each layer. Throughout the lifetime of a technology node, design rules are sometimes updated, e.g. because experiments have shown that some design rules can be relaxed without causing problems during manufacturing.

The cell layout problem is subject to a multitude of concurrent objective functions, which further increase its complexity. Typically, cell layouts

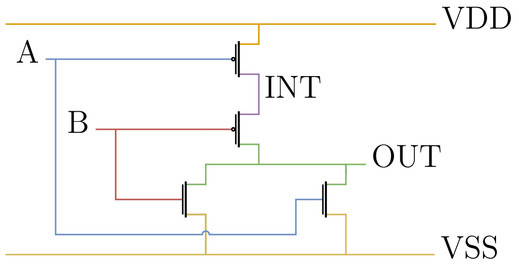


Figure 1.1.1: Visualization of a schematic of a NOR2 cell. This cell computes for the two input nets A and B the logical function (A NOR B), which can be obtained at the output net OUT. Three of the four transistors (black double lines) are connected to either the power supply (VDD) or ground (VSS).

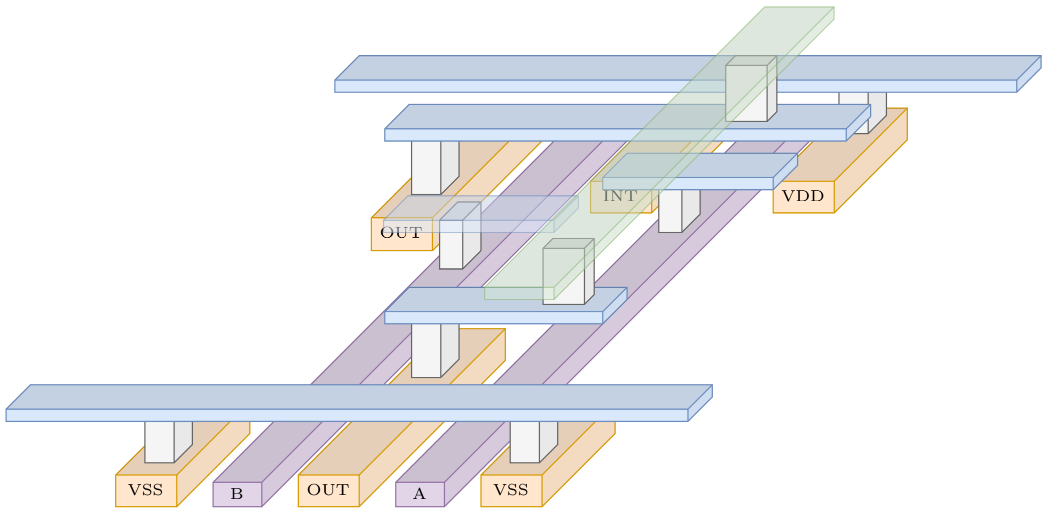


Figure 1.1.2: A layout corresponding to the NOR2 schematic above. Shapes are colored by layer. Some shapes are drawn with transparency for better overall visibility. Wires on adjacent layers are interconnected by gray vias. On the lowest layers, shape triples consisting of an orange, a purple, and another orange shape form transistors. Here, the two orange shapes are the source and drain contacts and the purple shape between them is the gate contact of the transistor. For example, the transistor formed in the lower left corner by the shapes marked with VSS, B, and OUT corresponds to the lower left transistor in the schematic above. Note that both orange and purple shapes can belong to multiple transistors. The blue shapes connected to the purple gates of net A and B do not interconnect anything, but serve as access points (called pins) for connections to other cells.

should be optimized for size and timing characteristics, but should also take into account e.g. electromigration and soft design rules that increase yield if obeyed, but do not have to be fulfilled at all costs.

1.2 Strategies for solving the cell layout problem

The set of cells that is used in the creation of a chip is called library. For each technology node, typically only a few different layouts are generated for each cell in a library, which are then reused with only slight modifications for all chips that are created using this technology node. There have been multiple strategies to obtain the layouts for a library in a new technology node.

One strategy commonly employed is the migration of layouts. In the evolution of technology nodes, there are often many similarities from one node to the next. Additionally, the set of cells in the library does not vary much. In many cases, this allows the cell layouts from one technology node to be migrated into layouts for the next node simply by scaling previous layouts and manually solving any design rule violations that occur in the new layouts. There have been many projects that aim to automate this migration process. For example, they provide automated fix-ups for design rule violations introduced in the migration step [HCN09] or introduce additional whitespace if the migrated cells have too many pins in a too small area [SJKS17].

The layouts that arise from such a migration retain many of the properties of the previous layouts. This is typically desirable because, if the previous layouts were optimized for a specific objective, the new layouts typically also perform well in that regard. However, it can sometimes be beneficial or necessary to redesign layouts from scratch, e.g. because the migration did not yield optimal results, objective considerations changed, or a migration was not possible due to major changes in the design rules.

For a long time, such redesigned cell layouts have been created manually by human experts. However, due to the ever-increasing complexity of the design rules, there have been several projects that aim to automatically generate layouts from scratch, solely based on the technology node at hand and the cell's schematic.

One such automatic layout tool is BonnCell, which is being developed at the Research Institute for Discrete Mathematics in Bonn in close cooperation with our industry partner IBM. The goal of BonnCell is to fully automate the cell design process. Designers at our industry partner can define a schematic and additional parameters, such as the position of pins or adjustments to the objective function, and BonnCell will generate optimized layouts that obey all design rules.

Due to the enormous complexity of the cell layout problem, BonnCell and most other automatic layout tools employ a two-stage approach. First, during the placement phase, the sizes and positions of the transistors are chosen. Then during the routing phase, the shapes that interconnect the transistor contacts are generated.

In this work, we will explore BonnCell's routing engine in detail. In the first part of chapter 3 we discuss the Cell Layout Routing Problem and how it can be modelled using an integer linear programming formulation (ILP). We compare the employed multicommodity flow ILP formulation to its extension, the multi-root multicommodity flow formulation, and prove that the latter has, in general, a worse integrality gap. In the second half of chapter 3, we introduce a streamlined methodology that simplifies the laborious task of implementing and updating the large number of design rules in BonnCell. To this end, we introduce the notion of partial polygons, which will act as a common model for all shapes that make up a layout. Based on this, we create a design rule framework that allows for efficient encoding of design rules by mimicking structures found in design rule manuals.

In chapter 4, we extend BonnCell's routing engine to incorporate another objective function. Here, our goal is to find layouts with improved pin accessibility, that is, layouts in which the pins are spaced out and enlarged to allow for simpler and more efficient interconnections between cells. We examine pin accessibility scoring formulations used in the literature and derive a new scoring formulation that is optimized for incorporation into our routing ILP formulation. We prove that computing our new scoring formulation is NP-hard, but observe that it can be efficiently computed in practice by using a SAT formulation. Additionally, we prove that it can be encoded concisely in the ILP, which allows us to compute pin accessibility optimized cell layouts within adequate runtime.

Chapter 2

Previous Work

There have been many publications about the automatic generation of cell layouts in chip design. We try to give an overview of the different strategies employed.

While most projects separate the cell layout generation into a placement and routing phase, there have been a few attempts to solve them simultaneously. In [Tho19] the entire cell layout problem is formulated as one large ILP. This allows to find optimum cell layouts with respect to netlength, albeit only for smaller instances due to runtime issues on larger instances. A similar approach is pursued in [Lee+21; Che+21; CHLL21] and related works. Instead of a MIP formulation, they encode the entire problem using satisfiability modulo theory (SMT), an extension to the well known SAT-based mathematical programming. With this, they are able to generate layouts for entire libraries within hours that optimize several objective functions simultaneously.

2.1 Placement

The task of the placement phase is to find positions and sizes for all the transistors in the schematic, such that, together with the subsequent routing step, an optimized layout is produced. With increased transistor density in recent technology nodes, not all combinations of transistor positions and sizes are routable. Therefore, most approaches incorporate some form of routability test or estimation during the placement phase.

In recent FinFET based technology nodes [23], transistors cover a rectangular area whose x and y coordinates are gridded. This enables the placement approach employed in BonnCell, namely a branch&bound strategy that can enumerate all possible combinations of transistor positions and sizes. Par-

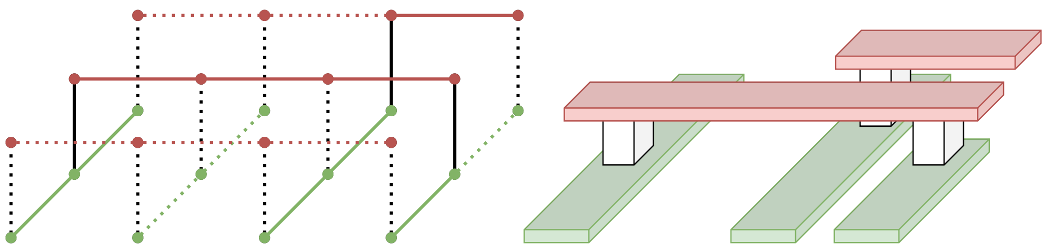


Figure 2.2.1: The left side depicts a routing graph. Edges and vertices are colored by layer. Solid lines indicate active edges. The right side shows the shapes that the active edges get translated into.

tial solutions and candidates for the final solution are tested for routability by calling BonnCell's routing phase as a subroutine. A detailed description of BonnCell's placement engine and its interaction with the routing engine can be found in [Cre19]. A related approach is presented in [Li+19], where dynamic programming is used to explore the entire placement search space.

There have been several other approaches to solve the placement phase. These include ILP formulations (e.g. [Lu+15]), SAT based approaches (e.g. [SR19]), and many heuristic approaches (e.g. [Wu+13]).

2.2 Routing

The task of the routing phase is to generate rectilinear shapes that implement the cell's internal wiring. The created shapes need to interconnect all nets correctly without producing any shorts or opens, and additionally the created shapes should adhere to all design rules. The common approach pursued by most projects is to define a routing graph, whose vertices and edges are embedded into the three-dimensional cell area. After solving the Steiner tree packing problem (STPP) in this routing graph to create one Steiner tree per net, those vertices and edges assigned to a net are transformed into shapes (see fig. 2.2.1). The underlying routing graph is typically chosen in a way that implicitly avoids most design rule violations. This way, most projects avoid encoding all design rules explicitly. Some design rules that are not implicitly obeyed by the model are explicitly obeyed while solving the STPP, by forbidding certain combination of vertices and edges. Additionally, many projects clean up some design rule violations that still occur after the translation from vertices and edges into shapes by a heuristic post-optimization step.

For the various steps of this procedure, several strategies have been employed. The design rule aware Steiner Tree packing is done e.g. by an SMT formulation in [Che+21], a machine learning approach in [Ho+23], or an ILP-based approach in [Li+19].

For the post-optimization step that cleans up remaining design rule violations, special algorithms have been proposed for several types of design rules. For example, [Xu+15] uses a mixed integer programming formulation to solve design rule violations arising from complex double patterning rules. Recently, there have been some publications ([RFK21; Ho+23] and related works) that use a machine learning approach to fix up design rules. A neural network that is trained on predesigned layouts is able to fix design rule violations in newly generated cell layouts. Such an approach is independent of the technology node and avoids the effort to create new post-processing routines for every new technology node.

All of these approaches have the following problem. During the solving of the STPP, design rules are only considered in terms of relatively coarse edges. Due to the large number of design rules, not all design rules can be represented exactly by forbidding combinations of edges to be active at the same time. An example of such a situation is visualized in fig. 2.2.2. For such a design rule, there are now two options. Either, the rule is translated into edge restrictions that are pessimistic, i.e. the solution space is artificially reduced. Or, the rule is translated into edge restrictions with some optimism, relying on the post-processing to fix potential remaining design rule violations. However, post-processing may fail to resolve all design rule violations, especially in cases where multiple conflicts have to be resolved simultaneously.

In the end, this can lead to suboptimal solutions or even to the failure to find any routing for a placement that is, in fact, routable without these artificially limitations. BonnCell tries to avoid these effects by encoding shapes and design rules as exact as possible in its ILP formulation. See [Cre19] for an example of how this has been done in the context of line end trim shapes. This exact modelling approach has several advantages.

- The shapes generated by BonnCell's ILP formulation contain virtually no design rule violations and do not need to be post-processed.
- The approaches presented above typically need to manually translate the design rules from the design rule manual into implications in terms of forbidden vertices and edges in the routing graph. This translation also needs to take into account the capabilities of the post-processing step. BonnCell's streamlined process to define design rules that we propose in chapter 3 automates most of this manual translation effort between the design rules and the routing model.
- Occasionally, BonnCell can produce innovative layouts that even expe-

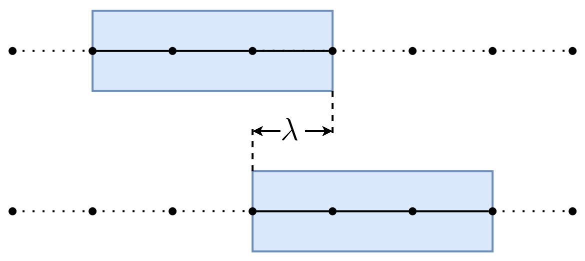


Figure 2.2.2: Depicts two blue shapes that are induced by the active (solid) edges of the routing graph. If we assume that a design rule requires that the space indicated by λ must be a bit larger than depicted, this is an example of a design rule that cannot be exactly represented in terms of allowed or forbidden edge combinations. One can now either forbid this combination of active edges, which would reduce the solution space. Or, one can rely on the post-processing step to slightly enlarge the shapes to fix the design rule violation. However, if multiple of these rule violations occur in close proximity, it might not be possible to resolve all design rule violations simultaneously by only a local post-processing step.

> rienced layout designers did not come up with. Even human experts typically do not know the entire design rule manual by hand, but design layouts based on patterns they saw before. When switching to new technology nodes, new patterns can become feasible and by closely adhering to the design rules, BonnCell can help to find such innovations.

Despite its many advantages, this exact modeling approach has one major drawback, namely its large implementation effort. Encoding shapes and their many design rules in an ILP is difficult. And since design rules change periodically, it requires a lot of effort to keep BonnCell up-to-date. The next chapter addresses this issue by introducing a novel modelling approach for BonnCell's routing engine.

Chapter 3

Modeling the Cell Layout Routing Problem

3.1 Introduction

The goal of this chapter is to provide a complete overview of the models and implementation techniques employed in BonnCell's routing engine. It is structured as follows. We first describe the Cell Layout Routing Problem and its main challenge, namely the large amount of ever-changing design rules. We explore practical and theoretical aspects of its implementation as an integer linear program (ILP) in section 3.2. Our main contribution in this chapter is the introduction of a new way to model BonnCell's routing phase. The motivation for this change is outlined in section 3.3. In short, we want to reduce the large programming effort that used to be required to keep BonnCell up-to-date with changes in the design rules. To this end, we first introduce a common model in section 3.4 that is able to unify the representation of all the shapes required in a cell layout. Finally, based on this unified model, we introduce a streamlined process to encode design rules in BonnCell in section 3.5 that significantly simplifies the implementation process.

3.1.1 Defining the routing problem

Let us define the Cell Layout Routing Problem. For this, we start with some definitions.

Definition 3.1. We define a **metal shape** to be a tuple (p, l), where p is a rectilinear polygon in $\mathbb{Z}^2$ without self-intersections or holes, and l is a layer.

Remark. The restriction of metal shapes to discrete coordinates is present in all technology nodes we have observed so far. Typically, the design rule manual requires metal shape coordinates to be a multiple of a base-unit, which is defined as 0.25 nanometers in current state-of-the-art technology nodes.

Definition 3.2. We say two metal shapes $(p_1, l_1), (p_2, l_2)$ are **connected**, if p_1 intersects p_2 and l_1 and l_2 are the same or adjacent layers. Additionally, we call a set of metal shapes $R = \{ (p_1, l_1), \dots, (p_k, l_k) \}$ **connected**, if the following undirected connectivity graph G is connected, namely $G = (V, E)$, with $V := R$ and $E := \{ (v, w) \in V^2 \mid v \neq w, v \text{ and } w \text{ are connected} \}$.

Definition 3.3. We define a **pin** to be a set of metal shapes and a **net** to be a set of pins. Sometimes we refer to pins as **terminals**.

Definition 3.4. A **routing** R_N of a net N is a connected set of metal shapes, so that one shape of each pin of N is connected. More precisely, we require for all $P \in N$ that there exists a pair of metal shapes $(p_1, l) \in P$, $(p_2, l) \in R_N$ such that p_2 completely covers p_1.

Definition 3.5. A **design rule** is a function f that decides if given set of routings $\{ R_{N_1}, \dots, R_{N_k} \}$ for nets $\{ N_1, \dots, N_k \}$ is feasible or infeasible.

Cell Layout Routing Problem (CLRP)

Instance:	• Allowed layers L • Feasible routing area $A = [x_{min}, x_{max}] \times [y_{min}, y_{max}] \subset \mathbb{Z}^2$ • Nets $\mathcal{N} = \{ N_1, \dots, N_k \}$ • A set of design rules
Task:	For each net $N \in \mathcal{N}$, find a routing R_N on layers L. The resulting set $\{ R_N \mid N \in \mathcal{N} \}$ has to be feasible according to all design rules. If no such solution exists, output "Infeasible".

For each technology node, there typically only exists a small number of combinations of allowed layers, cell heights $[y_{min}, y_{max}]$, and design rules, that occur in real-world CLRP-instances. It is common to define one specialized algorithm for each of these combinations. We will therefore from now on assume allowed layers, cell height $[y_{min}, y_{max}]$, and design rules to be constant, leaving only the cell width $[x_{min}, x_{max}]$ and nets as input for the CLRP.

3.1.2 Complexity and algorithm strategy

The CLRP is very hard to solve in practice. Its main source of complexity comes from the large set of design rules, which are difficult to handle for both human and algorithmic solvers. These design rules serve two main purposes.

- Design rules are defined by the foundry, which will eventually produce the chip, to ensure that the desired layouts can be manufactured and the resulting chip works as intended. E.g. metal shapes require a minimum size to be manufacturable and metal shapes of different nets require a certain minimum spacing in order to prevent shorts.
- Design rules can also be introduced to simplify and structure the entire chip design process. E.g. cell heights are typically limited to a few discrete values, so that cells can be packed easily and efficiently. Similarly, on many layers the feasible positions of metal shapes are limited. This ensures that multiple cells can be easily connected with each other and also simplifies some of the foundry's design rules.

Modern cell layouts must adhere to hundreds of design rules that are changed and updated periodically, most notably when the manufacturing process changes. BonnCell strives to create layouts that adhere to all design rules. The large number of ever-changing rules make it too great of an effort to design combinatorial algorithms that solve the LCRP exactly. Instead, defining an integer linear program (ILP) and solving it using a general purpose ILP solver has proven to be a more practical approach. However, defining such an extensive ILP is still a challenging task.

3.2 Modeling constraints in the ILP

When writing complex ILPs, great care has to be taken that the resulting ILP formulation is both correct and can be solved efficiently. Furthermore, plain ILP constraints are often difficult to understand and can obfuscate the logical reasoning they represent. To mitigate these problems, in this work we typically describe ILP constraints using a more abstract syntax than plain ILP constraints. Nevertheless, these abstract descriptions can be translated into ILP constraints. In the following, we will describe some of these abstractions.

Remark. For the same reasons outlined above, we also use a similar abstraction layer in our ILP generating code.

3.2.1 Modeling implications

Most prominently we will be using the notion of implications. In the following, x_i will denote integer variables with bounds $l_i \leq x_i \leq u_i$, z will denote a binary “activation” variable, and λ_i, θ will be constants. Implications of the form

$$(z = 1) \implies \left(\sum_{i=1}^{k} \lambda_i x_i \geq \theta\right) \tag{3.1}$$

can be modeled as an ILP constraint as follows:

$$\left(\sum_{i=1}^{k} \lambda_i x_i \geq \theta + M(1 - z)\right) \tag{3.2}$$

where

$$M := m - \theta$$

with

$$m := \left(\sum_{i \in \{1,\ldots,k\} \text{ where } \lambda_i > 0} \lambda_i l_i\right) + \left(\sum_{i \in \{1,\ldots,k\} \text{ where } \lambda_i < 0} \lambda_i u_i\right).$$

If $z = 1$, the right-hand side in eq. (3.2) equals θ, which yields exactly the desired constraint. Conversely, if $z = 0$, the right-hand side in eq. (3.2) equals $(\theta + M)$, which is the minimum value that the left-hand side can attain for any choice of the x_i, i.e. the entire constraint is trivially fulfilled. The method is known as the big-M method (see [Rub11], not to be confused with the simplex big-M method [22a]) and have been used in BonnCell for a long time (see [Cre19]).

Note that we can chain the big-M method multiple times to model constraints that depend on multiple activation variables. In this case, it does not matter in which order the big-M method is applied to the activation variables, because the value of M remains unchanged after each application of the method. To see this, let us compute the M values when applying the big-M method twice for binary variables z_1 and z_2.

$$\begin{aligned}
& (z_2 = 1) \implies \left[(z_1 = 1) \implies \left(\sum_{i=1}^{k} \lambda_i x_i \geq \theta\right)\right] \\
\Leftrightarrow \quad & (z_2 = 1) \implies \left(\sum_{i=1}^{k} \lambda_i x_i + M_1 z_1 \geq \theta + M_1\right) \\
\Leftrightarrow \quad & \sum_{i=1}^{k} \lambda_i x_i + M_1 z_1 \geq \theta + M_1 + M_2(1 - z_2)
\end{aligned}$$

Here, by construction we have

$$
\begin{aligned}
M_1 &= m_1 - \theta \\
M_2 &= \begin{cases} m_1 - \theta - M_1 & \text{if } M_1 > 0 \\ m_1 + M_1 - \theta - M_1 = m_1 - \theta = M_1 & \text{if } M_1 \leq 0 \end{cases}
\end{aligned}
$$

We can assume that $M_1 \leq 0$ and thus $M_2 = M_1$. Otherwise, we would have $m_1 > \theta$ which means that the original constraint $\left(\sum_{i=1}^{k} \lambda_i x_i \geq \theta\right)$ was trivially fulfilled anyway, regardless of the values of the variables, in which case we omit the entire construction.

Real world constraints often require chaining implications. For example, a design rule may require that a constraint C holds, but only if conditions A and B are met. Care has to be taken not to chain the big-M method more times than necessary, because this can lead to ILP formulations with a large integrality gap, which are typically difficult to solve. In the following section we describe such an example.

3.2.2 Modeling constraint disjunctions

Another common technique for modeling real world problems, such as design rules, as ILPs are constraint disjunctions. A constraint disjunction is a family of sets of constraints $C_1, \dots, C_k$ with the semantic that only the constraints of one set C_i have to be fulfilled. This construction is frequently used to model equally valid choices (e.g. "either go left i.e. fulfill C_1, or go right i.e. fulfill C_2"). Oftentimes, constraint disjunctions occur together with an additional binary activation variable z in the form:

$$(z = 1) \implies (\text{One of the sets of constraints } C_1, \dots, C_k \text{ has to hold})$$

Such a requirement could be modeled in the ILP by adding variables and constraints as follows.

Algorithm 1:

```
1 Create binary variables $a_1, \dots, a_k$
2 Add constraint $(a_1 + \cdots + a_k = 1)$
3 for all $i \in \{1, \dots, k\}$ and constraint $c \in C_i$ do
4     Add constraint $((z = 1) \implies ((a_i = 1) \implies c))$
5 end
```

However, using a small reformulation, we can eliminate one of the implications.

Algorithm 2:

```
1 Create binary variables $b_1, \ldots, b_k$
2 Add constraint $(b_1 + \cdots + b_k = z)$
3 for all $i \in \{1, \ldots, k\}$ and constraint $c \in C_i$ do
4 |   Add constraint $((b_i = 1) \implies c)$
5 end
```

The two ILP formulations do not differ in their integer feasible solutions. But, depending on the constraint disjunction $C_1, \ldots, C_k$, algorithm 1 may yield a weaker ILP relaxation than algorithm 2. This can be seen in the following example.

Assume that all $C_1, \ldots, C_k$ contain the constraint $(x \geq 1)$ for some binary variable x. Then the formulation in algorithm 1 would yield the constraints $(a_1 + \cdots + a_k = 1)$ and $(x \geq a_j + z - 1)$ for $j = 1, \ldots, k$. Algorithm 2 would yield the constraints $(b_1 + \cdots + b_k = z)$ and $(x \geq b_j)$ for $j = 1, \ldots, k$.

In the LP relaxation where the variables need not be integer, let us examine the lowest possible value for x. In the first formulation, the lowest possible value for x is attained by setting $a_1 = \cdots = a_k = 1/k$, which yields $x \geq \frac{1}{k} + z - 1$. In the second formulation, it is attained by setting $b_1 = \cdots = b_k = \frac{z}{k}$, which induces a lower bound $x \geq \frac{z}{k}$. Note that for $k \geq 2$, $0 < z < 1$ we have

$$\left(\frac{1}{k} + z - 1\right) - \frac{z}{k} = \frac{k(z-1) + 1 - z}{k} \leq \frac{2(z-1) + 1 - z}{k} = \frac{z-1}{k} < 0$$

and therefore the second formulation yields a stronger bound on x.

3.2.3 Modeling Steiner tree packing

3.2.3.1 Multicommodity flow formulation

At its core, our ILP will solve an instance of the vertex-disjoint Steiner tree packing problem on a predefined routing graph $G = (V, E)$, in order to solve the CLRP. Each terminal of a net will be represented as a set of vertices in G. For simplicity of notation, in the following we assume that each terminal is represented by a single vertex. To model the Steiner tree packing problem in the ILP, we will use the multicommodity flow formulation that has already been described in [GM93; HK12; vCHSW19]. It uses $\mathcal{O}(|E| + |V|)$ variables and constraints for each terminal of a net. While its exact integrality gap is unknown, [Vic18] has shown it lies between $\frac{6}{5}$ and 2. Closely related versions of this formulation have already been studied in [Won84; Cho94]. [Pol03] generalized the multicommodity flow formulation we present here to a scheme which can gradually improve the integrality gap, at the cost of increasing the size of the ILP formulation.

The multicommodity flow ILP formulation works as follows. For each net n, we will mark one of its terminals as root $t_{\text{root},n} \in n$. We denote by $\bar{n} := n \setminus \{ t_{\text{root},n} \}$ the non-root terminals of net n. The ILP formulation will model a net's connectivity by sending a unit-flow from the net's root terminal t_{root} to all other terminals. Let us define the set of directed edges as $\overrightarrow{E} := \{ (v,w) \mid v,w \in V \text{ and } \{ v,w \} \in E \}$. The ILP formulation is chosen as follows.

Variables:

$x_v^n \in \{0,1\}$	$\forall v \in V$ and net n	Vertex usage variables
$x_e^n \in \{0,1\}$	$\forall e \in E$ and net n	Edge usage variables
$\overrightarrow{x}_e^n \in \{0,1\}$	$\forall e \in \overrightarrow{E}$ and net n	Directed edge usage variables
$\overrightarrow{f}_{e,t}^n \in \{0,1\}$	$\forall e \in \overrightarrow{E}$, net n, $t \in \bar{n}$	Directed flow variables

Objective

$$\min \left(\sum_{e \in E} \operatorname{cost}(e,n)\ x_e^n \right) + \left(\sum_{v \in V} \operatorname{cost}(v,n)\ x_v^n \right)$$

Constraints:

$$\sum_{\text{net } n} x_v^n \leq 1 \qquad \forall v \in V \tag{3.3}$$

$$\overrightarrow{f}_{e,t}^n \leq \overrightarrow{x}_e^n \qquad \forall e \in \overrightarrow{E}, \text{net } n, t \in \bar{n} \tag{3.4}$$

$$\overrightarrow{x}_{(v,w)}^n + \overrightarrow{x}_{(w,v)}^n \leq x_e^n \qquad \forall \{ v,w \} = e \in E, \text{net } n \tag{3.5}$$

$$x_v^n \geq x_e^n \qquad \forall e \in E, v \in e, \text{net } n \tag{3.6}$$

$$\sum_{w \in \Gamma(v)} \overrightarrow{f}_{(v,w),t}^n - \overrightarrow{f}_{(w,v),t}^n = \begin{cases} 1 & v = t_{\text{root},n} \\ -1 & v = t \\ 0 & \text{else} \end{cases} \qquad \forall v \in V, \text{net } n, t \in \bar{n} \tag{3.7}$$

We refer to this ILP as the **multicommodity flow formulation** or **mcf formulation**. In a solution to the *mcf* ILP, the edge and vertex usage variables with value 1 will form Steiner trees for each net. Constraints 3.3 will ensure that these Steiner trees are vertex and thus edge disjoint. Constraints 3.7 will ensure that the flow variables indeed form a flow and constraints 3.4 and 3.5 ensure that active flows induce edge usages. Here, constraints 3.5 utilize that due to the costs being non-negative, there exists an optimum solution in which the active flow variables do not form a cycle, which implies that an edge cannot be used in both directions at the same time. Finally, constraints 3.6 enforce integrity between the vertex and edge usage variables.

3.2.3.2 Corridors

In a typical CLRP instance, for most nets the bounding box of their terminals only cover a small portion of the feasible routing area. In [Cre19], this is leveraged to significantly reduce the size of the multicommodity flow formulation by using so-called routing corridors. For each net n and terminal $t \in \bar{n}$, only the flow variables for edges are generated, which lie in the terminal's routing corridor. The routing corridor is defined as a slightly enlarged bounding box of t and $t_{\text{root},n}$. The total sizes of these bounding boxes are reduced by choosing $t_{\text{root},n}$ as a terminal that lies “in the middle”. This choice of corridors is visualized in the left half of fig. 3.2.1.

Of course, this approach limits the solution space and can thus reduce the solution's quality or even fail to find any solution for feasible CLRP instances. However, tests in [Cre19] have shown that the solution quality is mostly unaffected by this limitation. At the same time, the limited solution space and the much smaller ILP formulation significantly improve solving speeds.

3.2.3.3 Multiple roots

In order to further reduce the number of variables and constraints in the multicommodity flow formulation, we introduce the idea to use multiple roots. Instead of having a single root terminal that sends flow to all other terminals for each net n, we precompute an arborescence R^n on the terminals of n. Each terminal $s \in \mathrm{V}(R^n)$ will now act as the root terminal for its descendants $\Gamma^+(s)$, i.e. each edge $(s,t) \in \mathrm{E}(R^n)$ will induce one $s-t$ flow. As in the previous approach, we limit each $s-t$ flow to a routing corridor. Of course, this approach can restrict the solution space even further. At the same time, fig. 3.2.1 illustrates how a well-chosen arborescence R^n can again significantly reduce the size of the ILP formulation, which in turn decreases solving times. Using this notion, the previous formulation can be described as choosing R^n as a star.

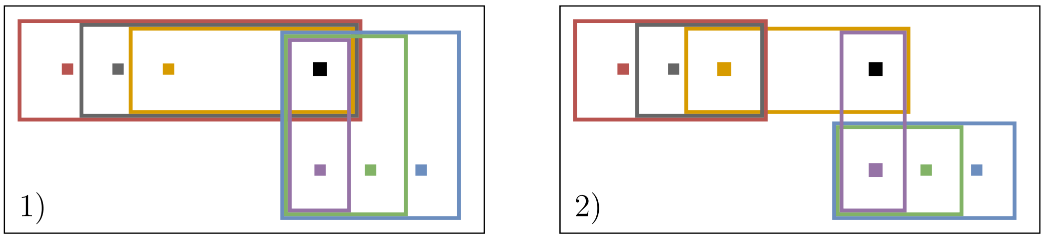


Figure 3.2.1: Both images visualize a net with 7 terminals (small colored squares) and their corresponding routing corridors (colored rectangles). 1) depicts the routing corridors as described in section 3.2.3.2, where the black terminal is chosen as root. 2) shows the routing corridors when multiple terminals are roots. Here, the arborescence R is defined by its edge set E(R) = { (⬛, 🟧), (⬛, 🟪), (🟧, ◼), (🟧, 🟥), (🟪, 🟩), (🟪, 🟦) }.

In the following we will first describe the necessary changes to the *mcf* ILP formulation and later describe how R^n can be chosen. The most notable difference in the ILP formulation comes from the fact that in an optimum solution an edge may have active flow variables in both directions, if both flows originate from different roots (see fig. 3.2.2). This means that constraints 3.5 cannot be used for flows from different roots.

The full ILP formulation is described as follows. For an arborescence R^n, we define the set of terminals that act as roots as

$$\bar{\mathrm{V}}(R^n) := \{\, s \in \mathrm{V}(R^n) \mid \Gamma^+(s) \neq \emptyset \,\}\,.$$

Then, for each net n and each $s \in \bar{\mathrm{V}}(R^n)$ we instantiate the variables and constraints from the previous formulation, with s as the root terminal and $\Gamma^+(s)$ as the non-root terminals. To distinguish the variables in the different instantiations of the *mcf* formulation, we annotate each of them with ^{s}. Then we add the following additional variables and constraints and choose the following objective function.

Variables:

$x_v^n \in \{0,1\}$	$\forall v \in V$ and net n	Vertex usage variables
$x_e^n \in \{0,1\}$	$\forall e \in E$ and net n	Edge usage variables

Objective

$$\min \left(\sum_{e \in E} \mathrm{cost}(e,n)\ x_e^n \right) + \left(\sum_{v \in V} \mathrm{cost}(v,n)\ x_v^n \right)$$

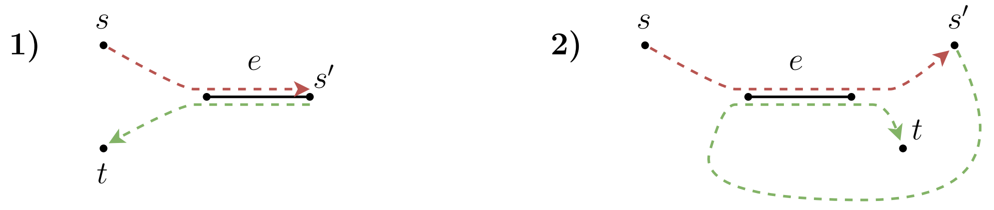


Figure 3.2.2: Both pictures show an $s - s'$ flow in red and an $s' - t'$ in green. 1) shows that constraints 3.5 are no longer valid for the *multi-root mcf* formulation. 2) illustration of the reasoning behind constraints 3.10.

Constraints:

$$
\begin{aligned}
& x_v^{n,s} \leq x_v^n && \forall v \in V, \text{ net } n,\ s \in \bar{\mathrm{V}}(R^n) && (3.8)\\
& x_e^{n,s} \leq x_e^n && \forall e \in E, \text{ net } n,\ s \in \bar{\mathrm{V}}(R^n) && (3.9)\\
& \overrightarrow{f}_{e,s'}^{n,s} + \overrightarrow{x}_e^{n,s'} \leq x_{\{v,w\}}^{n,s} && \forall \{v,w\} = e \in \overrightarrow{E}, \text{ net } n, & \\
& && (s,s') \in \mathrm{E}(R^n) \text{ with } \Gamma_{R^n}^+(s') \neq \emptyset && (3.10)
\end{aligned}
$$

We refer to this ILP formulation as the **multi-root multicommodity flow formulation** or, in short, **multi-root mcf**. Constraints 3.8 and 3.9 simply identify the vertex and edge usages of the different instantiations of the previous formulation with each other. Constraints 3.10 are not necessary for correctness, but improve the ILP relaxation significantly, as can be seen in fig. 3.2.3. They are a weaker version of the constraints 3.5, utilizing again that there are optimum solutions to the underlying Steiner tree packing problem in which the active edges do not form any cycles in G. Figure 3.2.2 illustrates how a violation of these constraints would induce a cycle in the resulting Steiner tree. The implementation and many details of above ILP description have been developed by Malte Schürks.

We have seen in fig. 3.2.1 on page 17 that the *multi-root mcf* formulation, together with the corridors, can reduce the size and solution space of the ILP formulation in comparison to the *mcf* formulation. Typically, this leads to faster solving times for the *multi-root mcf* formulation. However, this effect is countered by the fact that that in general, the *multi-root mcf* formulation yields a weaker ILP formulation than the *mcf* formulation. At the end of this section, we will present experiments that show that in most cases the first effect is more dominant, meaning that the *multi-root mcf* formulation is faster than the *mcf* formulation. But first we will prove that the *mcf* formulation, in general, yields the stronger ILP formulation. More precisely, the following theorem shows that on any instance, the integrality gap of the

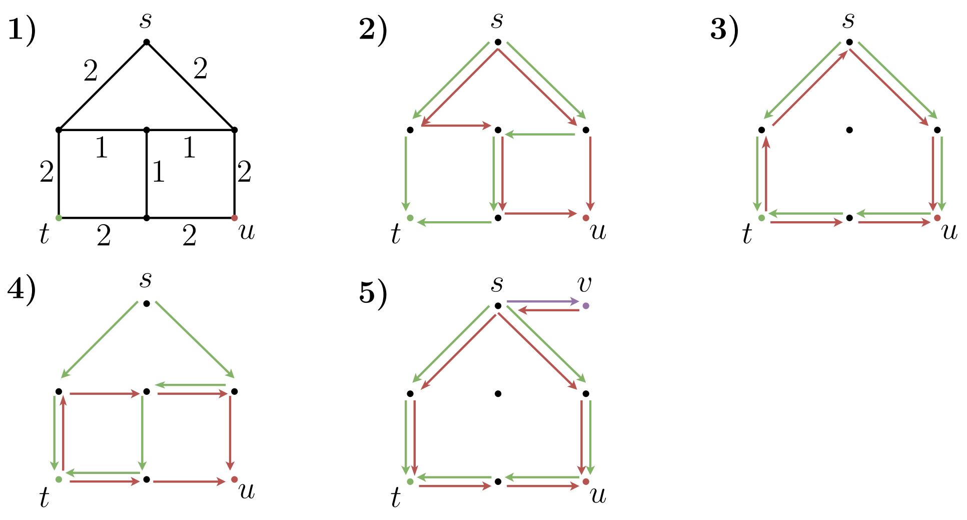


Figure 3.2.3:
1) depicts a routing graph with three terminals s, t, u and edge costs between 1 and 2. A Steiner tree connecting all three terminals has cost at least 8. The following four pictures all show fractional flows, where each arrow indicates a flow of $\frac{1}{2}$ (except for two arrows in the last picture).
For this instance, the optimum objective value for the LP relaxation of the *mcf* formulation is 7.5, which is attained by setting all edge usages to $\frac{1}{2}$. 2) depicts the corresponding flows $s - t$ (green) and $s - u$ (red), assuming that s is chosen as root terminal. This is a standard example which has already been used, e.g. in [Pol03].
For 3) and 4), let us fix an arborescence R on the terminals by setting $\mathrm{E}(R) := \{ (s,t), (t,u) \}$. Then, in the *multi-root mcf* formulation without constraints 3.10, the optimum value of the LP relaxation is 6. This is attained by choosing the flows $s-t$ and $t-u$ as depicted in 3), which results in all outer edges having a usage value of $\frac{1}{2}$. For the *multi-root mcf* formulation with constraints 3.10, the optimum value of the LP relaxation is again 7.5, as in the *mcf* formulation. This is attained by choosing the flows as shown in 4). Note that the $t - u$ flow in 4) can be constructed from the two flows shown in 2) by "subtracting" the $s - t$ flow from the $s - u$ flow, as it is also done in the proof of theorem 3.6.
However, there are instances in which the *multi-root mcf* formulation with constraints 3.10 still yield worse results than the *mcf* formulation. This can be seen in 5), where we have extended our instance by a terminal v which is connected to a single edge $\{ s, v \}$ with cost 0. The depicted flow shows an LP solution of cost 6 for the *multi-root mcf* formulation with $\mathrm{E}(R) := \{ (s,t), (s,v), (v,u) \}$, while the *mcf* formulation again yields an LP solution of cost 7.5, analogously to 2).
We have verified computationally that all depicted LP solution are of minimum costs for their respective instances and formulations.

multi-root mcf formulation is at least as large as the integrality gap of the *mcf* formulation for that instance, regardless of the chosen arborescences.

Theorem 3.6. Assume we are given an instance of the Steiner tree packing problem with routing graph $G = (V, E)$ and a set of nets with one terminal marked as root $t_{\mathrm{root},n} \in n$ for each net n. Let (x, f) be a solution of the LP relaxation of the corresponding *mcf* formulation and $o(x, f)$ its objective value. Additionally, for each net n, let R^n be an arborescence with root $t_{\mathrm{root},n} \in n$ on the net's terminals. Then, for the LP relaxation of the corresponding *multi-root mcf* formulation, there exists a solution (x', f') that has cost $o(x, f)$.

Proof. The idea of the proof is to construct an $s-t$ flow for all $(s, t) \in \mathrm{E}(R^n)$, by "subtracting" the $t_{\mathrm{root}}-s$ flow from the $t_{\mathrm{root}}-t$ flow from the *mcf* solution. An example of this is shown in fig. 3.2.3, 4). More precisely, we construct such a solution (x', f') to the *multi-root mcf* formulation from the given solution (x, f) as follows. For each net n, $(v, w) \in \overrightarrow{E}$, $e \in \mathrm{E}$, and $(s, t) \in \mathrm{E}(R^n)$ we set:

$$\begin{aligned}
d^{n,s}_{(v,w),t} &:= \overrightarrow{f}^{n}_{(v,w),t} - \overrightarrow{f}^{n}_{(w,v),t} - \overrightarrow{f}^{n}_{(v,w),s} + \overrightarrow{f}^{n}_{(w,v),s}{}^{*} \\
\overrightarrow{f}'^{n,s}_{(v,w),t} &:= \max(0, d^{n,s}_{(v,w),t}) \\
\overrightarrow{x}'^{n,s}_{(v,w)} &:= \max_{t \in \Gamma^+(s)} \overrightarrow{f}'^{n,s}_{(v,w),t} \\
x'^{n,s}_{e} &:= x^{n}_{e} \\
x'^{n}_{e} &:= x^{n}_{e} \\
x'^{n,s}_{v} &:= x^{n}_{v} \\
x'^{n}_{v} &:= x^{n}_{v}
\end{aligned}$$

We claim that (x', f') is a valid solution to the *multi-root mcf* formulation of cost $o(x, f)$.

By construction, the objective values of (x, f) and (x', f') are the same in their respective ILP formulations. It remains to show that (x', f') fulfills all constraints and variable bounds. To see the latter, we observe that

$$\left| d^{n,s}_{(v,w),t} \right| = \left| (\overrightarrow{f}^{n}_{(v,w),t} + \overrightarrow{f}^{n}_{(w,v),s}) - (\overrightarrow{f}^{n}_{(v,w),s} + \overrightarrow{f}^{n}_{(w,v),t}) \right| \leq 1,$$

*Here and in the following we simplify the notation by setting $\overrightarrow{f}^{n}_{(v,w),s} = \overrightarrow{f}^{n}_{(w,v),s} = 0$ if s is the root of the arboresence R^n. This avoids a case distinction, because there is no flow defined that goes into s. Note that the following computations remain valid with this extension of the definition.

because by constraints 3.4 and 3.5 both terms in the brackets are between 0 and 1. Thus, also $\overrightarrow{f'}^{n,s}_{(v,w),t}$ lies between 0 and 1.

We will now check the constraints one by one. Constraints 3.3, 3.6, 3.8, and 3.9 hold for (f', x') because they also hold for (f, x). Constraints 3.4 are fulfilled by definition. To see that the flow conservation constraints 3.7 hold, we first observe that for $\{v, w\} \in E$ we have $d^{n,s}_{(v,w),t} = -d^{n,s}_{(w,v),t}$ and thus $\overrightarrow{f'}^{n,s}_{(v,w),t} - \overrightarrow{f'}^{n,s}_{(w,v),t} = d^{n,s}_{(v,w),t}$. Then, the following observation for $v \in V$ proofs that constraints 3.7 hold:

$$\sum_{w\in\Gamma(v)} \left(\overrightarrow{f'}^{n,s}_{(v,w),t} - \overrightarrow{f'}^{n,s}_{(w,v),t}\right) = \sum_{w\in\Gamma(v)} d^{n,s}_{(v,w),t}$$

$$= \sum_{w\in\Gamma(v)} \left(\overrightarrow{f}^{n}_{(v,w),t} - \overrightarrow{f}^{n}_{(w,v),t}\right) - \sum_{w\in\Gamma(v)} \left(\overrightarrow{f}^{n}_{(v,w),s} - \overrightarrow{f}^{n}_{(w,v),s}\right)$$

$$= \left\{\begin{array}{rl} 1 & v = t_{\text{root},n} \\ -1 & v = t \\ 0 & \text{else} \end{array}\right\} - \left\{\begin{array}{rl} 1 & v = t_{\text{root},n} \\ -1 & v = s \\ 0 & \text{else} \end{array}\right\} = \left\{\begin{array}{rl} 1 & v = s \\ -1 & v = t \\ 0 & \text{else} \end{array}\right.$$

Let us now check that constraints 3.5 hold for (x', f'). Let $s \in \bar{\mathrm{V}}(R^n)$, $t, t' \in \Gamma^+(s)$ and $(v, w) \in \overrightarrow{E}$. Then, we make the following two preliminary observations.

- $$\begin{aligned} d^{n,s}_{(v,w),t} - d^{n,s}_{(v,w),t'} &= \left(\overrightarrow{f}^{n}_{(v,w),t} - \overrightarrow{f}^{n}_{(w,v),t} - \overrightarrow{f}^{n}_{(v,w),s} + \overrightarrow{f}^{n}_{(w,v),s}\right) - \\ & \left(\overrightarrow{f}^{n}_{(v,w),t'} - \overrightarrow{f}^{n}_{(w,v),t'} - \overrightarrow{f}^{n}_{(v,w),s} + \overrightarrow{f}^{n}_{(w,v),s}\right) \\ &= \overrightarrow{f}^{n}_{(v,w),t} - \overrightarrow{f}^{n}_{(w,v),t} - \overrightarrow{f}^{n}_{(v,w),t'} + \overrightarrow{f}^{n}_{(w,v),t'} \\ &\le \overrightarrow{f}^{n}_{(v,w),t} + \overrightarrow{f}^{n}_{(w,v),t'} \\ &\le \overrightarrow{x}^{n}_{(v,w)} + \overrightarrow{x}^{n}_{(w,v)} \\ &\overset{**}{\le} x^{n}_{\{v,w\}} \end{aligned}$$

- $$\begin{aligned} d^{n,s}_{(v,w),t} &= \overrightarrow{f}^{n}_{(v,w),t} - \overrightarrow{f}^{n}_{(w,v),t} - \overrightarrow{f}^{n}_{(v,w),s} + \overrightarrow{f}^{n}_{(w,v),s} \\ &\le \overrightarrow{f}^{n}_{(v,w),t} + \overrightarrow{f}^{n}_{(w,v),s} \\ &\le \overrightarrow{x}^{n}_{(v,w)} + \overrightarrow{x}^{n}_{(w,v)} \\ &\overset{**}{\le} x^{n}_{\{v,w\}} \end{aligned}$$

**Using constraints 3.5 which we know to be true for the original solution (f, x).

Then, if we assume that $t, t' \in \Gamma^+(s)$ are the terminals for which the maximums in the definitions of $\overrightarrow{x'}^{n,s}_{(v,w)}$ respectively $\overrightarrow{x'}^{n,s}_{(w,v)}$ are attained, we have

$$\begin{aligned}
\overrightarrow{x'}^{n,s}_{(v,w)} + \overrightarrow{x'}^{n,s}_{(w,v)} &= \overrightarrow{f'}^{n,s}_{(v,w),t} + \overrightarrow{f'}^{n,s}_{(w,v),t'} \\
&= \max(d^{n,s}_{(v,w),t}, 0) + \max(d^{n,s}_{(w,v),t'}, 0) \\
&\overset{\dagger}{=} \max(d^{n,s}_{(v,w),t} + d^{n,s}_{(w,v),t'}, d^{n,s}_{(v,w),t}, d^{n,s}_{(w,v),t'}, 0) \\
&= \max(d^{n,s}_{(v,w),t} - d^{n,s}_{(v,w),t'}, d^{n,s}_{(v,w),t}, d^{n,s}_{(w,v),t'}, 0) \\
&\leq x^n_{\{v,w\}} \;=\; x'^{n,s}_{\{v,w\}}
\end{aligned}$$

This proves that constraints 3.5 hold. To see that the last inequality holds, we use the two observations we made before.

Finally, it remains to show that constraints 3.10 hold for (f', x'). This can be seen as follows. Let $(s, s') \in \mathrm{E}(R^n)$ with $\Gamma^+(s') \neq \emptyset$, $(v, w) \in \overrightarrow{E}$, and $e := \{v, w\} \in E$. We have to show that $\overrightarrow{f'}^{n,s}_{(v,w),s'} + \overrightarrow{x'}^{n,s'}_{(v,w)} \leq x'^{n,s}_e$. To this end, let $t' \in \Gamma^+(s')$ such that $\overrightarrow{x'}^{n,s'}_{(v,w)} = \overrightarrow{f'}^{n,s'}_{(v,w),t'}$. Then, we get

$$\begin{aligned}
\overrightarrow{f'}^{n,s}_{(v,w),s'} + \overrightarrow{x'}^{n,s'}_{(v,w)} &= \overrightarrow{f'}^{n,s}_{(v,w),s'} + \overrightarrow{f'}^{n,s'}_{(v,w),t'} \\
&\leq d^{n,s}_{(v,w),s'} + d^{n,s'}_{(v,w),t'} \\
&= \overrightarrow{f}^{n}_{(v,w),s'} - \overrightarrow{f}^{n}_{(w,v),s'} - \overrightarrow{f}^{n}_{(v,w),s} + \overrightarrow{f}^{n}_{(w,v),s} + \\
&\quad\; \overrightarrow{f}^{n}_{(v,w),t'} - \overrightarrow{f}^{n}_{(w,v),t'} - \overrightarrow{f}^{n}_{(v,w),s'} + \overrightarrow{f}^{n}_{(w,v),s'} \\
&= \overrightarrow{f}^{n}_{(w,v),s} - \overrightarrow{f}^{n}_{(v,w),s} + \overrightarrow{f}^{n}_{(v,w),t'} - \overrightarrow{f}^{n}_{(w,v),t'} \\
&\leq \overrightarrow{f}^{n}_{(w,v),s} + \overrightarrow{f}^{n}_{(v,w),t'} \\
&\leq \overrightarrow{x}^{n}_{(w,v)} + \overrightarrow{x}^{n}_{(v,w)} \\
&\leq x^n_e \;=\; x'^{n,s}_e
\end{aligned}$$

□

Corollary 3.7. For the *mcf* ILP formulation, the choice of the root terminal $t_{\mathrm{root},n}$ for each net n does not affect the value of an optimum LP solution.

Proof. While this has been observed before, e.g. by [GM93], this can be also seen by the proof of theorem 3.6. Let $s, t \in n$ be two terminals for net n. Then, we can use the construction from the proof to transform a solution

[†]Note that $\max(a, 0) + \max(b, 0) = \max(a + b, a, b, 0)$.

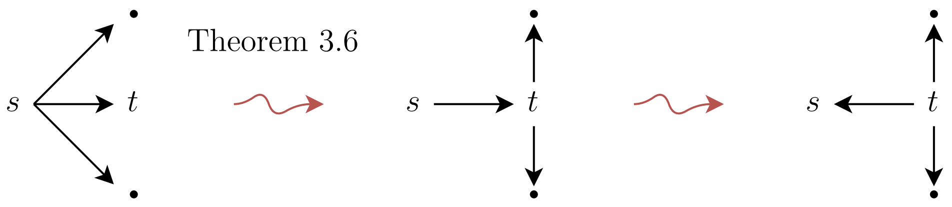


Figure 3.2.4: The construction utilized in the proof of corollary 3.7.

for the *mcf* formulation with root s into a solution for the *multi-root mcf* formulation with arborescence R defined by

$$\mathrm{E}(R) = \{\,(s,t)\,\} \cup \{\,(t,u) \mid u \in n \setminus \{\,s,t\,\}\,\}\,.$$

By reversing the $s-t$ flow in this solution, we obtain a solution for the *mcf* formulation with root t, which has the same objective value as the original *mcf* solution for root s. See fig. 3.2.4 for a visualization of these two steps. To see that the second transformation works, observe that constraints 3.10 in the *multi-root mcf* solution ensure that constraints 3.5 in the derived *mcf* solution hold. □

In contrast, example 5) in fig. 3.2.3 shows that the choice of the arborescence for the *multi-root mcf* has an influence on the integrality gap. There, the chosen arborescence yields a formulation with an LP solution value of 6. But, if the arborescence had been chosen as a star, the *multi-root mcf* formulation would be equivalent to the *mcf* formulation, which has an LP value of 7.5 for this instance.

Example 5) also illustrates that this problem occurs, when flows from different roots interact. In practice, we therefore try to limit the number of non-leaf terminals in the arborescence. This observation and conclusion has been made by Malte Schürks, who also developed the following algorithm for constructing the arborescence R for each net. This algorithm works as follows. The terminals of a net are clustered, mostly based on their position. Additionally, for each cluster, one of its terminals is marked as the cluster-root. The arborescence is then defined on two levels. The algorithm from section 3.2.3.2 is used to define a star on the cluster-roots. This arborescence is then extended by adding the remaining terminals as children to their respective cluster-roots. The number of non-leaf terminals in R is further reduced by forbidding clusters of size two. Picture 2) in Figure 3.2.1 on page 17 shows a possible result of this algorithm. There, three clusters are formed { 🟧, ⬛, 🟥 }, { 🟪, 🟩, 🟦 }, { ■ }, with the first terminal in each set being the cluster-root.

See fig. 3.2.5 for a runtime comparison between the two approaches presented in this and in the previous section 3.2.3.2, whose runtimes are denoted by $t_{\text{clustering topology}}$ and $t_{\text{star topology}}$ respectively.

In conjunction with the routing corridors, the *multi-root mcf* formulation has a smaller solution space than the *mcf* formulation. To test if this is relevant in practice, we ran another experiment. We use BonnCell to create routable placements while using the *mcf* formulation with corridors as routability oracle. After 24 hours, BonnCell found placements for 61 out of 96 cells in our testbed. Then, we test if these 61 placements remain routable when switching to the *multi-root mcf* formulation with corridors and the clustering-based topology generation. 57 or 93% of these 61 placements remained routable. Thanks to the fallback mechanism introduced [Cre19], BonnCell automatically reruns failed routings without the corridor restrictions, so we do not lose any results in practice.

3.3 Motivation: The need for reusable design rule encodings

As described in the introduction, the CLRP is subject to a large number of periodically changing design rules. The way these design rules used to be implemented in the BonnCell code required a substantial programming effort to keep BonnCell in sync with the latest design rules. Significant changes in the design rules, e.g. when changing the technology node, often required months of programming, which limited the availability and consequently the usefulness of BonnCell.

In this section, we will briefly describe the traditional routing model. We illustrate its shortcomings by examining the implementation of a typical design rule. From this we will derive requirements that an improved model should fulfill in order to reduce the programming effort for design rules. We will present our improved routing model in sections 3.4 and 3.5.

3.3.1 Traditional routing model

The traditional BonnCell routing flow is created as follows. Given the design rules, we hand-craft the routing graph $\mathcal{G}$. The vertices of $\mathcal{G}$ are assigned to layers and are embedded into the routing area. Edges can either lie on a layer, i.e. connecting two vertices on the same layer, or interconnect vertices on adjacent layers. Vertices and edges that lie on a layer induce metal shapes at roughly the embedding positions of the respective vertices and edges. Both,

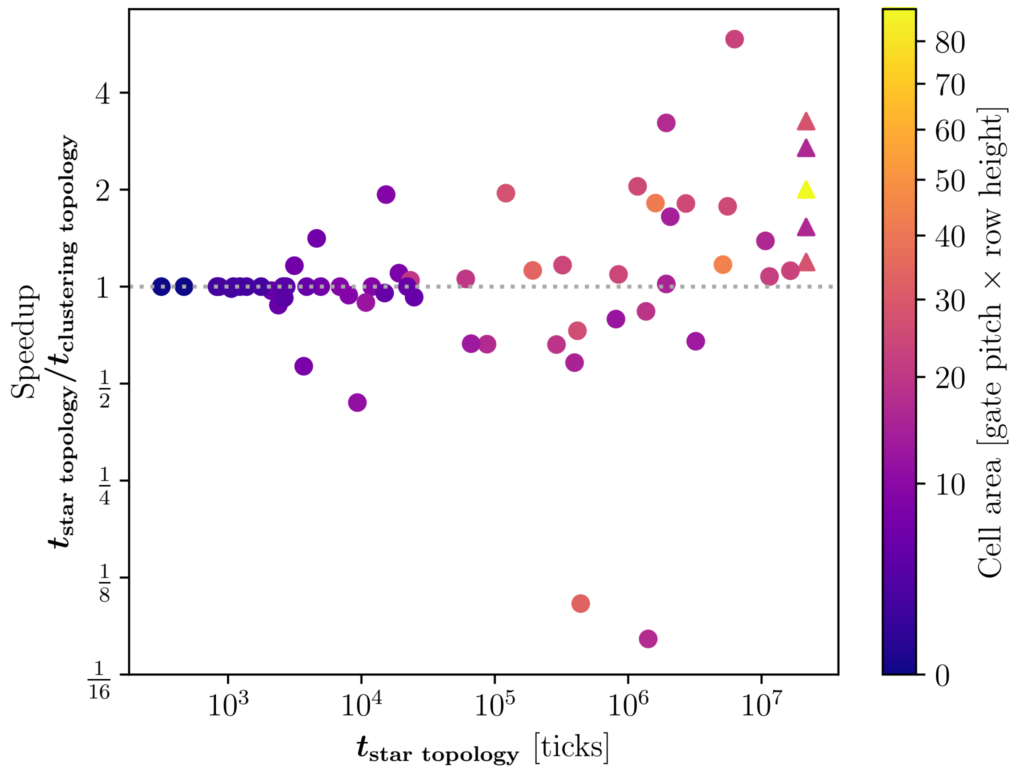


Figure 3.2.5: Comparison between the CPLEX routing runtimes when choosing the topology R for each net either as a star or by the clustering algorithm described in section 3.2.3.3. Using precomputed placements for our testbed of 94 cells, we ran both routing settings with a runtime limit of 24 hours. Those 65 cells for which we were able to compute optimum solutions (within a gap of at most 2%) for at least one run are displayed as markers in the plot. Markers are colored by the cell area. Those cells where the clustering approach found an optimum solution but the star topology approach ran into a timeout are marked as triangles. In this experiment, there were no cells for which the clustering approach ran into a timeout while the star topology approach found a solution. The majority of markers lie above the gray line, i.e. they are solved faster with the clustering based topology than with the star topology. While there are considerable outliers, larger cells tend to have a larger speedup than smaller cells. The median speeup for cells with an area greater than that of a 25 gate pitches wide single row cell is 19%.
All runtimes are reported in deterministic CPLEX ticks (see [22b] for details).

vertices and edges, may be annotated with additional ILP variables that control their metal shape's exact position and size.

Given an instance of the CLRP with nets $\mathcal{N}$, we translate the pins of each net into vertices whose associated metal shapes match the pins' shapes. Using the Steiner tree packing formulation presented in section 3.2.3, we generate an ILP formulation whose solution, disjoint Steiner trees in $\mathcal{G}$, are translated back into routings for $\mathcal{N}$. We add additional constraints to the ILP formulation, to ensure that the net routings obey all design rules.

Figure 2.2.1 on page 6 depicts parts of $\mathcal{G}$ on the left. Some edges are marked as active. They will be translated into the metal shapes depicted on the right.

In the following, we give a typical example of how design rules can be translated into constraints for our ILP formulation. We consider the following two rules that occur (with a bit more complexity) in real technologies:

> *Rule 1:* The maximum length in x-dimension of the shapes on layer B is θ.
>
> *Rule 2:* Shapes on layer B are rectangles whose $x_{\min}$ and $x_{\max}$ positions lie on the red coordinates (see fig. 3.3.1).

We model these constraints as follows. Let us assume that we have defined $\mathcal{G}$ as depicted in fig. 3.3.1, where the positions of the vertices stem from the intersection with vertical tracks on the layer above and below. We enforce the second rule by adding the constraints that the edge usage variables of edge e_{2i-1} and e_{2i} are equal. Then we can implement the first rule by adding constraints that allow at most two of three consecutive odd indexed edges to be used. In figure fig. 3.3.1 this would mean adding the constraint $x_{e_1} + x_{e_3} + x_{e_5} \leq 2$. To see that this is sufficient, note that due to the second rule, if two consecutive odd indexed edges are used, the edge between them must also be used. The first rule is violated exactly if 5 or more consecutive edges are used. Due to rule 2, this happens if and only if 3 or more consecutive odd indexed edges are used.

Formulating rule 1 in this way has a number of shortcomings:

- The formulation is only correct because knowledge about rule 2 has been used. If rule 2 is dropped or the allowed positions $x_{\min}$ or $x_{\max}$ change significantly, the implementation is no longer correct.
- The formulation is highly dependent on the exact graph structure. For example, if edges on layer B need to be subdivided into smaller pieces (e.g. to allow more possible positions to connect to the adjacent layers above and below), we have to adjust the implementation of the rule.

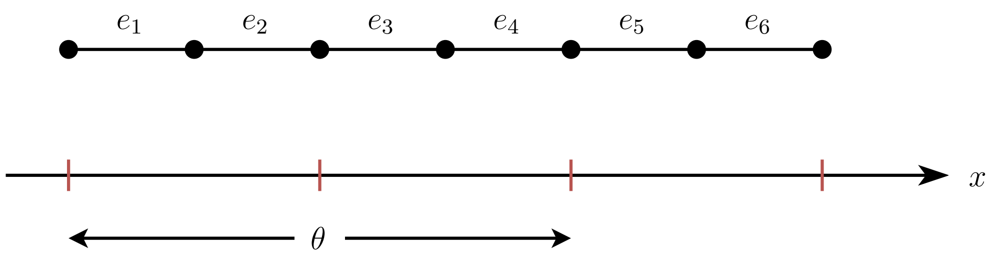


Figure 3.3.1: Displays six edges of the routing graph. Additionally, the x-axis with some coordinates marked in red are shown. Similar to fig. 2.2.2 on page 8, each edge, if chosen for a net, will induce a rectangular metal shape with the same x-coordinates as the edge.

Conversely, if some vertices on layer B are removed and their two adjacent edges on B are joined into one larger edge, the rule formulation has to be changed significantly as well.

- Due to the first two points, it is difficult to reuse the rule formulation for a similar rule on a different layer, or when changing technologies. The formulation is tailored to the specific layer and its surroundings.
- Code that implements this rule formulation might be hard to understand and adapt later on. To see that the code is correct, one has to follow the arguments laid out above.

All of these points contribute to the large programming effort required to keep BonnCell up-to-date with the latest design rules.

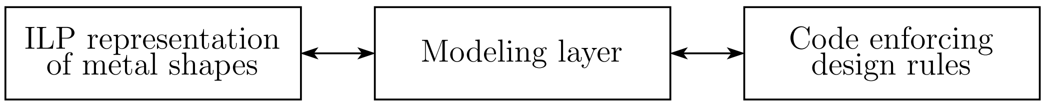
3.3.2 Requirements of the new implementation

Based on these shortcomings, we derive a number of requirements that our new approach should fulfill in order to reduce the programming effort for design rules.

1) If similar rules on different layers or sets of metal shapes are implemented, it should be possible to share most of the code between them.
2) The code shall be easily adaptable to changes in the design rules. This means that small design rule changes should only entail small code changes and that the resulting rule code should be descriptive and easily understandable.
3) The new approach should produce an ILP formulation that can be solved quickly.

It is not immediately obvious how to reconcile the above requirements into one framework. In particular, at first glance, requirements 1) and 3) seem to contradict each other. What metal shapes are allowed by the design rules differs vastly from layer to layer. Since 3) asks us to generate a concise ILP formulation, we most likely need to generate different ILP formulations for different layers. In contrast to that, 1) seems to require that shapes on different layers are represented by a common model.

To resolve this conflict, we propose the introduction of an intermediate modeling layer between the ILP variables and the code that enforces the design rules. This modeling layer hides the exact variable representation of a shape from the code that enforces the design rules. It acts as an interface to the ILP representation of the shapes.

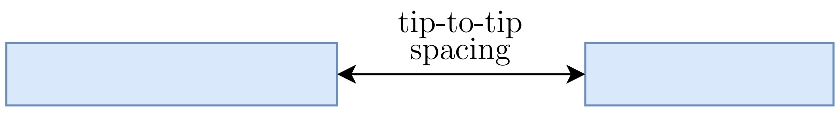

ILP representation of metal shapes
Modeling layer
Code enforcing design rules

In sections 3.4 and 3.5 we propose such a modeling layer. Based on this, in section 3.5.3 we will present a methodology for writing design rules. Together, they form a framework that tries to fulfill the requirements formulated above.

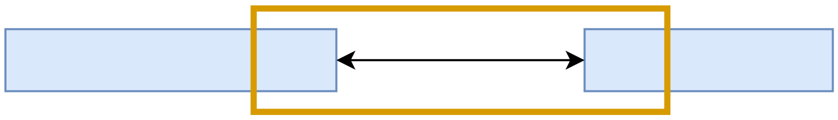
3.3.3 More observations

We make two more observations that are essential for the model we present in the following section.

- Design rules almost never depend on whether the participating shapes belong to the same or different nets. In the following example, the minimum tip-to-tip spacing rule applies, regardless of whether the two participating shapes belong to the same net or not.



- We also observe that almost all design rules depend only on the properties of the participating shapes in a small region. In other words, if a design rule violation occurs, it is often possible to draw a small rectangle, so that only the parts of the shapes that lie inside this small rectangle "create" the error, regardless of how these shapes might be extended beyond the small rectangle. For example, a violation of the tip-to-tip spacing rule can be detected by only knowing the properties of the two blue shapes within the orange rectangle. The rule violation will occur, regardless of how the shapes are extended outside of this orange rectangle.



Both observations have some exceptions, but most rules we have seen adhere to them. In the following, this will allow us to efficiently implement design rules in the MIP, by only defining constraints on local aspects of the shapes.

3.4 Partial Polygons

In this section we introduce a concept we call partial polygons, that will be the main building block of the modeling layer we want to introduce between the ILP and the rule code. A partial polygon will represent a part of a metal shape, encapsulating some ILP variables and assigning a geometric meaning to them.

Definition 3.8. A **partial polygon** ϕ consists of:

- A layer denoted by layer(ϕ)
- A set of additional integer variables $\mathcal{V}(\phi)$. Each $v \in \mathcal{V}(\phi)$ has a feasible range $[v_{min}, v_{max}] \subset \mathbb{Z}$. $\mathcal{V}(\phi)$ may be empty.
- A polygon $p(\phi)$ whose vertices' coordinates are ILP expressions in $\mathcal{V}(\phi)$, i.e. a constant plus a linear combination of variables in $\mathcal{V}(\phi)$.
  We require that for any assignment of values to the variables $\mathcal{V}(\phi)$, the polygon $p(\phi)$ is translated into a rectilinear polygon in $\mathbb{Z}^2$.
- A partition of the edges of $p(\phi)$ into "open" and "closed" edges.

We impose the following restrictions on partial polygons. They are not strictly necessary for our next steps, but will simplify many aspects of the implementation.

- $p(\phi)$ has no holes

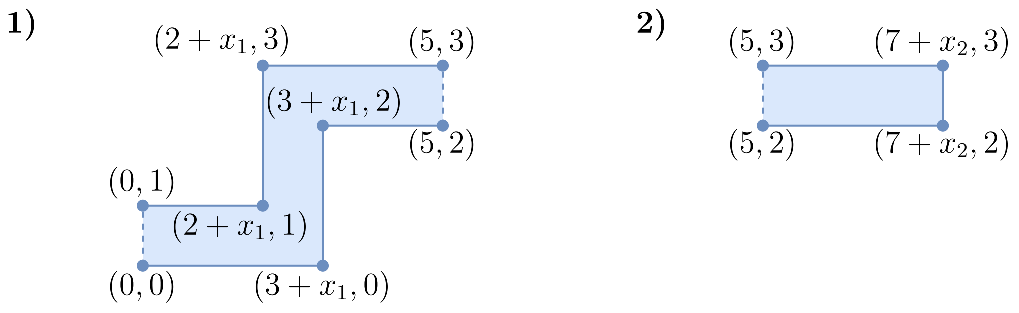


Figure 3.4.1: 1) shows an example of a non-rectangular partial polygon, annotated with its vertices' coordinates. It has one additional variable x_1 and two open edges indicated by dashed lines. Choosing different values for x_1 changes the position of the vertical piece of the wire. Notice that some edges of the polygon "disappear" when choosing $x_1 \geq 2$ or $x_1 \leq -2$. Typically, we prevent this from happening by choosing the feasible range of the variable appropriately or adding further constraints to the ILP. 2) shows a rectangular partial polygon with one open edge on the left. Both partial polygons coincide in exactly one open edge and can thus be merged.

- All edges are non-degenerated, i.e. the edge orientation is the same for all possible variable values and edges cannot have zero length for certain variable values (see fig. 3.4.1).
- Open edges have constant coordinates.

We have marked the edges of partial polygons as either open or closed. An open edge indicates, that the partial polygon will be extended here, while closed edges mean that the partial polygon ends here and cannot be extended. This distinction will be important later on for implementing design rules and will be discussed in detail in remark 3.11 on page 51.

Two partial polygons that lie on the same layer and coincide in exactly one edge that is marked as open in both polygons can be **merged** into one larger partial polygon by gluing the two partial polygons together (see fig. 3.4.1). We do not allow partial polygons to be merged if they intersect in any other way. The merged partial polygon has two open edges less than the sum of open edges of the two initial partial polygons. Since we require open edges to have constant coordinates, we can compute if two partial polygons can merge without knowing the values for their additional variables, i.e. without having obtained a solution for the ILP. Using a quadtree [22c] for the open edges, we can efficiently compute all pairs of partial polygons that can be merged, which will be important for computing the routing graph in the following section.

3.5 A complete routing model

We give a short overview of the parts that make up our novel routing model and afterwards inspect each of those parts in depth (see fig. 3.5.1).

Given an instance to the CLRP, we start by choosing a ground set of partial polygons P based on the problem's design rules and routing area. How to arrive at such a ground set is not a trivial task and will be discussed in section 3.5.1. For now, we assume that we have defined such a ground set, with the property that any feasible routing can be represented by a subset of P.

We then build a routing graph $\mathcal{G}$ as follows. P becomes our vertex set $\mathrm{V}(\mathcal{G})$. Two vertices are connected by an edge, if either:

- they lie on the same layer and their respective partial polygons can be merged, or
- they lie on adjacent layers and intersect[‡] each other, i.e. they are electrically connected (e.g. a via on top of a wire).

The pins of each net will dictate that some partial polygons, respectively some vertices, belong to certain nets. To ensure that the components of a net are connected with each other, we use the Steiner tree packing ILP formulation described in section 3.2.3. We also add the variables of the partial polygons to the ILP.

A solution to the ILP will translate into an CLRP solution as follows: If a vertex is used by a net, we transform the corresponding partial polygon ϕ into a polygon in $\mathbb{Z}^2$, using the values for $\mathcal{V}(\phi)$ from the ILP solution. The set of these metal shapes will make up the net's routing. We will add constraints to the ILP that ensure that the constructed metal shapes obey all design rules, which will be discussed in detail in section 3.5.3.

Note that compared with the approach presented in section 3.3.1, the meaning of the routing graph $\mathcal{G}$ has changed: While previously, active edges and vertices of $\mathcal{G}$ would induce metal shapes, now only vertices of $\mathcal{G}$ induce metal shapes and edges merely model their connectivity.

3.5.1 Defining a ground set of partial polygons

At the core of our approach, we have to define a ground set P of partial polygons. The choice of P dictates what solutions to the CLRP we can find. When defining P, we need to make sure that any routing we want to

[‡]If it is unclear if two shapes intersect without knowing the values of their additional variables, we do the following. We add the edge to $\mathcal{G}$. Additionally, we add constraints to the ILP that enforce the partial polygons to intersect each other if and only if the edge's usage variable equals 1.

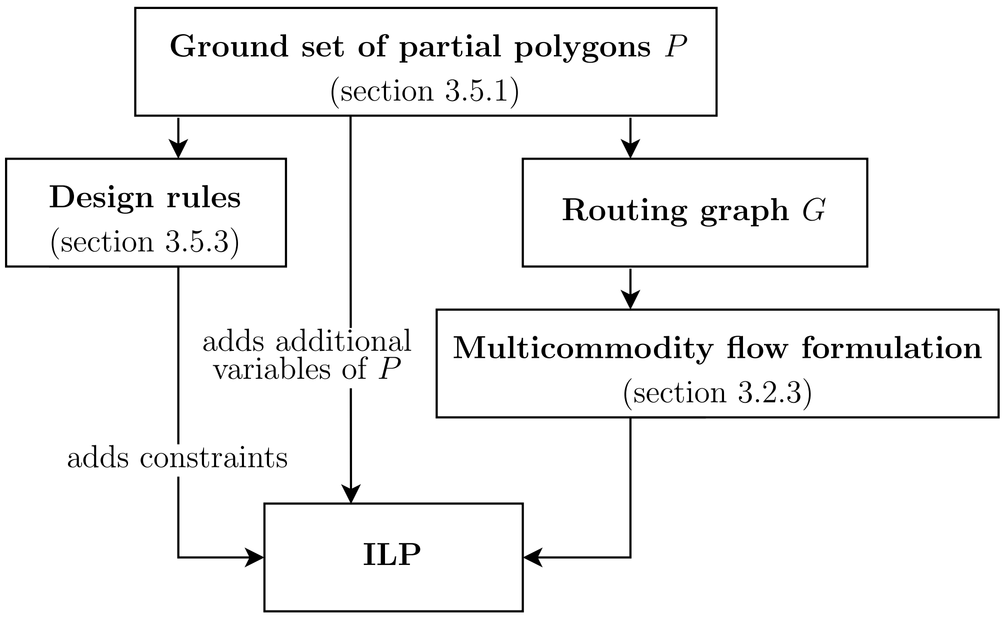

Ground set of partial polygons P
(section 3.5.1)
Design rules
(section 3.5.3)
Routing graph G
adds additional variables of P
Multicommodity flow formulation
(section 3.2.3)
adds constraints
ILP

Figure 3.5.1: The main components of our novel routing model.

be included in our solution space, can be represented by piecing together elements of P. At the same time, we want to ensure that the resulting ILP formulation can be solved quickly. To this end, we need to keep the following aspects in mind while choosing P.

- $|P|+\sum_{\phi \in P} |\mathcal{V}(\phi)|$ should be small, resulting in an ILP formulation with few variables.
- We should avoid symmetry as much as possible. That means that, ideally, we want to define P such that there is a one-to-one correspondence between ILP solutions and CLRP solutions. While this will not be achievable, we still strive to ensure that a single metal shape in any CLRP solution can only be represented by exactly one subset of P.
- As described in section 3.5, the elements of P induce our routing graph. Thus, the choice of P can dramatically influence the size and complexity of the routing graph, which in turn influences the size and complexity of the Steiner tree packing ILP formulation.

In the following, we explore some examples of how P can be defined. They all trade off the above aspects differently, resulting in different ILP formulations and solving runtimes. Note that, due to the chosen separation of concerns (fig. 3.5.1), changing the set P requires few to none modifications to the implementation of the design rules and the implementations of the routing graph and flow formulation.

The choice of P highly depends on the design rules present in a certain technology node. In the following we explore options for P assuming a technology in which metal shapes are rectangular and track pattern based. That means, we assume we are given a track pattern as part of the CLRP, which consists of a set of tracks, each consisting of a layer and a rectangle that spans either the entire width or height of the routing area. In any valid CLRP solution, the metal shapes are either:

- A **wire** metal shape, i.e. a segment of a track, or
- A **via** metal shape, which is a small rectangular metal shape that connects two wires on adjacent layers

The right image of fig. 2.2.1 on page 6 shows an example routing where the green and red metal shapes adhere to a evenly spaced horizontal or vertical track pattern respectively. Some via shapes interconnecting adjacent layers are displayed in white.

Remark. Our approach is versatile and can also model layers that do not (or only partially) adhere to a certain track pattern or even allow non-rectangular metal shapes. Modelling such cases is highly dependent on the specific design rules and there is no general way of modelling a layer that suits all possible cases. Instead, we will describe the common case of track pattern based wire metal shapes connected by via metal shapes in detail here. We hope that the techniques and arguments presented will also be useful for modeling other types of layers.

3.5.1.1 Modeling via metal shapes

Depending on the design rules, the via connection between two tracks on adjacent layers can either have a fixed position, or some room for movement. For each pair of tracks that can be connected with a via metal shape, we add one partial polygon ϕ to P. If the via only has one valid position, we choose the polygon $p(\phi)$ accordingly. If the via has room to move within the intersection of the two tracks, we add two variables x, y to $\mathcal{V}(\phi)$ and choose the polygon $p(\phi)$ as the rectangle $[x, x + w] \times [y, y + h]$. Here, $w, h \in \mathbb{R}$ are appropriate constants and the ranges of x, y are chosen according to the feasible positions of the via.

While the representation of via metal shapes in P is rather straight forward, there are many ways to model the wire metal shapes. In the following, we will present three distinct approaches.

3.5.1.2 Modeling wire metal shapes with disjoint pieces

Let us begin with the first and perhaps most straight forward way to define wire metal shapes in P. It is based on the following two types of design rules that are present in all technology nodes we have witnessed so far:

- Disjoint wire metal shapes on the same track are always required to have a minimum distance $d_{\min}$ between them.
- Each wire has to span a certain minimum length $l_{\min}$ along the track.

For simplicity of notation, let us assume that both values are the same for all tracks and layers and let $\lambda := \min(d_{\min}, l_{\min}) - \epsilon$ [§] .

Our approach for defining the ground set P of partial polygons is as follows. We partition each horizontal track of the given input track pattern into segments of length at most λ. Since all segments of a horizontal track are rectangles with the same y-coordinates, we will omit the y-coordinates in the following notation. For each a segment $[x_{\min}, x_{\max}]$ of the track, we add three partial polygons to P:

- A "left line end" rectangle $[c, x_{\max}]$, where $c \in [x_{\min}, x_{\max} - \epsilon]$ is an additional integer variable and the *right* boundary is marked as open.
- A "right line end" rectangle $[x_{\min}, c]$, where $c \in [x_{\min} + \epsilon, x_{\max}]$ is an additional integer variable and the *left* boundary is marked as open.
- A "through connect" rectangle $[x_{\min}, x_{\max}]$ on which the left and right boundaries are marked as open.

See fig. 3.5.2 for a visualization of these three partial polygons. Analogously, we define partial polygons for vertical tracks.

Proposition 3.9. Assuming the above definition of P, the wire metal shapes of an CLRP solution can always be represented by a unique subset of P. Additionally, for each segment of a track, at most one of the three types of partial polygons associated to it will be active.

Proof. Let m be a wire metal shape in an CLRP solution spanning $[x_{\min}, x_{\max}]$ on a horizontal track (the case for vertical tracks follows analogously). Due to the length $(x_{\max} - x_{\min})$ of m being larger than λ, m intersects at least two segments of its track. For the leftmost openly intersected segment, we choose the left line end rectangle with its variable c set to $x_{\min}$. Similarly, for the rightmost openly intersected segment, we choose the right line end rectangle with its variable c set to $x_{\max}$. And for the other intersected segments, if

[§]Here and in the following, ϵ denotes the technology node specific "base unit". The coordinates of the corners of all shapes of a layout must be integer multiples of this base unit.

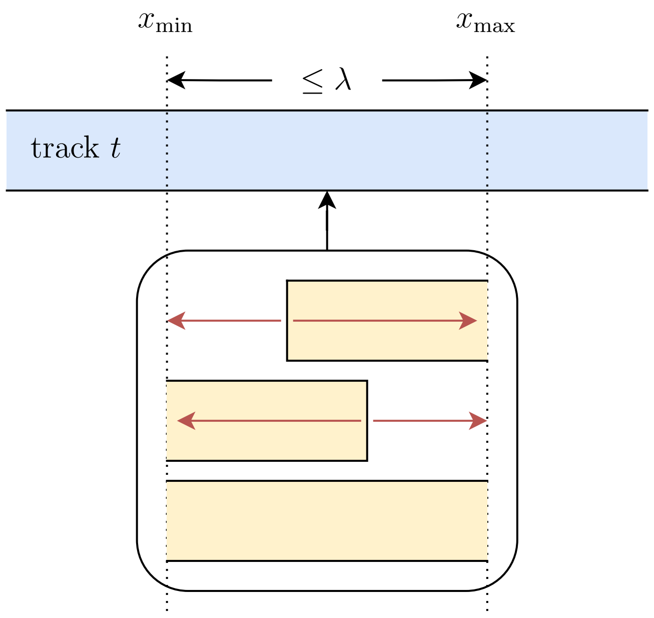


Figure 3.5.2: Depicts the three partial polygons associated with track segment s_i. The red arrows indicate how far the partial polygons can be resized by adjusting their integer variables. Note that we visualize the shapes with some vertical spacing. The partial polygons they represent all have the same y-coordinates as track t.

any, we choose their through connect rectangle. Then, the chosen subset of P, together with the chosen variable values, will exactly represent m.

This construction yields the unique representation of m in our model. This follows from the fact that for each x-coordinate of right or left line end, there is only one "line end" partial polygon that can represent it.

It remains to show, that for multiple wire metal shapes in an CLRP solution, the above construction does neither use a partial polygon twice nor use more than one partial polygon associated with the same track segment. This follows directly from the definition of λ, because two wire metal shapes on the same track need to have a distance of at least λ. In particular, this means that in any CLRP solution, no track segment can intersect more than one metal shape. $\square$

3.5.1.3 Modeling wire metal shapes with interleaved pieces

We will now present another possibility to define wire routing shapes in P. Compared to our previous approach, we will use fewer partial polygons in total. In turn, we will lose the uniqueness property from the previous proof.

Using the previous definition of λ, we construct the partial polygons of a horizontal track as follows. As before, we will only describe the construction for horizontal tracks. To further simplify the notation, we assume that the cell width is divisible by 2λ. We start by partitioning the track into consecutive segments $s_1, \dots, s_k$ of length exactly 2λ. For each such segment $s_i = [x_{\min}, x_{\max}]$, we add the following four partial polygons to our ground set P. The first three are similar to the previous approach.

- A "left line end" rectangle $[c, x_{\max}]$, where $c \in [x_{\min} - \lambda + 1, x_{\max}]$ is an additional integer variable and the *right* boundary is marked as open.
- A "right line end" rectangle $[x_{\min}, c]$, where $c \in [x_{\min}, x_{\max} + \lambda - 1]$ is an additional integer variable and the *left* boundary is marked as open.
- A "through connect" rectangle $[x_{\min}, x_{\max}]$ on which the left and right boundaries are marked as open.
- A "standalone" rectangle $[a, b]$, where $a \in [x_{\min} - \lambda + 1, x_{\max}]$ and $b \in [x_{\min}, x_{\max} + \lambda + 1]$ are additional integer variables and all boundaries are marked as closed.

See fig. 3.5.3 for a visualization of the four partial polygons. Compared to the previous approach, the total number of partial polygons reduces from three per segment of length λ to four per segment of length 2λ.

It is relatively easy to see, that any individual wire metal shape can be represented by some combination of these partial polygons. However, it is much harder to proof that this also holds for multiple wire metal shapes on the same track. In some sense, we have to show that even if there are many small wire metal shapes close together, we have defined "enough" partial polygons to be able to represent all of them.

Proposition 3.10. Assuming the above definition of P, the wire metal shapes M of an CLRP solution can always be represented by a subset of P, so that for every segment of a track, at most one of the four types of partial polygons associated to it will be active.

Proof. We will define a function f that maps a wire metal shape $m = [x_{\min}, x_{\max}] \in M$ and a track segment s_i to either *none* or one of the partial polygons associated with s_i. We set $f(m, s_i) \coloneqq$ *none* if s_i and m do not intersect. Otherwise, we define $f(m, s_i)$ as follows. For $s_i = [x_0^i, x_0^i + 2\lambda]$ we define nine equally-spaced fix x-coordinates $x_{-2}^i, x_{-1}^i, \dots, x_6^i$ as depicted in fig. 3.5.3 and we call the set of those nine variables T_i. We define the interval $X_i \coloneqq [\alpha, \beta]$ where $\alpha \coloneqq \min\{\, t \in T_i \cap m \,\}$ and $\beta \coloneqq \max\{\, t \in T_i \cap m \,\}$. Note that, since $x_{\max} - x_{\min} > \lambda$ and the distance between neighboring elements

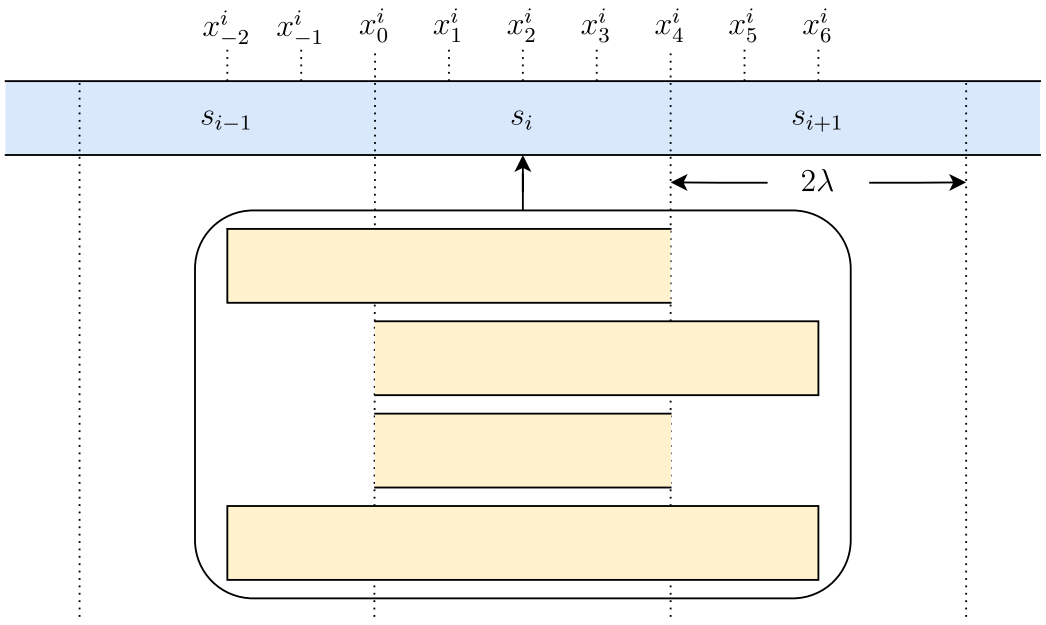


Figure 3.5.3: Shows the four partial polygons associated to track segment s_i. Each partial polygon is depicted with its maximum size.

in T is exactly $\lambda/2$, we have that either $\alpha = \beta \in \{ x^i_{-2}, x^i_6 \}$, or $\alpha < \beta$. We perform a case distinction on α and β to define the value of $f(m, s_i)$.

$$f(m, s_i) := \begin{cases} \text{left line end} & \text{if } \alpha \in \{ x^i_{-1}, \dots, x^i_2 \},\ \beta = x^i_6 & \text{(a)} \\ \text{right line end} & \text{if } \alpha = x^i_{-2} \quad ,\ \beta \in \{ x^i_2, \dots, x^i_5 \} & \text{(b)} \\ \text{standalone} & \text{if } \alpha = x^i_{-1} \quad ,\ \beta \in \{ x^i_2, \dots, x^i_5 \} & \text{(c)} \\ & \text{or if } \alpha \in \{ x^i_0, \dots, x^i_3 \} \ ,\ \beta \in \{ x^i_1, \dots, x^i_5 \} & \text{(d)} \\ \text{through connect} & \text{if } \alpha = x^i_{-2} \quad ,\ \beta = x^i_6 & \text{(e)} \\ \textit{none} & \text{if } \alpha \in \{ x^i_{-2}, x^i_{-1} \} \ ,\ \beta \in \{ x^i_{-2}, \dots, x^i_1 \} & \text{(f)} \\ & \text{or if } \alpha \in \{ x^i_3, \dots, x^i_6 \} \ ,\ \beta = x^i_6 & \text{(g)} \\ & \text{or if } \alpha = x^i_4 \quad ,\ \beta = x^i_5 & \text{(h)} \end{cases} \tag{3.11}$$

We observe that this case distinction is well-defined and covers all possible values of α and β. We also observe that in the first three cases, the additional variables of the partial polygon can be chosen so that the left, the right, or both line ends of m align with that of the partial polygon. An example of how f maps metal shapes to partial polygons can be seen in fig. 3.5.4.

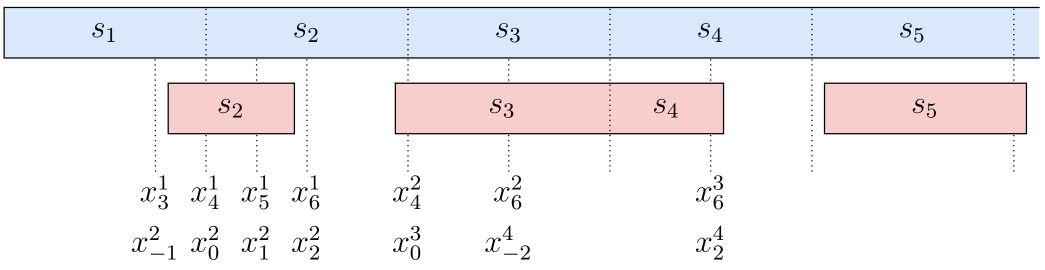

s_1
s_2
s_3
s_4
s_5
s_2
s_3
s_4
s_5
x^1_3 x^1_4 x^1_5 x^1_6
x^2_4 x^2_6 x^3_6
x^2_{-1} x^2_0 x^2_1 x^2_2
x^3_0 x^4_{-2} x^4_2

Figure 3.5.4: Example of how f maps the red wire metal shapes to partial polygons. The picture also indicates some named x-coordinates that play a role in the corresponding case distinctions of f. The left and right wire metal shapes are mapped to standalone partial polygons of s_2 and s_5 respectively. The middle wire metal shape is mapped to both a left and right line end of s_3 and s_4 respectively.

Let

$$R := \bigcup_{m \in M} \left(\bigcup_{\text{track segment } s} f(m, s) \right).$$

We claim that $R \subset P$ represents exactly the metal shapes in M. To proof this, we make the following three claims:

1. For each metal shape $m \in M$, there is one track segment s for which $f(m, s) \neq$ *none*.
2. For each track segment s, there exists at most one $m \in M$ such that $f(m, s) \neq$ *none*. If such an $m \in M$ exists, we call it m_s and define $f(s) := f(m_s, s)$. Otherwise, we set $f(s) :=$ *none*.
3. For two neighboring segments s_i, s_{i+1} the partial polygons $f(s_i)$ and $f(s_{i+1})$ fit together, i.e. $f(s_i)$ has an open right boundary if and only if $f(s_{i+1})$ has an open left boundary. In that case, we have $m_{s_i} = m_{s_{i+1}}$.

Using these three claims, we see that every $m \in M$ will be represented by a non-empty set of partial polygons in R that can be merged to a single closed partial polygon. By construction of the left-, right line end and standalone partial polygons, it follows that the closed partial polygon's line ends have to align with that of m, meaning it represents m exactly. Together with claim 2 that proofs our proposition.

We now show that our three claims indeed hold.

- *Proof 1.* Let $m \in M$ be a metal shape. If there exists a segment s_i such that $s_i \subset m$, for $X_i = [\alpha, \beta]$ we have that $\alpha \in \{ x^i_{-2}, x^i_{-1}, x^i_0 \}$ and $\beta \in \{ x^i_4, x^i_5, x^i_6 \}$. We observe that for these values of α and β,

$f(m, s_i) \neq$ *none*.
Otherwise, m does not span any segment completely. This means that either there exists a segment s_i such that $m \subset s_i$, or that m intersects two neighboring segments s_i and s_{i+1}. In the first case, observe that for $X_i = [\alpha, \beta]$ we have that $\alpha \in \{ x_0^i, x_1^i, x_2^i, x_3^i \}$ and $\beta \in \{ x_1^i, x_2^i, x_3^i, x_4^i \}$, which means that by definition $f(m, s_i) =$ standalone. In the second case, namely when $m \subset s_i \cup s_{i+1}$, for $X_i = [\alpha_i, \beta_i]$ and $X_{i+1} = [\alpha_{i+1}, \beta_{i+1}]$ we observe the following. If $f(m, s_i) \neq$ *none*, our claim is proven. Otherwise, if $f(m, s_i) =$ *none*, one of the cases (3.11.f), (3.11.g), or (3.11.h) has to hold for α_i and β_i. It cannot be (3.11.f), because $\alpha_i > x_0^i$, because by our case distinction, m does not span s_i. If it is (3.11.h), we have $\alpha_{i+1} = x_0^{i+1}$ and $\beta_{i+1} = x_1^{i+1}$, which means that by definition, $f(m, s_{i+1}) =$ standalone. If it is case (3.11.g), we have $\beta_i = x_6^i$. Thus, we have $\beta_{i+1} \geq x_2^{i+1}$ and since $\beta_{i+1} < x_4^{i+1}$ by the assumption that $s_{i+1} \not\subset m$, we get that $f(m, s_{i+1}) \neq$ *none* by the definition of f.

- *Proof 2.* Assume there are two metal shapes $m, m' \in M$ and a segment s_i such that $f(m, s_i) \neq$ *none* $\neq f(m', s_i)$. Let $X_i := [\alpha, \beta]$ and $X_i' := [\alpha', \beta']$ be defined for m and m' as before. Without loss of generality, we assume $\beta < \alpha'$. Due to the minimum distance of m and m' being δ, we get $\beta \leq \alpha' - \delta$. By assumption and definition of f, for both, α, β and α', β', one of the cases (3.11.a), ..., (3.11.e) has to apply. If we assume that $\beta \geq x_2^i$ and thus $\alpha' \geq x_4^i$ then, none of the five cases can apply to α', β', which leads to a contradiction. Thus, we can infer that cases (3.11.a), (3.11.b), (3.11.c), (3.11.e) do not apply to α, β. If instead for α, β case (3.11.d) applies we get that $\alpha = x_0^i$ and $\beta = x_1^i$. Note that since $x_{-1}^i \notin m$, by the minimum length of m we have $x_1^i + \epsilon \in m$ for some small $\epsilon > 0$. This means that $x_3^i \notin m'$ and thus $\beta' \geq x_4^i$, which in turn means, that none of the five cases (3.11.a), ..., (3.11.e) apply to α', β'. Thus, $f(m', s_i) =$ *none*, which is a contradiction.

- *Proof 3.* Let $m \in M$, s_i, s_{i+1} be some neighboring segments intersecting m and $X_i = [\alpha_i, \beta_i], X_{i+1} = [\alpha_{i+1}, \beta_{i+1}]$ as before. We need to show that $f(m, s_i)$ produce and open right boundary, i.e. either a left line end or a through connect partial polygon, if and only if $f(m, s_{i+1})$ produce an open left boundary, i.e. either a right line end or a through connect partial polygon. If $f(m, s_i)$ yields an open right boundary, then $\alpha_i \leq x_2^i$ and $\beta_i = x_6^i$. Thus, $\alpha_{i+1} = x_{-2}^{i+1}$ and $\beta_{i+1} \geq x_2^{i+1}$. By the case distinction of f, $f(m, s_{i+1})$ either yields a right line end or a through connect partial polygon.

Conversely, if $f(m, s_{i+1})$ yields an open left boundary, we have that $\alpha_{i+1} = x_{-2}^{i+1}$ and $\beta_{i+1} \geq x_2^{i+1}$. Consequently, we get $\alpha_i \leq x_2^i$ and $\beta_i = x_6^i$. Hence, by definition, $f(m, s_i)$ yields either a left line end or a through connect partial polygon.

It remains to show, that on the leftmost and rightmost segments s_l and s_r, f does not produce any open right or left boundaries respectively for some metal shape m. This can again be seen by examining the case distinction of f. $f(m, s_l)$ produces an open left boundary, if and only if m continues to the left of s_l. Similarly, $f(m, s_r)$ produces an open right boundary, if and only if m continues to the right of s_r. But, since m cannot extend past the feasible routing area, this cannot happen.

$\square$

3.5.1.4 Modeling wire metal shapes with a few large pieces

We now present one final way of how wire metal shapes can be represented in P. For every horizontal track t (vertical tracks are dealt with analogously), we add $k_t \in \mathbb{N}$ closed partial polygons $\phi_1, , \ldots, \phi_{k_t}$ to P. Each ϕ_i shall be freely movable within track t and k_t is chosen so that in any CLRP solution, at most k_t metal shapes shall lie on t. For example, a valid value for k_t can be computed by dividing the cell's width by 2λ (= minimum length + minimum spacing of metal shapes). In practice however, k_t can usually be chosen much smaller than this, based on additional knowledge about the CLRP.

Using this definition of P, it is obvious that any CLRP solution can be represented by a subset of P, by one-to-one mapping wire metal shapes in the CLRP solution to partial polygons. Typically, this mapping is not unique, because a wire metal shape on track t can be mapped to any of the $\phi_1, , \ldots, \phi_{k_t}$. To remove this redundancy in the model, we can add the following constraints to the ILP:

- Force ϕ_i to lie to the left of ϕ_{i+1}.
- Force ϕ_i to be active if ϕ_{i+1} is active.

Under these constraints the aforementioned mapping of metal shapes to partial polygons in P is unique.

3.5.1.5 Experimental results

To determine which of the previously presented methods of defining wire metal shapes yields the best runtimes in practice, we run some experiments. In the following we will describe our test setup and refer to the method from section 3.5.1.2 as "**disjoint**", from section 3.5.1.3 as "**interleaved**", and from section 3.5.1.4 as "**standalone**". We test all of these methods on a state-of-the-art technology node. It features three metal layers M1, M2, M3, of which M1 and M3 are oriented horizontally (i.e. orthogonally to the gates) and M2 is oriented vertically (i.e. parallel to the gates). See fig. 4.5.1 on page 64 for a visualization of the layers. The different methods are applied to these three M-layers. For the methods *disjoint* and *interleaved*, for each of the three layers λ is chosen according to the layer's design rules. For the *standalone* method, we have to choose how many wire shapes to define for each track on each layer. For this, one could utilize a heuristic based e.g. on cell width, number of nets, and design rules. However, such a heuristic would need to be fine-tuned. Therefore, we use the following heuristic instead. For each cell, we create m_l shapes for every track on layer l, where m_l is the maximum number of shapes on one track on layer l in the routing solution obtained by the *disjoint* method for this cell. For layers M1 and M2 we always create at least one standalone shape per track for technical reasons. Of course, this heuristic can only be used when a routing solution is already known, which limits its usefulness in practice. However, we believe that it is close to what an optimum heuristic for determining the number of wire shapes per track for the *standalone* method can achieve, which makes it well suited to determine the overall performance of the *standalone* method.

BonnCell solves the routing ILP in two very different contexts. Firstly, it is used during BonnCell's routing phase to find an optimal CLRP solution for a previously chosen placement of the transistors. This is done by solving the ILP to optimality using the commercial general-purpose ILP solver CPLEX. Secondly, BonnCell uses the ILP during the placement phase, to determine which placement positions for the transistors are routable. Here, mostly feasibility of the ILP is of interest. To this end, BonnCell automatically transforms the ILP formulation into a SAT formulation, which is then checked for feasibility by a state-of-the-art SAT solver CaDiCaL [BFFH20]. Compared to checking the ILP's feasibility using CPLEX, the SAT based approach is several orders of magnitudes faster. This technique of transforming an ILP into a SAT formulation in the context of cell routing has first been described by [Par+20]. Its refinement and implementation in BonnCell has been described in [Sch23]. Since we use completely different solvers during the placement and routing phases of BonnCell, we report their runtime results individually.

The results for the routing phase are presented as a hat graph [Wit19] in fig. 3.5.5. Out of our testbed of 94 cells, only those 44 cells for which all three runs found an optimium routing solution (within a 2% gap) within 24 hours are displayed. Most green bars lie on the right side of the black bar, meaning that the *disjoint* runs are solved faster than the *interleaved* runs, despite the latter creating fewer partial polygons in total. This might be due to the effect that the *interleaved* partial polygons actually lead to a routing graph with more edges in total. The reason for this can be seen when we imagine that in fig. 3.5.3 on page 37, a via lies at position x_5^i. This via now intersects two partial polygons belonging to segment s_i and additionally all four partial polygons belonging to segment s_{i+1}, inducing a total of six edges between the vias and the partial polygons on the examined layer. In contrast to this, in the *disjoint* method, a via typically only intersects three partial polygons (fig. 3.5.2 on page 35). This effect on the size of the routing graph and the size of the ILP formulation required to model design rules is quantified in fig. 3.5.6.

When comparing the *disjoint* run with the *standalone* run, we observe that the *standalone* run is typically faster, if few standalone partial polygons are created (indicated by the color-coding). For the more complex cells, for which the routing requires more standalone partial polygons per track, the *standalone* method is typically slower than the *disjoint* technique. The runtime effect on these more complex and more difficult to solve instances has a greater impact on the usefulnes of BonnCell. Therefore, we come to the conclusion that the *disjoint* method is suited best for general usage in BonnCell's routing stage.

To get a fair comparison between the SAT solver runtimes during BonnCell's placement phase, we face the following problem. Some placements that are considered unroutable using the *standalone* method might be considered routable using the other two methods. This is because the number of shapes for the standalone method have benn chosen based on the *disjoint* routing for the optimium placement of a cell. For other placements of that cell which BonnCell considers in its search for the optimium placement, more shapes may be needed during routing, resulting in slight differences in the routability assessment of the different methods. This can have large impacts on the placement runtime, because it changes in which order BonnCell's placement engine considers placement candidates. To avoid this, in all runs we sort the nodes in the placement search tree strictly by their objective value (see the "Dijkstra strategy" described in [Klo18]). This way, the first routable placement that is found is automatically the optimium placement, and on this placement the routability assessments of all three methods are identical.

Again, of our testbed of 94 cells, we only show those 57 runs for which all three methods yielded a solution within 24 hours. Note that we do not display the total runtime, but only the time spent in the SAT solver.

For the SAT solver runtime during placement, the overall picture is similar to that of the routing runtimes. Again, in most cases, the *interleaved* method performs worse than the *disjoint* method. For the *standalone* method, those

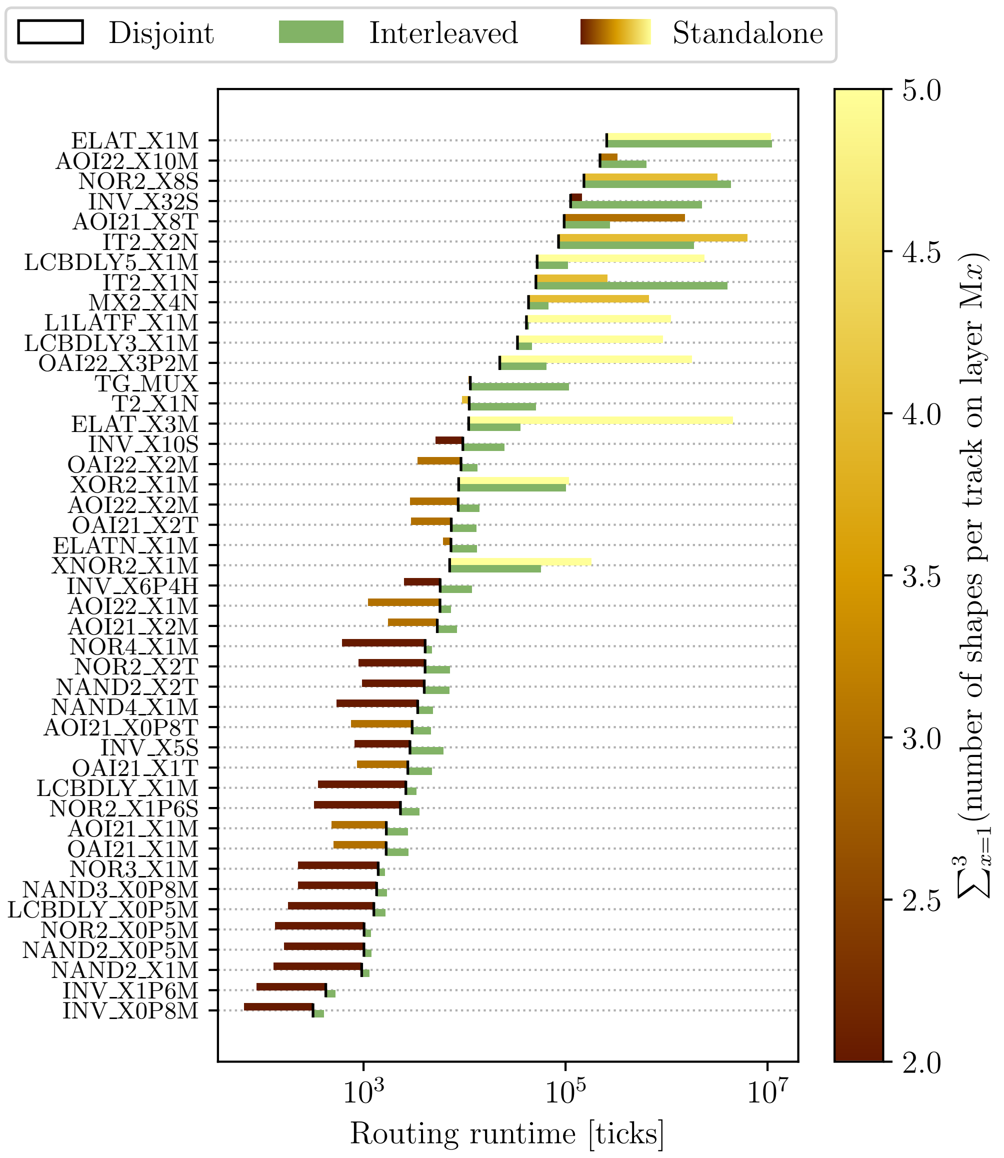


Figure 3.5.5: Hat graph comparison of the routing runtime for the *disjoint*, *interleaved*, and *standalone* runs. The runtimes of each cell for the *disjoint* run are displayed as black vertical lines. Green and brown bars visualize the difference between the runtimes of the *disjoint* run and the *interleaved* respectively *standalone* runs. The later ones are color-coded by the number of standalone partial polygons created per track, summed up over all layers.

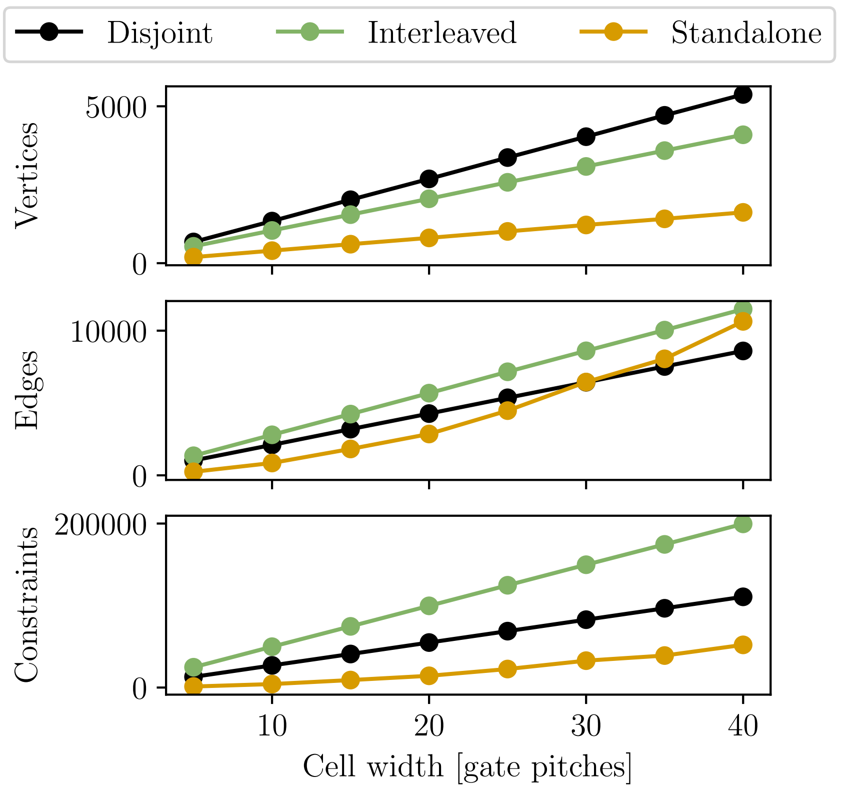


Figure 3.5.6: Visualizes the number of vertices and edges in the routing graph and the number of constraints in the ILP formulation responsible for enforcing design rules. These sizes are shown for different cell widths and for the *disjoint*, *interleaved*, and *standalone* runs. Here, unlike in our previous experiment, the number of shapes in the *standalone* run is based linearly on the cell width. We create $\frac{\text{cell width}}{\text{3 gate pitches}}$ standalone shapes for each track. Since the standalone shapes on horizontal tracks always span the entire cell width, they intersect a number of vias that is also linear in the cell width. Therefore, the total number of edges increases quadratically with the cell width.
The size of the ILP formulation responsible for enforcing design rules mainly depends on how many partial polygons are close together and could potentially create a design rule violation. While there are a bit less partial polygons in the *interleaved* method than in the *disjoint* method, the individual partial polygons are larger and interact with more neighboring partial polygons. Effectively, this almost doubles the number of ILP constraints required to enforce design rules. The same effect is occuring when comparing the number of constraints required for modeling design rules in the *standalone* and in the *disjoint* runs. However, this effect is thwarted by the much smaller number of partial polygons in the *standalone* run, which leads this run to require the smallest number of ILP constraints to enforce design rules.

runs with few partial polygons in total are solved faster than the *disjoint* runs, while those with many partial polygons are typically solved slower. For the same reasons as before, we come to the conclusion that the *disjoint* method is suited best for general use in BonnCell.

3.5.2 Connectivity and shape integrity

In section 3.5 we have defined the routing graph $\mathcal{G}$ based on a chosen ground set of partial polygons P. Each pin of the CLRP will be mapped to a subset of P and thus to a set of vertices in $\mathcal{G}$. Based on this, we will utilize the vertex disjoint Steiner tree packing formulation described in section 3.2.3 to model the pin's connectivity requirements in the ILP. Recall that a partial polygon $\phi \in P$ may have open edges, that indicate that ϕ does not end at this edge but will be extended by merging with another partial polygon. We will add constraints to the ILP that ensure that if such a partial polygon with an open edge is active in the ILP solution, a suitable merging partner will also be active in the ILP solution, and they will both be assigned to the same net. The guearantee that the partial polygons active in an ILP solution will merge together leaving no open edge behind is important for the following method of implementing design rules (also see remark 3.11 on page 51).

3.5.3 Implementing design rules

Recall that our overall goal is to reduce the coding effort required to implement design rules. To this end, we have introduced the concept of maintaining a set of partial polygons that act as our routing model, assigning a geometric meaning to the variables in the ILP. We have explored several ideas on how to define such a set in detail. In the following, we will discuss how design rules can be implemented on the basis of this model. First, in section 3.5.3.1 we give the basic idea of how design rules can be formulated within our model. To further simplify the process of implementing design rules, we introduce another abstraction layer in section 3.5.3.3. It will mimic the structure of design rule manuals, which we will examine in section 3.5.3.2, in order to simplify the translation of design rules from the manual into our framework.

3.5.3.1 Implementing design rules based on partial polygons

In order to ensure that our model only produces solutions that obey all design rules, we need to add further constraints on the ILP. Here, the observations from section 3.3.3 come in handy. Since most design rules are net independent

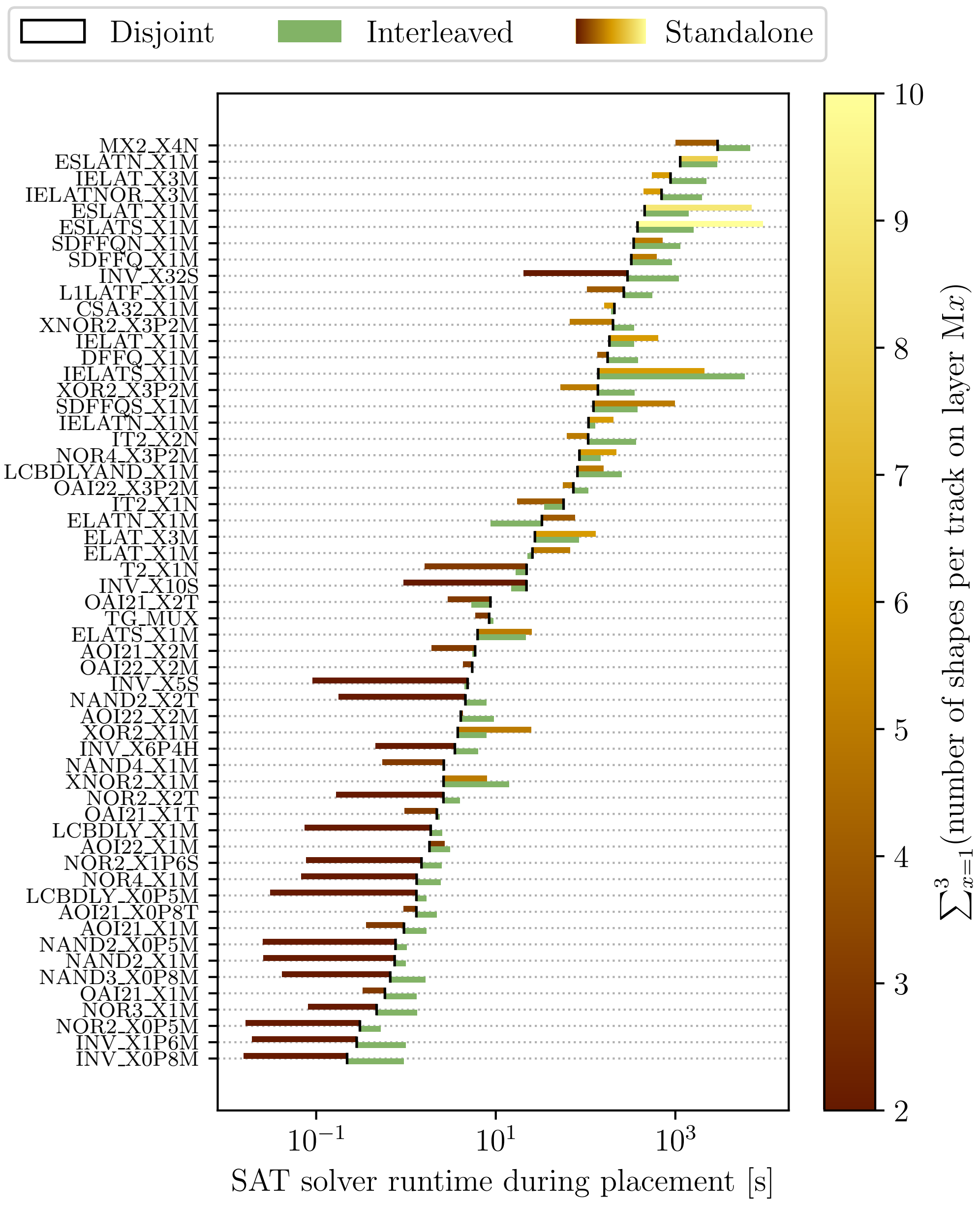


Figure 3.5.7: Comparison between the total SAT solver runtimes during BonnCell's placement phase for the *disjoint*, *interleaved*, and *standalone* methods.

and detectable locally, we can express most of them using constraints of the following form. For partial polygons $\phi_1, \dots, \phi_k \in P$:

$$\phi_1, \dots, \phi_k \text{ are all active} \implies \text{Some ILP constraints have to hold.} \tag{3.12}$$

For example, assuming we have chosen P as described in section 3.5.1.2, the tip-to-tip spacing rule described in section 3.3.3 could be implemented as follows. For every pair of right and left line end partial polygons $\phi_1, \phi_2 \in P$ on the same track that face each other, we add the constraint

$$\phi_1 \text{ and } \phi_2 \text{ active} \implies c_2 - c_1 \geq \lambda \ ,$$

where c_1, c_2 are the partial polygons' additional variables as depicted in fig. 3.5.8 and λ is the minimum required tip-to-tip distance.

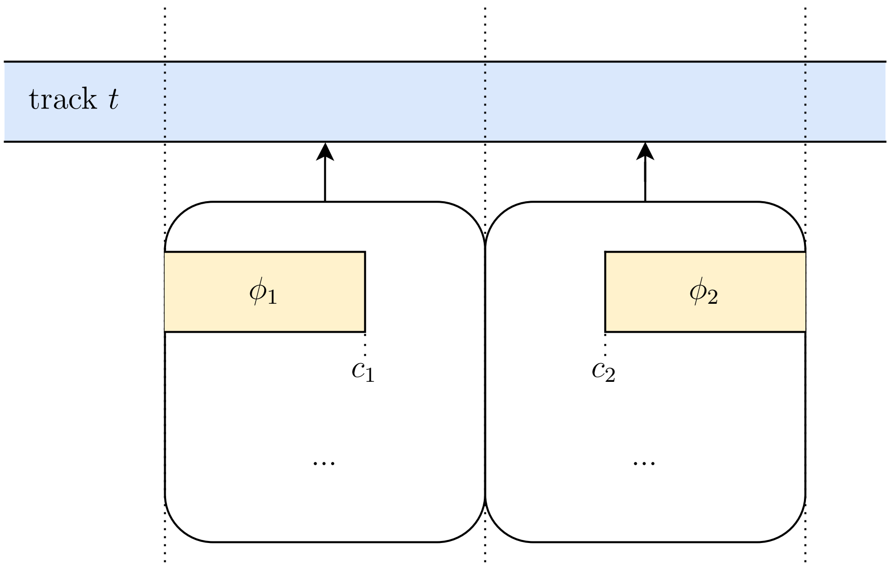


Figure 3.5.8: Example of a subset $\{\phi_1, \phi_2\}$ of P, on which a tip-to-tip spacing rule can be implemented.

Some design rule violations cannot be detected by examining only individual partial polygons. Instead, larger parts of the metal shapes need to be known to detect them. For such design rules, we enumerate merged partial polygons that are comprised of two or more pieces.

3.5.3.2 Examining the design rule syntax

The above method of encoding design rules in the ILP is suitable for most design rules. However, since there are so many design rules and they change periodically, it still requires significant coding effort to implement all design rules in this fashion. We can reduce this by leveraging that many design rule definitions share a similar structure and syntax. By providing a framework that replicates this structure in our code, we can significantly reduce the required coding effort for implementing design rules.

Therefore, let us examine how design rules are defined in detail. Unfortunately, due to confidentiality requirements, we cannot present examples of real world design rules here. Nevertheless, we can still discuss their common structure. We have observed this structure in a number of technology nodes from different companies. Thus, we believe that basing our framework on this structure will not limit our framework's applicability to future technologies.

Typically, design rules are made available in a textual format. For each rule, a short text that adheres to a certain syntax defines restrictions on the metal shapes. The definition of most rules is in the following form:

$$\text{Operator}(\text{Selector}_1, \text{Selector}_2, ...)$$

Each selector describes a set of 2d shapes or edges. This is typically done by starting with all shapes on one layer, and then filtering or modifying them iteratively. For example, a selector might be described as "take the horizontal edges of all shapes on layer L, but not those, that touch shapes on layer M, and then extend those by 5 nanometers". An operator has a fixed arity, meaning it takes a fixed number of selectors, and defines a requirement on them. Examples for operator are:

- Require a minimum L2 spacing, pairwise between all matched elements of Selector_1 and Selector_2.
- Require that all matched shapes in Selector_1 to have a minimum area.
- Forbid that Selector_1 matches anything.

Operators and selectors are often used with only slight modifications in several design rules.

3.5.3.3 A design rule framework

Let us now present our design rule framework. In this framework, for each rule R, two components have to be defined.

- A **query function** q_R, that maps a ground set P of partial polygons to a set of query results $q_R(P) = \{\text{res}_1, \text{res}_2, \dots\}$. Each query result res_i is a set of partial polygons. These partial polygons are not necessarily a subset of P, but can also arise from merging several elements of P together. We denote by usage(res_i) the set of usage variables of the subset of P that gave rise to the set of partial polygons in res_1.

- An **assessment function** f_R, that maps a query result res_i to a constraint disjunction $f_R(\text{res}_i) = \{C_1, C_2 \ldots\}$. As described in section 3.2.2, a constraint disjunction consists of one or more sets of constraints $C_1, C_2 \ldots$, with the semantic that only the constraints of one C_i have to be fulfilled. Note that if one of the C_i is empty, the constraint disjunction is trivially fulfilled.

Given a rule R, we will generate the ILP formulation as follows. For each $\text{res}_i \in q_R(P)$, add constraints

$$(\text{all of usage}(\text{res}_i) \text{ equal } 1) \implies (\text{ constraint disjunction } f_R(\text{res}_i))$$

by using the formulation presented in section 3.2.2. Figure 3.5.9 shows an example of how a design rule can be implemented using this setup of query and assessment functions.

We have chosen this separation of rules into query and assessment functions for the following reasons.

- First and foremost, it resembles the structure of design rule definitions described in section 3.5.3.2, where query functions represent selectors and assessment functions represent operators. This allows us to structure our code similarly to the design manual, which reduces overhead when translating rules from the design manual into code. Also, like the design manual reuses operators and selectors, we will be able to reuse code for query and assessment functions.
- Secondly, this step allows us to separate the code that defines ILP constraints from our ground set of partial polygons P. The assessment functions f_R do not interact with P directly. This allows us to change P without having to modify large parts of the rules implementations. In particularly, since query functions may merge partial polygons from P into larger ones, the sizes of the partial polygons in P are not relevant to the implementations of assessment functions. When implementing an assessment function f_R, one can always choose the corresponding query function such that the partial polygons processed by f_R are large enough so that all potential design rule violations can be found.
  The query functions too can largely be implemented to be independent of P. For example, finding all partial polygons that may intersect some other partial polygon can be implemented in a way that is independent of the choice of P. However, for more complex query functions, this is not true, leaving some dependence between the choice of P and the implementation of the query functions.

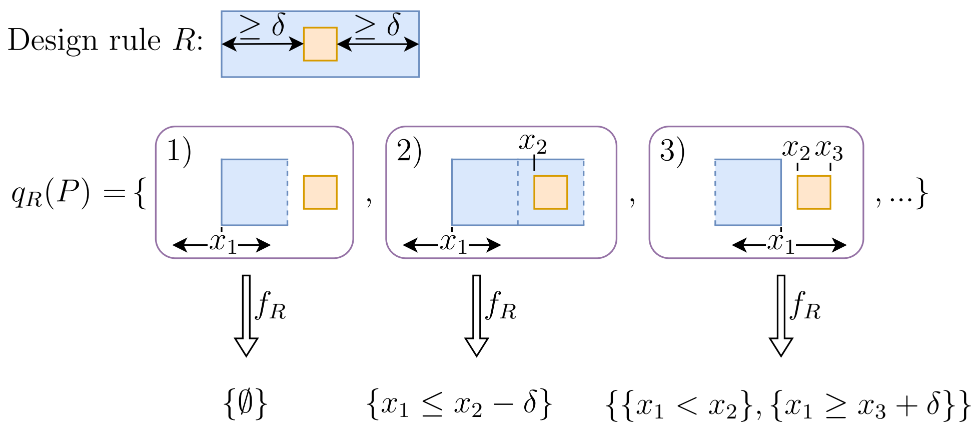


Figure 3.5.9: Visualization of how the design rule R can be implemented using query and assessment functions. Our example design rule R requires a minimum horizontal via overlap of δ between orange vias and blue wires, as depicted in the top row. Of course, this rule only applies for pairs of vias and wires that touch each other. The query function q_R enumerates all possible pairs of via and wire shapes that could violate the rule. Three of these pairs are shown in lilac boxes. For all, we assume that only the depiced x_1 coordinate is non-constant and can be chosen in the range depicted by the arrows. The assessment function f_R is applied to all results of the query function individually. For the pair 1) of partial polygons (left lilac box), the two shapes do not intersect, regardless of the value of x_1. Thus, here the f_R does not impose any constraints on the situation. For situation 2), q_R has merged together two blue partial polygons into one larger on that intersects the orange via. For this situation, f_R adds a single constraint that requires x_1 and x_2 to keep the required distances. Note that no constraint for the right side of the blue partial polygon is introduced, because there the partial polygon has an open edge and f_R does not know how it will be extended there. In situation 3), the blue and orange shapes may or may not intersect, based on the value of x_1. Note that adding the rule R only applies, if they do intersect. To model this, f_R returns a constraint disjunction to model the two cases: Either, the first constraint is fulfilled, which means that the two shapes do not intersect. Or the second constraint is fulfilled, requireing x_1 to lie sufficiently far to the right to obey the via overlap rule.

- Thirdly, with this highly structured definition of design rules, we will be able to easily implement a number of utility features. In section 3.5.3.4, we explain how we can utilize our framework to check existing CLRP solutions for design rule violations to verify the correctness of our implementation. Furthermore, the structured design rule definitions allow us to automatically generate a visualization of the results of all calls to assessment functions, which can be a great help when implementing new design rules.

Remark 3.11. Having presented our novel routing model and its accompanying design rule framework, let us revisit the question of why we introduced the partition of the edges of partial polygons in open and closed in definition 3.8 on page 29. On the one hand, this distinction allows us to easily control which partial polygons can merge. For example, via shapes typically have fixed sizes and the merging of two via shapes should never occur. On the other hand, the distinction into open and closed edges is useful to limit the complexity of the assessment functions. To demonstrate this, we will use the situation displayed in the middle lilac box 2) in fig. 3.5.9. In this situation, we do not need to add constraints to enforce the via overlap to the right of the orange via, because the right boundary of the blue wire partial polygon is open. We know that at this boundary, the blue partial polygon will be extended by some other partial polygon. q_R will enumerate all possible ways to extend the blue partial polygon. If, in one of these extensions, the blue partial polygon has a closed right boundary, f_R will add the appropriate constraints to enforce the via overhang rule, if necessary.

If we had not made this distinction between open and closed edges, the assessment function f_R would have to deal with the uncertainty, if a partial polygon it is presented with is extended at a certain boundary or not. For example, this could be done by requiring f_R to assemble for each generated constraint a list of edges of the participating partial polygons that must not be extended in order for this constraint to be valid. While doing so, ideally the assessment function would not enumerate situations that cannot occur anyway. E.g., in the given example, it would not make sense to consider the case where the orange via shape is extended, because we know from our definition of the set of partial polygons P, that this could not happen.

The definitive distinction of partial polygon edges into closed and open edges avoids this complexity within the assessment function. The necessary case distinctions for whether a shape is extended somewhere or not is done implicitly by the query function, which enumerates exactly those cases that can occur.

3.5.3.4 Implementing design rules is similar to implementing design rule checking

The framework we have defined, in many cases, reduces the task of implementing a design rule to the task of implementing a corresponding assessment function. An assessment function is, in essence, a "geometric" algorithm that operates solely on partial polygons. In fact, it is very similar to a design rule checker, a program that checks fixed metal shapes for design rule violations. While implementing a design rule checker is a challenging task in its own (the complexity depends very much on the design rule in question), one can argue that anybody able to implement such a design rule checker, is, with very little additional knowledge, able to implement a corresponding assessment function. This circumstance demonstrates the significant simplification our framework provides for the task of implementing design rules, in comparison to our initial example in section 3.3.1. The main complexity that still arises now stems from the design rules themselves, rather than from the translation of the design rule into the ILP model.

We will now discuss some differences between implementing an assessment function and implementing a design rule checker. We hope that this can serve as a guideline for anybody who needs to implement an assessment function in the future.

While a design rule checker detects violations on fixed rectilinear polygons, an assessment function has to create constraints for a set of partial polygons. This leads to three main differences between these two types of algorithms.

- Unlike a design rule checker which only has to report if a violation occurs, an assessment function has to produce constraints that prevent the violation from happening. However, in many cases an implementation of a design rule checker naturally leads to a corresponding ILP formulation. For example, if the program code of a design rule checker for the tip-to-tip spacing rule contains the line of code

  $$\textbf{if } c_2 - c_1 < \delta \textbf{ then } \text{report\_violation}();$$

  for some coordinates c_1, c_2 and distance δ, then a corresponding constraint $c_2 - c_1 >= \lambda$ could be easily derived.
- Partial polygons may have non-constant ILP-expressions as coordinates, whose values are only known to lie in a certain range. This can create difficulties when implementing an assessment function, because the relative position of two partial polygons might not be determinable by only examining the coordinate expressions. For example, this situation occurs in fig. 3.5.9 in situation 3) (right lilac box). In such a case,

the assessment function can branch over the different possibilities and utilize the fact that its return type is a constraint disjunction.
Remark. The flexibility of the partial polygons defined in P has an impact on the complexity of the rule constraints. This needs to be taken into account when choosing P. With increasing flexibility of the partial polygons, the rule generating code has to cover more cases, and the resulting ILP formulation becomes more complex.

- Partial polygons may be incomplete (i.e. have open edges). As discussed previously, most design rule violations can be detected locally. Care has to be taken to ensure that the partial polygons presented to the assessment function by the query function are large enough so that all potential design rule violations are detectable. An example of this can be seen in the first two situations examined in fig. 3.5.9. Note that, it is enough to enumerate merged partial polygons that are just large enough to detect the potential design rule violation. In the example of fig. 3.5.9, the number of blue partial polygons that have to be merged together to be able to detect all potential design rule violations depends on the size of the partial polygons and the value δ.

Remark. Many, but not all rules will fit into our framework. For example, some rules are too complex to be representable by a single constraint disjunction, but rather need additional binary variables to be representable concisely. In such a case, adding constraints to the ILP has to be performed "manually" by situation-specific code, outside of the framework. Additionally, design rules might not be representable by linear constraints at all. For example, this can happen when the design rule description utilizes euclidean distances, whose quadratic terms are not representable in linear constraints. In such a case, one can typically either discretize the solution space or approximate the design rule. See [Bib22] for examples of such techniques.

3.5.3.5 Using the framework for design rule checking

We have seen in the previous section, that assessment functions are similar to design rule checking algorithms. In fact, we can use our framework to detect if a given set of metal shapes contains design rule violations. For this, we simply define our ground set P to be the given set of metal shapes we want to check for violations. Then, using our design rule framework, we can generate an ILP formulation that is feasible, if and only if the initial set of metal shapes did not contain any design rule violations. The resulting ILP can be solved quickly without using a fully featured ILP solver. To do this, we simply examine if all the constraint disjunctions created by the assessment functions are feasible.

This is trivial, because all constraints within the constraint disjunctions do not contain any variables, since the ground set P only contains absolute partial polygons. By utilizing the structured definition of design rules, we can even provide detailed feedback on which shapes violate which design rules.

The possibility to check for design rule violations comes in handy for assuring correctness of our code.

- On the one hand, we can verify our code against layouts which are known to be free of design rule violations. This allows us to verify that our formulation is not too strict.
- On the other hand, we can verify that the CLRP solutions we produce are correct and all aspects of our code work together correctly. If, for example, a query function fails to enumerate all necessary situations, this error will be noticed since our framework will occasionally produce solutions that do not pass its own design rule checking.

3.5.4 Runtime comparisons

We have discussed how our novel routing model can significantly reduce the effort required to implement design rules in BonnCell. The resulting new ILP formulation differs significantly from the old formulation that was previously used in BonnCell. Of course, it is important to ensure that the new formulation does not significantly slow down the ILP solving times, because this could negatively affect the usefulness of BonnCell, especially when trying to design large cells. To this end, we compare the runtimes of two versions of BonnCell that were created for the same technology node, one using the old and one using the new routing model. The version using the new routing model was built based on the old version of BonnCell some 1.5 years after the old version had been completed. During the development of the new version, a few more rules were added and some minor errors of the old version of BonnCell were fixed. As a result, the two versions of BonnCell that we are about to compare to each other do not implement the exact same set of design rules. Creating two versions that are perfectly in sync with each other, in terms of design rules, would require a large programming effort, that is unreasonable compared to the insights we would gain from exact runtime comparisons.

Instead, we will compare the runtimes of these two versions of BonnCell that have slight differences in the design rules they implement, to obtain a broad overview of the impact the new routing model has on the runtimes. To minimize the effect that the differences between these two BonnCell versions

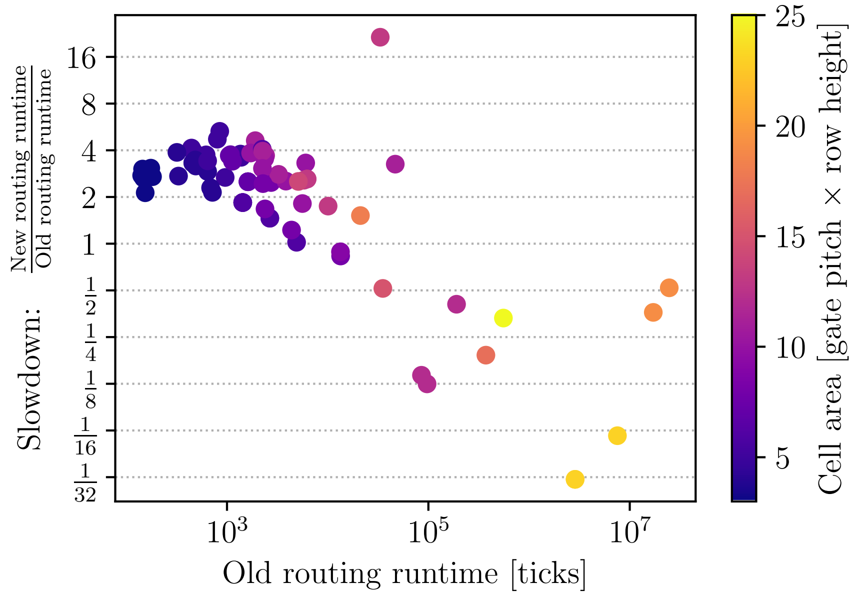


Figure 3.5.10: Comparison of the routing runtimes of two versions of BonnCell. In the **new** version, the novel routing model presented in this chapter is employed, while in the **old** version, the previous routing model approach presented in section 3.3.1 is used. For each cell, the runtimes of both versions are displayed as a single dot. For both runs of a cell, the transistor placement is fixed. Its width is indicated by the color-coding of the dots.

have on runtime, we limit the comparison to cells for which both versions generate the same transistor placements. Additionaly, we only test BonnCell's routing phase, because small differences in the routability of cells can have large impacts on the runtime of the placement phase.

The runtime results are presented in fig. 3.5.10. Small cells are solved consistently slower in the new approach, around a factor of two to four in most cases. However, the runtimes of the larger, more difficult to solve cells actually decreases with our new routing model in most cases. The runtimes of these larger cells are generally more important for the overall usefulness of BonnCell. Therefore, we regard the overall changes in runtime in the new version of BonnCell as beneficial. Although we cannot determine whether changes in runtimes are caused by the differences in the design rules of the two tested versions or by the different ILP formulations, it seems unlikely that the new routing model has a significant negative impact on the ILP solving speed, at least for larger cells.

3.6 Conclusion

In this chapter, we have discussed theoretical and practical aspects of modeling the CLRP as an ILP. We have inspected the *multi-root mcf* formulation that was implemented in BonnCell by Malte Schürks, and we compared its integrality gap to that of the *mcf* formulation.

Our main contribution was the tackling of the large programming effort that stems from maintaining the large set of ever-changing design rules in BonnCell. To this end, we introduced and implemented a novel routing model in BonnCell. This routing model allows the representation of almost arbitrary rectilinear polygon shapes in the ILP through a common building block called partial polygon. We have examined and compared several ways of how a set of partial polygons that covers the entire CLRP solution space can be composed. By unifying the representation of all routing shapes, we were able to introduce a design rule framework that significantly simplifies and structures the task of implementing design rules. In most cases, it reduces this task to the definition of an assessment function, which is only slightly more complex than the definition of a design rule checking function. In the framework, the translation of rules from the design rule manual into code is further streamlined by mimicking the structure of the manual. This ensures high readability and reusability of the resulting code.

The resulting framework has been used to implement a state-of-the-art technology node in BonnCell, which had previously been implemented using the traditional routing model of BonnCell. While difficult to quantify, this practical experience has shown us that, thanks to the new routing model, the programming effort required for modeling shapes and implementing design rules has indeed been greatly reduced. We presented a runtime comparison to the former routing model used in BonnCell, which indicated that on large cells, the new model can be solved even faster than the old one. We conclude that the introduction of the new routing model in BonnCell has been very successful, and we are confident that it will reduce the coding effort to adapt to new technology nodes in the future.

Chapter 4

Pin Accessibility in Cell Layouts

4.1 Introduction

Throughout recent technology developments, the accessibility of pins in cell layouts has become a major challenge. While cell layouts shrink with each new technology node, especially in height, the design rules regarding the metal shapes used for accessing a cell's pins become more and more restrictive and complex. This effectively leads to fewer routing resources being available per pin of a cell layout. The resulting problems in accessing pins and potential mitigation techniques have been studied in the literature mainly with regard to three steps during the chip design process, namely detailed placement (e.g. [Tag+10; DCM17; Tse+19]), detailed routing (e.g. [Ahr+15; KWX20]), and cell layout generation (see section 4.3). We would like to focus on the latter aspect and enable BonnCell to produce cell layouts optimized for pin accessibility. BonnCell's flexible algorithms will enable us to formulate pin accessibility as an objective function, allowing us to generate layouts with the best possible pin accessibility among all layouts. Such optimized layouts can allow for shorter and more efficient wiring between pins. Additionally, it can allow cells to be placed denser together, leading to smaller and more efficient chips.

Optimizing pin accessibility in cell layouts has been the focus of several recent publications. A major problem that all these approaches face is to decide what constitutes "good pin accessibility". Here, one can differentiate between two types of approaches, namely experimental approaches and analytical scoring approaches (see [Tse+19]).

Experimental approaches try to evaluate pin accessibility through a series of routing experiments. In order to compare two layouts for the same cell, they run parts of the routing flow on small artificial chip instances, which utilize many instances of one of the cell layout versions. The cell layout version that leads to the best routing result, e.g. in terms of the number of design rule violations, is chosen as the preferred layout. Examples of this approach are [Tse+19] and [Che+22]. These approaches are able to take into account many effects of the routing flow, some of which might not even be known to the person running the experiments. Thus, running parts of the routing design flow is a very good instrument to assess pin accessibility. The downsides of these experimental approaches are their high complexity, resource intensive computations, and reliance on the availability of other advanced tools, most importantly a router. Especially during early exploration of a new technology, cell layouts often have to be designed before a high quality router for the new technology is available. With regard to BonnCell, it would be almost impossible to embed such an experimental evaluation of pin accessibility into the ILP formulation BonnCell uses to optimize cell layouts.

Consequently, in order to incorporate pin accessibility into BonnCell, we need to consider an alternative approach of measuring pin accessibility. In the literature, a common alternative to experimental approaches are analytical scoring approaches. Here, cell layouts are compared by computing a pin accessibility score for each layout. These scores can quickly be calculated based solely on features detected in the layout and the computation does not rely on external tools.

The remainder of this chapter is structured as follows. In section 4.2, we will introduce some basic notation regarding pin accessibility and review previous work in section 4.3. Based on this, in section 4.4 we develop our own pin accessibility scoring framework, optimized for integration into BonnCell. In section 4.5 we give an in-depth description of our framework in the context of a state-of-the-art technology node and discuss its implementation in BonnCell in sections 4.6 and 4.7. Finally, we present experimental results in section 4.8.

4.2 Notation

We denote the **external nets of a cell** by $\mathcal{N}$. A net is called external if it is part of either the input or the output of a cell. Because cell-internal nets do not need to be accessed from outside the cell itself, they are of little relevance to a cell's pin accessibility and will be ignored in the rest of the chapter. For a given cell layout and a net $n \in \mathcal{N}$, a **pin access point** describes one

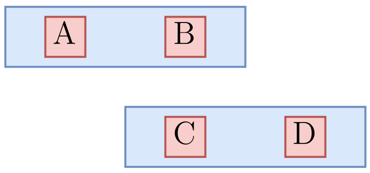


Figure 4.2.1: Two wires belonging to different nets, marked in red with two possible pin access points each. In this example, a pin access point is a valid position to place a via on top of the wire. Assuming certain via distance rules, it may be that pin access points B and C are in conflict with each other and at most one of them can be used. In this case the valid pin access combinations would be $\{ (\mathrm{A}, \mathrm{C}), (\mathrm{A}, \mathrm{D}), (\mathrm{B}, \mathrm{D}) \}$.

possible way of how the existing metal shapes of n can be extended so that n can be accessed either from the ceiling or one of the sides of the cell area. Let $\mathbf{PAP}_n$ denote the set of pin access points of net n. For $k \geq 2$ different nets $n_1, \ldots, n_k$ and $p_1 \in \mathrm{PAP}_{n_1}, \ldots, p_k \in \mathrm{PAP}_{n_k}$, we say $p_1, \ldots, p_k$ are **in conflict** with each other, if adding the shapes of $p_1, \ldots, p_k$ to the layout simultaneously would create a design rule violation. For $\mathcal{N} = \{ n_1, \ldots, n_{|\mathcal{N}|} \}$ we call a tuple $(p_1, \ldots, p_{|\mathcal{N}|}) \in (\mathrm{PAP}_{n_1} \times \cdots \times \mathrm{PAP}_{n_{|\mathcal{N}|}})$ a **valid pin access combination**, if the pins $p_1, \ldots, p_{|\mathcal{N}|}$ are not in conflict with each other. See fig. 4.2.1 for an example. We denote the set of all valid pin access combinations of a layout by **PAC**. Similar definitions have been used by [Xu+15] and [DCM17].

4.3 Previous work

Xu et al. have examined pin accessibility in a number of publications. Most of their work is focused on improving not the cell layout itself, but the cell placement and the detailed router to avoid inaccessible pins. In [XLLP17] they improve the traditional track assignment, a step between global and detailed routing that assigns global wires to a set of local routing tracks, to reduce the number of pins that become inaccessible during the following sequential detailed routing. In [Xu+16] they examine the difficulties of accessing pins that are introduced by restrictive line end rules of self-aligned double patterning techniques. They propose several techniques to avoid blocked pins during sequential detailed routing.

In [Xu+15], Xu et al. propose to maximize the number of valid pin access combinations (which they call hit point combinations) when designing a cell layout in order to optimize pin accessibility. This method rewards spreading the available pin access points among the nets (see fig. 4.3.1 for some exam-

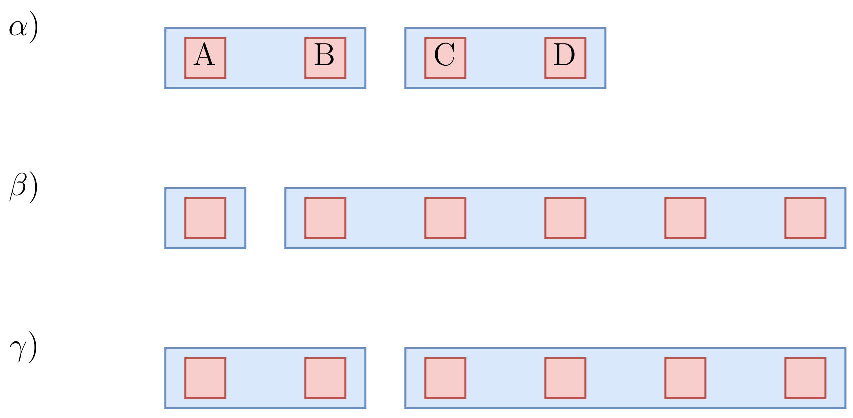


Figure 4.3.1: Comparison of three layouts, each containing one blue metal shape for nets 1 (left) and 2 (right) with some pin access points marked in red. In layout α, two pins from nets 1 and 2 each have two pin access points. The four valid pin access combinations for layout α are: $\{$ (A,C),(A,D),(B,C),(B,D) $\}$. If we only count the total number of pin access combinations when comparing the pin accessibility of two layouts, it can happen that excessively large pins "mask" bottlenecks. For example, by increasing the size of the shape of net 2 as depicted in layout β, we can increase the number of pin access combinations to five. However, this fails to capture the fact that accessing pin 1 becomes more restrictive in comparison to layout α. Nevertheless, measuring |PAC| does somewhat account for this situation, as moving one of the pin access points from net 2 to net 1, as it is done in layout γ, significantly improves the number of valid pin access combinations to a value of eight.

ples). If there are no conflicts between pin access points, maximizing |PAC| is identical to maximizing the product or the geometric mean over the number of pin access points per net. The number of valid pin access combinations can be seen as a variant of the geometric mean over the number of pin access points per net, adjusted for conflicts.

In [SJKS17], Seo et al. measure the remaining pin access (RPA) for each pin p as the expected number of free pin access points for p, once all other pins in the neighborhood of p are connected. They try to ensure that all pins have an RPA value greater or equal than 1, by injecting whitespace into the examined cell layout near the most problematic pins.

In [Tag+10], Taghavi et al. follow a similar approach, penalizing pairs of pins that are close together and are likely to interfere with each other. They add another term to their scoring function that penalizes small pins, because they will be the most difficult to contact. This work is done in the context of a technology node that does not use track patterns for cell layout routing. Therefore, the score they compute is continuous in the pin positions.

4.4 An abstract pin accessibility score

Our goal is to enable BonnCell to generate layouts that optimize some pin accessibility score. To this end, we need to be able to model this score in BonnCell's routing ILP. Unfortunately, the pin accessibility scores introduced in previous works seem to be unsuitable here. The approach by Taghavi et al. does not fit well for the technology node we are considering, because it does not account for discrete track patterns. Both the approaches in [Xu+15] and [SJKS17] base their score on all possible pin access points. This most likely leads to an inefficient ILP formulation, because one needs to add some variables and constraints to the ILP for every possible pin access point of a net, which can be very many on larger cells. Therefore, we will introduce a new scoring function, which is similar to the existing ones, but can be implemented efficiently in the routing ILP. In fact, for our new pin accessibility score it will be sufficient to model only two pin access points per net in the ILP (see section 4.7.1).

Whenever we add a feature to BonnCell, we try to keep it as independent of the technology node as possible to reduce the amount of work required when transitioning to new technology nodes. However, pin accessibility is inherently highly dependent on the technology because it incorporates technology-specific details such as the available layer stack, track patterns, and design rules. Therefore, in addition to defining a particular pin accessibility scoring function for a current technology node, we will provide a simple and generic framework that we hope will act as a guide in the definition of sensible scoring functions for future technology nodes. We will establish this framework and its motivation now and discuss its concrete, technology-dependent implementation in section 4.5.

A cell layout will be instantiated many times on the final chip. Each of these instantiations will have a local environment of other cells, routes, and blockages that will, most likely, cause some pin access combinations of the layout to become undesirable or even infeasible. E.g. blockages or other routes may not leave enough space for the shapes required by a pin access point. To quantify this, we assign to each instantiation i of the cell layout a rating function $f_i \colon \mathrm{PAC} \to \{0, 1\}$, which equals 1 for exactly those pin access combinations that are suitable for the given instantiation i of the layout. Let us group the instantiations of the layout by this rating function. We call such a group **context** and denote the set of all contexts by $\mathcal{C}$. For each context

$c \in \mathcal{C}$, its corresponding rating function is denoted by f_c.

Ideally, we would like to design a cell layout such that for all contexts $c \in \mathcal{C}$, there exists at least one $p \in \mathrm{PAC}$ such that $f_c(p) = 1$. This would mean, that, in every context, the pins of our cell are all accessible. In practice however, most cells will not reach this ideal, mainly due to the following two reasons. Firstly, while designing a cell, a number of different objectives have to be balanced simultaneously, e.g. size of the layout, timing properties, the amount of occupied routing resources, and pin accessibility. Optimality can usually not be achieved simultaneously for all objectives, leading in practice to suboptimal pin accessibility for many cells. Secondly, different contexts will most likely make requirements that cannot be fulfilled at the same time. For example, one context could require net A to be accessed on track t, while another context requires net B to be accessed on track t, but due to spacing constraints, track t cannot harbor pin access points for both net A and B. Such problems can be mitigated by creating different layouts for different contexts, as is done in [KJK21]. While this can improve pin accessibility, working with multiple layouts for the same cell introduces a significant complexity for the whole chip design flow. Notably, timing estimations early on in the flow become less reliable because it is uncertain which cell layouts will eventually be used. As such a feature would require adding support throughout the chip design flow, it is beyond the scope of the BonnCell project, and we will not pursue this idea further in this work.

We are thus striving to generate a single layout for a cell that balances pin accessibility in all contexts. To determine which layout performs best in this regard, we propose the following metric. Let $\mathrm{s} : \mathcal{C} \to \mathbb{R}_{\geq 0}$ denote for every context how important it is to have good pin accessibility in this context. We then define the pin accessibility score of a layout as

$$\mathrm{PinAccessibilityScore} := \sum_{c \in \mathcal{C}} \mathrm{s}(c) \left(\max_{p \in \mathrm{PAC}} \mathrm{f}_c(p) \right)$$

The definitions in this section have been kept highly abstract. Before the pin accessibility score can actually be calculated for the layout of a cell in a certain technology, choices and simplifications have to be made for the different parts of the formula. These include the following:

- What constitutes a pin access point?
- As cell layouts are usually designed early on in the development of a chip, their exact instantiations are unknown and the set of contexts $\mathcal{C}$ must be estimated or guessed.
- How to choose the context rating function s?

These choices are highly technology-dependent, and we cannot provide exact formulas that work for all environments. However, we hope that the above definition of a pin accessibility score can be used as a guideline when optimizing pin accessibility in future technologies.

Recall that our goal was to define a pin accessibility score that can easily be incorporated into BonnCell's routing ILP. Let us assume that for each context $c \in \mathcal{C}$ we can define an ILP that sets a binary variable m_c to 1 if and only if a valid pin access combination under the restrictions of context c exists. Then, the defined pin access score can be expressed as a linear combination in these m_c variables, meaning it can be chosen as an objective function for the ILP. Through the choice and especially the size of $\mathcal{C}$, we can control the complexity and size of this ILP modelling approach. In the following section, we discuss the ILP realization of our scoring function in detail, demonstrating that our pin accessibility score can indeed be modelled and optimized using the outlined approach.

4.5 Pin accessibility in a modern technology node

In this section, we give an in-depth example of how our abstract pin accessibility score can be implemented for a state-of-the-art technology node. Cell layouts in this node consist of a front end of line (FEOL) and, on top of that, a back end of line (BEOL). Both of these consist of a number of layers. The transistors are created in the FEOL and are then interconnected in the BEOL. For a cell layout, three metal layers M1, M2, M3 and their interconnecting vias called V1 and V2 are available in the BEOL. All metal layers are based on track patterns and have a fixed orientation, i.e. wires on M1 and M3 can only be placed horizontally, while M2 wires must be oriented vertically (see fig. 4.5.1 for a visualization). Typically, cell layouts use up most of their M1 routing resources but leave some resources free on M2 and M3. These free routing resources on M2 and M3 can then be used to create space and timing efficient connections between close-by cells.

We will denote a pin access point by a tuple (r, t), where r is one of the circuit rows of the layout and t is one of the M2 tracks. We say such a tuple (r, t) is a pin access point of net n, if one of the following two options holds:

- Either the routing of net n already contains a piece of M2 wire on the track t in the circuit row r,
- or the track t is not occupied by any other net in the circuit row r,

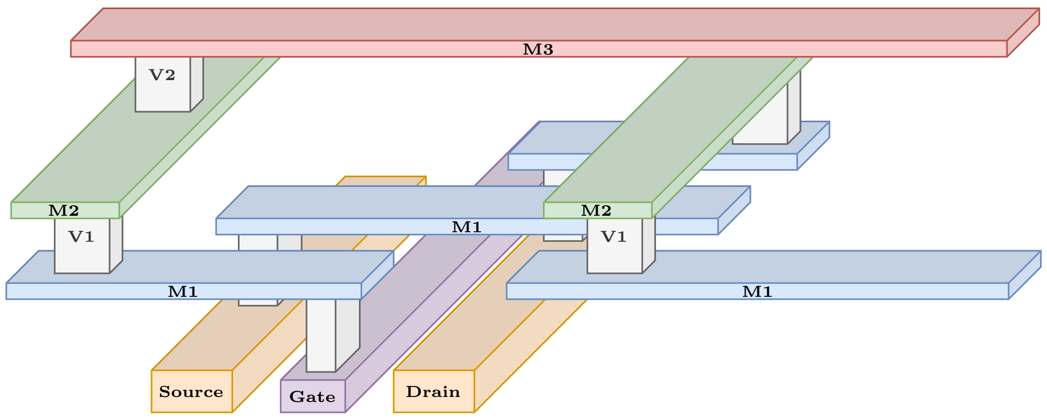


Figure 4.5.1: Visualization of the available layers in the technology node we are considering. All M-layers are track-pattern based. M1 and M3 are oriented horizontally, while M2 is oriented vertically.

and the routing of net n can be extended by one V1 shape and one M2 metal shape within that track and bit, so that no design rules are violated.

To simplify our definition, we require that in a valid pin access combination, no two nets share the same pin access point (r, t).

The rationale behind our pin access point definition is as follows. We expect the detailed router*, which will eventually create the metal shape connections to our cells, to not be able to connect to the FEOL layers directly, nor to modify or extend M1 shapes, due to complicated design rules on these layers. Therefore, the router is left with the possibility to access pins with metal shapes above layer M1. Since most cell layouts in this technology use none or few M3 routing resources, we expect M2 pins to be always accessible from above with M3. Therefore, our focus lies on ensuring that every cell routing leaves enough space for one M2 metal shape per pin.

Choosing a pin access point for each net means to perform a matching between nets and pairs of circuit rows and M2 tracks. To simplify the notation in the rest of the chapter, we typically leave out the circuit rows and pretend that this matching is only between nets and M2 tracks.

Remark. Our above definition of pin access points does not account for the possibility to connect to nets by extending metal shapes on layer M3 or accessing those shapes from the layer above M3. In fact, we believe that all external nets in our cell that utilize layer M3 for its routing are relatively

*At the time of writing, a detailed router for the examined technology node is not yet available at our industry partner.

easy to access. We chose to not explicitly model potential pin access points on M3 for the following two reasons.

- Firstly, we want to avoid that layouts optimized for pin accessibility utilize M3 solely to increase the pin accessibility score. Instead, we aim to keep the amount of occupied M3 routing resources small to allow flexibility in accessing pins located on layer M2.
- Secondly, a net that utilizes metal shapes on M3 for its in-cell routing already occupies two M2 tracks, which are typically some distance apart. In many cases, for the pin accessibility score we are about to define, this is enough to ensure that the pin access of this net is not the bottleneck, meaning that a consideration of potential further pin access points on M3 is superfluous.

We can therefore ignore pin access points on M3, which also helps to keep our pin access score formulation simple.

Having chosen what constitutes a pin access point, the next step proposed by our framework is to define the set of contexts $\mathcal{C}$ that we want to differentiate. For the examined technology, we propose the following definition:

$$\mathcal{C} := \{\text{ “no restrictions” }\} \cup \bigcup_{\text{M2 track } t^{**}} \{\text{ “}t\text{ is blocked” }\}$$

With this, we want to capture the following aspects of detailed routing. If a cell layout is only used in uncongested areas of the chips, the “no restrictions” context would suffice to ensure that all pins can be accessed. However, if the layout is used in a congested area where many nets need to be routed, the detailed router might be forced to occupy one or more of the free M2 tracks of the cell layout to efficiently route some nets unrelated to our cell. We account for this scenario by including the cases where a single M2 track is occupied in our set of contexts. Of course, here one could also include more complex blockage patterns, where two or more M2 tracks are already occupied. We decided to limit the complexity of $\mathcal{C}$ to simplify the implementation.

We choose the feasibility of a pin access combination p in the context c as:

$$\mathrm{f}_c(p) := \begin{cases} 0 & \text{if } c = \text{“}t\text{ is blocked”, and a pin access point on M2} \\ & \text{track } t \text{ is utilized by } p \\ 1 & \text{otherwise, i.e. } p \text{ is compatible with } c \end{cases}$$

**Here and in the following, the set of M2 tracks are always those M2 tracks intersecting the cell area.

Finally, we define the importance of the contexts as:

$$\begin{aligned} \mathrm{s}(\text{“no restrictions”}) &:= 1 \\ \mathrm{s}(\text{“}t\text{ is blocked”}) &:= \frac{1}{\text{Number of M2 tracks}} \end{aligned}$$

Recall our definition from page 62:

$$\text{PinAccessibilityScore} := \sum_{c \in \mathcal{C}} \mathrm{s}(c) \left(\max_{p \in \mathrm{PAC}} \mathrm{f}_c(p) \right)$$

Using this, we see that, for any cell layout, our chosen definitions lead to a pin accessibility score between 0 and 2. It is 0 if connecting all pins of a cell simultaneously is not possible. If the score is 1, all pins of the cell are accessible, but blocking any one of the M2 tracks will make accessing all pins simultaneously impossible. A score of 2 means that the pins of the cell layout remain accessible, even if an arbitrary M2 track is blocked.

Note that with the proposed scoring function, the score for a cell layout can be improved by simply increasing the cell width. Therefore, scores should only be compared between layouts for the same cell that have the same size. In practice, this poses no problem because typically optimizing cell width is more important than optimizing pin accessibility.

Having chosen our scoring function, we can now compare layouts with regard to their pin accessibility. In the next section, we briefly explain how we perform the actual computation. We will discuss how we can optimize a layout to achieve the best possible pin accessibility score in section 4.7.

4.6 Computing the pin accessibility score

As part of the BonnCell tools suite, we provide a program to compute the pin accessibility score defined in the previous section for a given cell layout. The computation is done by solving a variant of the bipartite matching problem for each of the contexts in $\mathcal{C}$, in which we match nets to pin access points. More precisely, we solve the Bipartite Matching With Edge Conflicts Problem, which is defined as follows.

Bipartite Matching With Edge Conflicts Problem (BMWECP)

Instance: A bipartite graph $G = (V, E)$, $V = A \dot{\cup} B$, and a subset of edge conflicts $C \subset \mathcal{P}(E)$

Task: Find a bipartite matching $M \subset E$ of maximum cardinality, such that for all conflicts $c \in C$ we have $c \not\subset M$.

To compute if, with the restrictions imposed by a context $c \in \mathcal{C}$, we can still connect all pins, we define the following BMWECP instance. We choose A as the set of nets and B as the set of pin access points valid under the restriction c. We create edges in E between a net and its pin access points. If two or more pin access points are in conflict with each other, either because they lie on the same M2 track or due to design rules, such as minimum via spacing rules (recall the example from fig. 4.2.1 on page 59), we add those conflicting pin access points as a set to C. We then check the BMWECP instance for feasibility by a straightforward SAT formulation which we solve with a state-of-the-art SAT solver, e.g. CaDiCaL [BFFH20]. While we will prove that the BMWECP is difficult to solve in general, its SAT formulation can be solved within microseconds for all practical instances we have observed. Since there are no guarantees that all instances can be solved this quickly, in practice, a runtime limit of ten seconds for the SAT solver guards against outliers.

Theorem 4.1. Solving the BMWECP is NP-hard, even when restricted to instances where all conflicts $c \in C$ are of size two and each edge $e \in E$ only participates in at most one conflict.

Proof. We will use a reduction from the SAT problem. Given a SAT instance with variables X and clauses $C \subset \mathcal{P}(X \cup \{\, \bar{x} \mid x \in X \,\})$, we will construct an instance of the BMWECP with the described restrictions, which will have a solution of size $|A|$ if and only if the original SAT instance is feasible. We can assume that no clause is empty, and for any variable $x \in X$, not both x and its negation $\bar{x}$ appear in the same clause. The construction works as follows.

1. For each clause $c \in C$, we add one vertex a_c to the set A.
2. For each clause $c \in C$, for each of its literals $l \in c$, we add one vertex $b_{c,l}$ to the set B, and add the edge $e_{c,l} \coloneqq \{\, a_c, b_{c,l} \,\}$ to E.
3. For a variable $x \in X$, let

$$\mathrm{N}(x) \coloneqq \{\, c \in C \mid x \in c \text{ or } \bar{x} \in c \,\} = \{\, c_{x,1}, \dots, c_{x,|\mathrm{N}(x)|} \,\}$$

   denote the clauses in which either x or $\bar{x}$ occur, for which we fix an arbitrarily chosen order. Then, we add a circle of length $2\,|\mathrm{N}(x)|$ to the matching instance by adding
   - vertices $a_{x,1}, \dots, a_{x,|\mathrm{N}(x)|}$ to A,
   - vertices $b_{x,1}, \dots, b_{x,|\mathrm{N}(x)|}$ to B, and
   - edges $\{\, a_{x,1}, b_{x,1} \,\}, \{\, b_{x,1}, a_{x,2} \,\}, \dots, \{\, b_{x,|\mathrm{N}(x)|}, a_{x,1} \,\}$ to E which we name $e_{x,1}, \dots, e_{x,2|\mathrm{N}(x)|}$ as depicted in fig. 4.6.1.

4. Finally, for each clause $c \in C$ and literal $l \in c$, let $x \in X$ be the variable corresponding to l and let $i \in \{1, \ldots, |N(x)|\}$ such that $c = c_{x,i} \in \mathrm{N}(x)$: Then, we add the following conflict to C.
   - $\{e_{c,l}, e_{x,2i}\}$ if $l = \bar{x}$, or
   - $\{e_{c,l}, e_{x,(2i-1)}\}$ if $l = x$.

Figure 4.6.1 on page 68 visualizes the above construction.

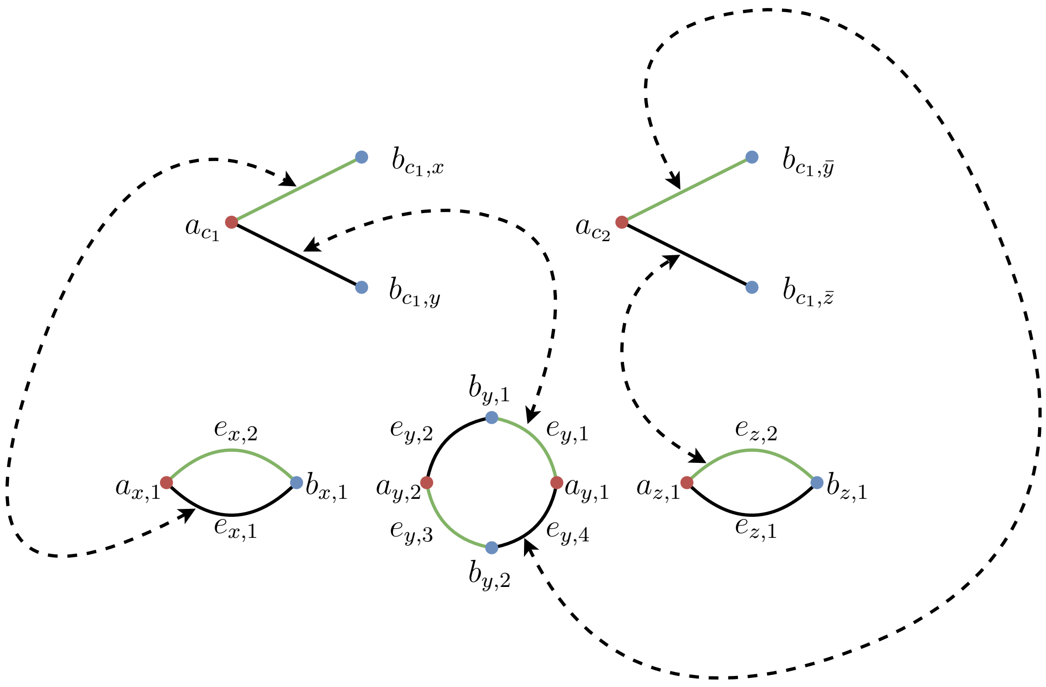


Figure 4.6.1: Visualization of the BMWECP instance construction in the proof of theorem 4.1 for a SAT instance with variables $X = \{x, y, z\}$ and two clauses $c_1 = (x \vee y)$ and $c_2 = (\bar{y} \vee \bar{z})$. Red vertices belong to set A and blue vertices to set B. Dashed arrows indicate pairs of conflicting edges. The green edges correspond to the feasible truth assignment $x =$ true, $y =$ false, and $z =$ true.

We observe that the size of the constructed BMWECP instance is polynomial in the size of the initial SAT instance. Furthermore, all additional restrictions imposed in the theorem statement are also fulfilled. It remains to show that a solution to the SAT instance exists, if and only if an optimum solution for the BMWECP instance of size $|A|$ exists.

First, let us prove that a solution $S \subset E$ of size $|A|$ for the BMWECP instance yields a solution for the SAT instance. We observe that in such a solution of maximum size, for each circle constructed for variable $x \in X$ in step 3 of the construction, S contains the edges $e_{x,i}$ for either all even or all odd $i \in \{ 1, 2, \ldots, |\mathrm{N}(x)| \}$. We construct a truth assignment for the variables of the SAT instance by setting $x \in X$ to true, if S contains all even indexed $e_{x,i}$, and false otherwise.

We claim that this truth assignment is a solution to the SAT instance. To see this, we first observe that, due to the size of S, for each clause $c \in C$, there exists exactly one $l \in c$, such that $e_{c,l} \in S$. If $l = \bar{x}$ for some $x \in X$, notice that $e_{c,l}$ is in conflict with $e_{x,i}$ for some even index i. Thus, in this case we have assigned $x =$ false in our truth assignment construction above. Therefore, our truth assignment fulfills clause c. Using the analogous argument for the case of $l = x \in X$, we see that our truth assignment indeed fulfills all clauses in C, thus constituting a solution to the SAT instance.

Now, it remains to show that, given a truth assignment for X that satisfies all clauses C, we can construct a solution for the BMWECP instance of size $|A|$. We will now construct such a solution S by reversing the construction above. For each variable $x \in X$ we add the edges $e_{x,i}$ to S for all even $i \in \{ 1, \ldots, 2\,|\mathrm{N}(x)| \}$ if $x =$ true, or for all odd i if $x =$ false. Then, for each clause $c \in C$, we choose one of its literals $l \in c$ with value true and add the edge $e_{c,l}$ to S. By construction, S covers all vertices in A and thus has size $|A|$. By the same arguments as before, none of the edges in S are in conflict with each other, meaning that we have constructed a feasible solution to the BMWECP instance. □

4.7 Optimizing layouts for pin accessibility

Having developed a tool to calculate our pin accessibility score for a given layout, we now would like to generate layouts optimized for this score. First, we will discuss how we can modify the ILP we use to solve the CLRP accordingly. At its core, we will use the following ILP formulation to model the pin accessibility score.

Variables:

$m^c \in \{ 0, 1 \}$	Equals 1, if a matching between all nets and some M2 tracks exists in the context $c \in \mathcal{C}$[†]
$p^c_{n,t} \in \{ 0, 1 \}$	Indicates if the matching edge between net n and M2 track t is active, for the matching in the context $c \in \mathcal{C}$
$p_{n,t} \in \{ 0, 1 \}$	Equals 1, if M2 track t is a feasible pin access point for net n

[†]We use the set of contexts $\mathcal{C}$ as defined on page 65.

Objective:

$$\max\left(\sum_{c\in\mathcal{C}} \mathrm{s}(c)m^c\right)$$

Constraints:

$$p_{n,t} \geq p^c_{n,t} \quad \text{for all nets } n\text{, M2 tracks } t\text{, contexts } c \in \mathcal{C} \tag{4.1}$$

$$\left(\sum_{\text{M2 track } t} p^c_{n,t}\right) \geq m^c \quad \text{for all nets } n \text{ and contexts } c \in \mathcal{C} \tag{4.2}$$

$$p^c_{n,t} = 0 \quad \text{for all nets } n \text{ and } c = \text{“}t \text{ is blocked”} \in \mathcal{C} \tag{4.3}$$

$$\sum_n p^c_{n,t} \leq m^c \quad \text{for all M2 tracks } t \text{ and contexts } c \in \mathcal{C} \tag{4.4}$$

Constraints 4.2 and 4.4 ensure that the variables $p^c_{n,t}$ form a matching for each context $c \in \mathcal{C}$, associating all nets to an M2 track if m^c equals 1. Constraints 4.3 require these matchings to obey the restrictions imposed by the contexts $\mathcal{C}$. Finally, constraints 4.1 ensure that each net has the pin access points available, which are required to realize the matchings. Since the objective function matches equation 4.4 on page 62, the above ILP computes a solution whose objective value equals the pin accessibility score.

For a given net n and M2 track t, we will ensure that the variable $p_{n,t}$ equals 1 if and only if M2 track t is a valid pin access point for net n. This is achieved using the following two techniques:

- Based on the via metal shapes described in section 3.5.1.1, we introduce “shadow V1 metal shapes”. Each active variable $p_{n,t}$ induces one active shadow V1 metal shape connecting to the M2 track t. Each active shadow V1 metal shape either coincides with their non-shadow counterparts, or it has to obey design rules with regard to other active metal shapes, just like regular V1 metal shapes. However, two shadow V1 metal shapes do not have to obey regular V1 design rules (most importantly spacing rules) between them. This allows us to model that two pin access points might mutually exclude each other, but might still both participate in the net-to-M2-track matchings for different contexts in $\mathcal{C}$. See fig. 4.7.1 for an example.
- To ensure that, if a shadow V1 metal shape is active it is actually also connected to the rest of the net, we add additional “optional terminals” to our Steiner tree packing problem ILP formulation described in section 3.2.3. They ensure connectivity by modelling a flow from the net’s root terminal to the shadow V1 metal shapes. Using the big-M

method, the constraints forming the flow are only activated based on the $p_{n,t}$ variables. Adding one flow for every M2 track t would add unacceptably many variables and constraints to the ILP. Thus, instead of creating one optional terminal for each $p_{n,t}$ variable, we generate two optional terminals for each net, each of them comprised of all shadow V1 metal shapes. We then use additional constraints to ensure that each of these optional terminals actually connect to the one shadow V1 that induced its activation. Consequently, this limited number of optional terminals also limits the number of active $p_{n,t}$ variables per net to two. This number has been chosen based on theorem 4.3, which shows that, at least in a simplified setting, there always exists a solution with optimium pin accessibility score, in which each net only has two pin access points.

Remark. In section 4.8, we will see that adding this formulation to the routing ILP significantly increases the time needed to find an optimum solution, albeit within a range that is still feasible for most of our applications. If, in a future technology, this changes so that optimizing above formulation becomes prohibitively slow, one could instead try to use a heuristic formulation. Such a heuristic formulation could trade off solution quality for solving speed. For example, instead of the above formulation, we could try to maximize the total number of pin access points. This might be faster to optimize but should still somewhat correlate with the pin accessibility score which we are trying to optimize.

4.7.1 Two pin access points per net are sufficient

In the following, we show that in a simplified setting without design rules or limited routing resources, the optimum pin accessibility score can always be achieved with only two pin access points per net. As in section 4.6, we will formalize this by modeling the pin accessibility score as a set of matchings in a bipartite graph $G = (V, E)$, $V = A \dot\cup B$, where A and B represent the nets and M2 tracks respectively. In this graph, for a net $a \in A$, its incident vertices in B represent the pin access points of net a.

Definition 4.2. We say that an undirected bipartite graph $G = (V, E)$, $V = A \dot\cup B$ is ***A*-factor-critical**, if for each $b \in B$ there exists an A-covering matching $M_b \subset E$, such that M_b exposes b, meaning that the edges M_b do not cover b.

A layout has a perfect pin acessibility score of 2, if its corresponding bipartite graph described above is A-factor-critcal. In this case, we can prove the following.

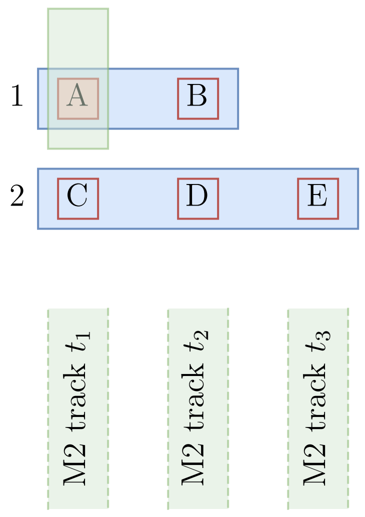


Figure 4.7.1: Example situation to illustrate the concept of shadow V1 vias. The dashed shapes indicate the horizontal positions of the M2 tracks t_1, t_2, t_3. Let us assume the four regular shapes (two blue M1 belonging to nets 1 and 2, one green M2 and one orange V1 marked with A) are active. Let us also assume that the via-to-via spacing rules forbid two vertically aligned active V1 metal shapes on the depicted M1 tracks. Then, activating a shadow V1 metal shape at position D would not be feasible, because it would conflict with the regular V1 shape A. However, activating both shadow V1 metal shapes at positions B and D would be feasible, because by our construction, they do not need to obey via spacing rules to each other. This is important, because both pin access points B and D could participate in one of the net-to-M2-track matchings for different contexts in $\mathcal{C}$. E.g. for $c_1 =$ "t_1 is blocked" $\in \mathcal{C}$, the matching could be (net 1, pin access point B), (net 2, pin access point E), and for $c_2 =$ "t_3 is blocked" $\in \mathcal{C}$ it could be (net 1, pin access point A), (net 2, pin access point D).

Remark. While the approach of disabling all design rules between shadow V1 metal shapes is suitable for the technology node we are working on, it would not be able to model arbitrary other V1 design rules. E.g. if a V1 metal shape at position B would be in conflict with a V1 shape at position E, we would have to modify our approach to prevent the two pin access points B and E to be simultaneously used for the matching in the same context $c \in \mathcal{C}$. This could be achieved by enabling or disabling the design rule evaluation between the V1 metal shapes based on the $p^c_{n,t}$ variables.

Theorem 4.3. Let $G = (V, E)$, $V = A \mathbin{\dot{\cup}} B$ be a bipartite A-factor-critical graph. Then, there exists a subgraph $G' = (V, E')$ with $E' \subseteq E$ that is also A-factor-critical and whose vertices in A all have a degree of at most two.

Proof. Let $G = (V, E)$, $V = A \mathbin{\dot{\cup}} B$ be such a bipartite A-factor-critical graph, where for at least one $a \in A$ we have $|\Gamma(a)| > 2$. We will show that we can remove one of the incident edges of a without destroying the A-factor-criticality. This step can then be used inductively to create a graph with the desired degree constraints for the vertices in a.

Let us number the neighbors of a by denoting $\Gamma(a) = \{ b_1, \ldots, b_k \}$. Additionally, we name the edge $e_1 := \{ a, b_1 \} \in E$. Let M_1 be an A-covering matching exposing b_1. Then $e_1 \notin M_1$. Let $\beta_1 \in B$ be the matching partner of a in M_1.

Similarly, let M_{β_1} be an A-covering matching exposing β_1, and let $\beta_2 \in B$ be the neighbor of a in M_{β_1}.

Let $f \in \delta(a) \setminus \{ \{ a, \beta_1 \}, \{ a, \beta_2 \} \}$. If we can now show that for any $x \in B$ there exists a matching R_x that exposes x and does not contain f, the smaller graph $G' := (V, E \setminus \{ f \})$ is still A-factor-critical and our proof is concluded.

Therefore, let $x \in B$. If $x = \beta_1$, M_{β_1} is a matching with the desired properties. Otherwise, if $x \neq \beta_1$, we construct R_x as follows. Let M_x be an A-covering matching exposing x. If $f \notin M_x$, we can use it as R_x and are done. Otherwise, let us examine the symmetric difference between M_x and M_1 denoted by $M_x \triangle M_1 \subset E$. Note that $f \notin M_1$ and thus $f \in M_x \triangle M_1$.

It is easy to see that the symmetric difference between two matchings consists of disjoint circles and/or paths. In any such circle or path the edges from M_x and M_1 appear alternatingly.

If $f \in C$ for some circle $C \subset M_x \triangle M_1$, we define R_x as M_x augmented along C, i.e. we set $R_x := M_x \triangle C$ [‡]. M_x and R_x cover the same elements in V, but R_x does not contain f. Thus, R_x fulfills all desired criteria, and we are done.

Otherwise, $f \in P$ for some path $P \subset M_x \triangle M_1$. Both M_x and M_1 cover all of A, i.e. that both endpoints of P lie in B and thus P has even length. If x is not covered by P, we can again simply augment M_x along P. The resulting matching R_x does no longer contain f, still exposes x and covers all elements in A. Thus, R_x fulfills all desired criteria, and we are done.

[‡]Augmenting a matching M along a circle or path is a common notation, employed e.g. in [KV18]. It works, whenever in that circle or path every second edge belongs to M. In this proof, augmentation will always happen along circles or paths with an even length, meaning that the resulting matching has the same cardinality as the original one.

Let us now consider the case that P covers x. Our goal is to modify P so that it still contains f but no longer covers x. Then, we can augment M_x along the modified path to obtain R_x with the desired properties. x must be one of the endpoints of P, because it is exposed in M_x. Note that the orientation of f in P is as depicted in fig. 4.7.2.1, because P contains edges from M_x and M_1 alternatingly and the edge in P incident to x is an element of M_1. If M_x exposes b_1, then we can simply define $R_x \coloneqq M_x \setminus \{ f \} \cup \{ e_1 \}$ and are done. Thus, we assume that M_x covers b_1. As b_1 is exposed in M_1, there exists a path Q in $M_x \triangle M_1$ for which b_1 is an endpoint.

If P and Q are the same path, we can augment M_x along the circle formed by by the sub-path $P_{[b_1,a]}$ and e_1 (see fig. 4.7.2.2). Note that this circle contains f, due to the orientation of f in P discussed above. Finally, if P and Q are disjoint paths, let y denote the endpoint of P that is not x. Then, we augment M_x along the joint path $P_{[y,a]}, e_1, Q$, as depicted in fig. 4.7.2.3. This is possible because exactly every second edge on this combined path is contained in M_x.

□

4.8 Practical results

To demonstrate that our implementation is suitable for optimizing our pin accessibility score, we run a number of experiments. We compare to the technique which was previously used in BonnCell to ensure pin accessibility, which we will refer to as the "**forced M2**" technique. In this, for each external net we would create an artificial pin spanning the entire M2 layer, forcing every net to connect to this layer. By definition of our pin accessibility score, this technique ensures that any layout has a score of at least 1.

Compared to the newly defined ILP formulation, which we refer to as "**MIP opt**" technique, the *forced M2* technique is at a disadvantage because it needs to place an actual M2 shape for every net, whereas in the *MIP opt* technique it suffices to ensure that such a connection is possible. See fig. 4.8.1 for an example where this leads to a worse score for the *forced M2* technique, even though both techniques yielded the same result except for the M2 metal shapes. Therefore, in order to level the playing field, as post-processing step, for all layouts generated in our experiments we remove all M2 shapes and their incident vias, if the M2 shape is only connected to a single via.

Our experiment now works as follows. We run BonnCell with four different settings. In all runs, we choose the cell width as the main order objective function.

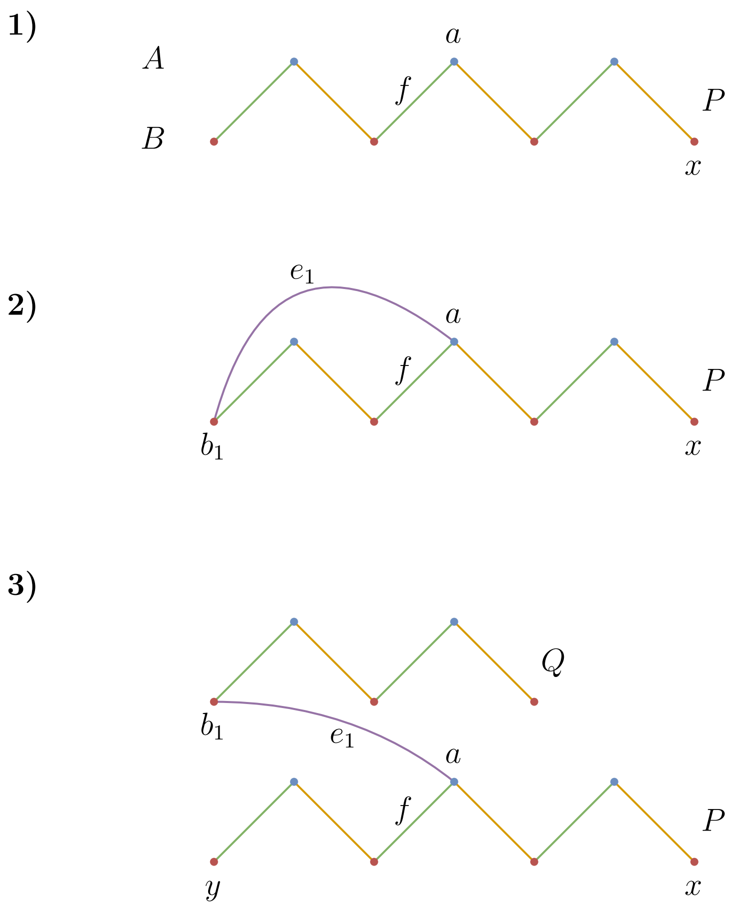


Figure 4.7.2: Depicted are three versions of the path P. The green edges belong to M_x, while the orange edges belong to M_1. Vertices in A and B are color-coded blue and red respectively. 1) shows the path P, its contained edge f and vertices x and a. Because $f \in M_x$, f is not contained in the sub-path $P_{[x,a]}$. 2) depicts the circle that can be constructed from P using e_1, if P has b_1 as one of its endpoints. 3) demonstrates the constructions of a path using parts of P, e_1, and Q.

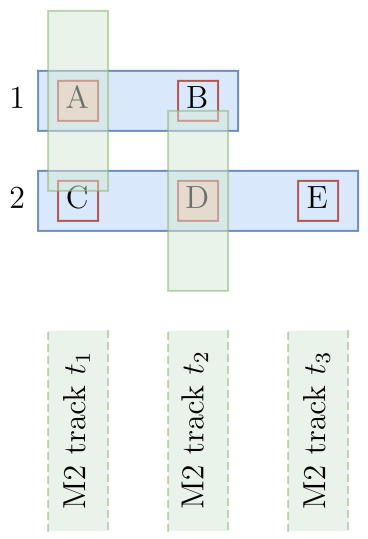


Figure 4.8.1: Shows the routing of the nets 1 and 2, both consisting of one M1, V1, and M2 shape each. With the given layout, the used pin access point A blocks the pin access point C and the M2 shape over D blocks pin access point B. Thus, we are left with the valid pin access combinations $\{ (A, D), (A, E) \}$, which would yield a pin accessibility score of 1.67. However, in the same layout, if we remove the V1 and M2, we get the pin access combinations $\{ (A, D), (A, E), (B, C), (B, E) \}$, yielding a perfect pin accessibility score of 2.

- As a *baseline*, we run BonnCell with its default settings. We use the *forced M2* technique described above to ensure that all nets have accessible pins. First, we find a routable placement for the transistors that minimizes the cell width, followed by an approximation of the netlength, namely the bounding box netlength. The bounding box netlength effectively measures the horizontal distance between the leftmost and rightmost pin or transistor contact of a net and its use in BonnCell is described in detail in [Cre19]. Then, for the chosen placement, we solve the CLRP once by optimizing the routing ILP for netlength.
- In the *routing-opt* run, we utilize the transistor placements from the baseline run. We rerun the CLRP ILP formulation, optimizing lexicographically the pin accessibility score formulated in section 4.5, followed by the netlength.
- In the *placement-opt* run, we solve the CLRP to optimality with the netlength objective function, once for every possible placement of the transistors with the same cell width as in the baseline run. Again, for

each CLRP instance we utilize the *forced M2* technique. Amongst the generated layouts, we choose the one that lexicographically maximizes the pin accessibility score and minimizes the horizontal netlength.

- In the *all-opt* run, we combine both previous techniques. We again solve the CLRP ILP to optimality for each possible placement of the transistors with the same cell width as the baseline run. Here, as in the *routing-opt* run, the ILP is solved by optimizing lexicographically the pin accessibility score formulated followed by the netlength. Then, as in the *placement-opt* run, amongst the generated layouts we choose the one maximizing the pin accessibility score and the horizontal netlength lexicographically.[§]

In the interest of runtime, in all cases we stop the ILP optimization once a solution is found and the CPLEX solver can prove that it lies within 2% of the optimum. Since the lexicographical objective function in the *routing-opt* and *all-opt* runs is realized by multiplying the first order objective with a large constant and then adding the second order objective, the 2% optimality threshold can have the effect that the second order objective is not optimized at all. We execute all runs with a single thread and a runtime limit of 24 hours.

The results are depicted as a hat graph [Wit19] in fig. 4.8.2. For each cell, a black vertical line indicates the pin accessibility score achieved in the *baseline* run. If the *routing-opt*, *placement-opt*, or *all-opt* runs produced layouts with a score better than that of the *baseline* run, the improvement is displayed as an orange, blue, or green bar respectively. Out of our testbed of 94 cells, we were able to compute solutions for all four runs for 51 cells within the time limit. For 17 cells, the pin accessibility score did not improve in the *all-opt* run compared to the *baseline* run, meaning that either a timeout occurred, or the *baseline* run already generated a layout with the best pin accessibility score amongst all possible layouts with the same cell width. In the table, we display the other 34 cells, for which we managed to find a layout with an improved pin accessibility score. If a *placement-opt*, *routing-opt*, or *all-opt* run did not manage to explore the whole search place within the given time frame of 24 hours, its bar is marked with a red square at the end, indicating that the optimum solution could yield an even better result with respect to the respective objective functions.

The *all-opt* run improved the pin accessibility score of 31 cells to its maximum value of 2. Only for 2 cells the *all-opt* run yielded a better pin accessibility score better than the *routing-opt* run. This indicates that in most cases, changing the transistor placement is not necessary to yield the

[§]This is similar to the approach used in [Cre19] to find globally optimum routings.

optimum pin accessibility score. For 19 cells, the *all-opt* run gave a better pin accessibility score than the *placement-opt* run. This illustrates that simply enumerating some layouts and taking the one with the best pin accessibility score is an insufficient method for optimizing the score. Instead, in many cases, the formulation as an ILP objective function is necessary to obtain a layout with the optimum pin accessibility score.

A manual comparison between some layouts from the *baseline* and *routing-opt* runs revealed that the CLRP solutions often only differ in the size of the pin shapes. However, there are also some cases where differences are more substantial. See fig. 4.8.3 for an example.

In terms of runtime, the *placement-opt* and *all-opt* runs are several orders of magnitude slower than the *baseline* run. This is because they optimize an CLRP instance for many placements, while the *baseline* run only optimizes a single CLRP instance. We do not give in-depth runtime comparisons between the *baseline* and the *placement-* and *all-opt* runs, because the exact runtime difference largely depends on the number of different transistor placements, which is not very informative. The smallest cell for which we could not compute a solution for the *all-opt* run had a cell width of ten gate pitches in the *baseline* run. Luckily, our test revealed that in almost all cases, one can obtain the best possible pin accessibility score by using the *routing-opt* setup, which is much faster than the *placement-* and *all-opt* runs.

To test if optimizing the pin accessibility score ILP formulation is suitable even for larger cells where the *baseline* run did not find a solution within 24 hours, we run another experiment. We fix routable transistor placements for all cells in our testbed, which have been obtained by employing multi-threading and various partitioning techniques described in [Cre19]. Unlike before, due to the used heuristics some of these transistor placements are not width-optimal. Then, for each cell we solve the CLRP twice by optimizing the routing ILP once with the normal netlength objective function (**netlength-routing**) and once with the pin accessibility score objective formulation (**PA-routing**). Both runs again have a runtime limit of 24 hours. For the *PA-routing* run, we provide the ILP solver with the solution of the *netlength-routing* run (if one exists) as an initial starting solution. This way, the *PA-routing* run is guaranteed to produce some solution, even if it encounters a timeout, in which case we can assess how much improvement in the pin accessibility score was attained after 24 hours of computation.

The results of this experiment are presented in fig. 4.8.4. Out of our testbed of 94 cells, BonnCell was able to produce a solution for the *netlength-routing* run for 82 cells. While the runtime increase for the *PA-routing* run is significant for most cells, BonnCell was still able to compute layouts with an improved pin accessibility score for 47 cells. Notably, even for those 23 cells

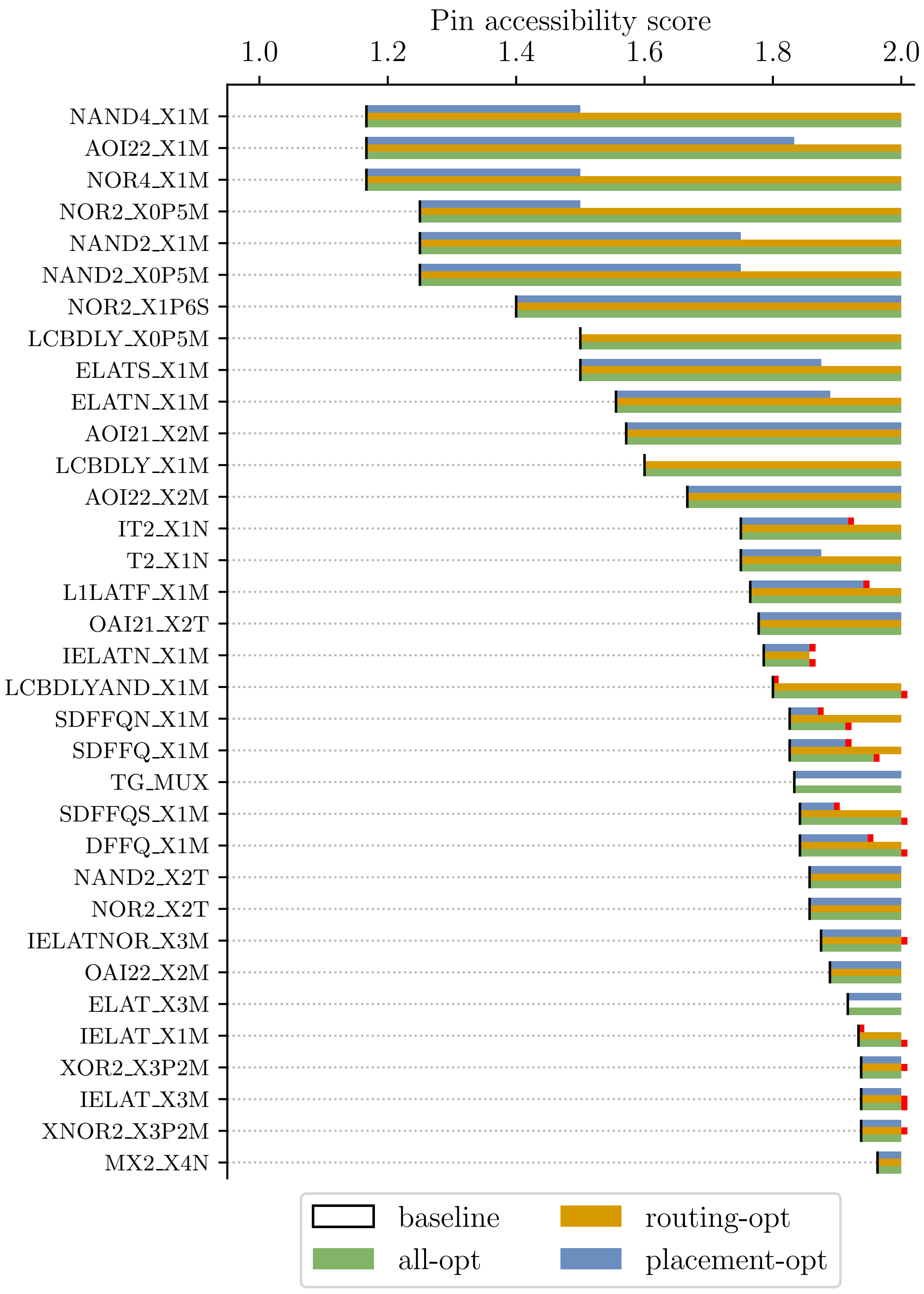
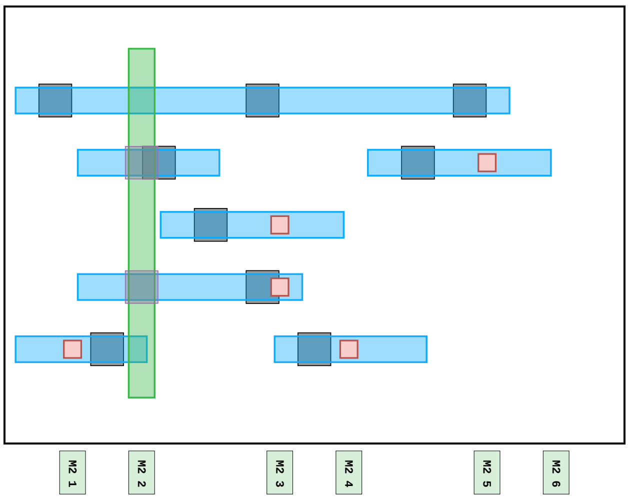
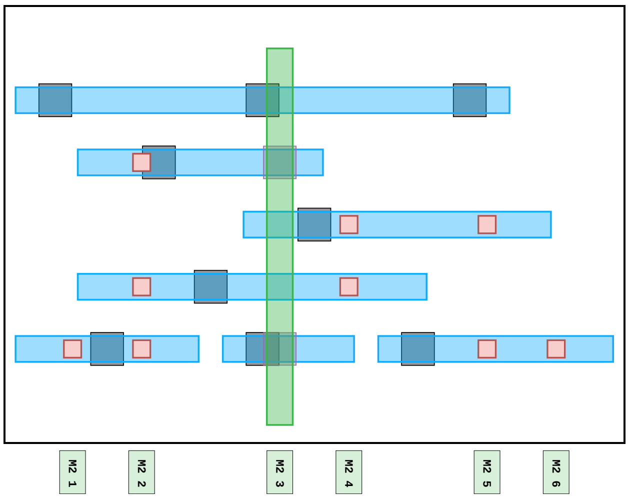


Figure 4.8.2: Comparison between the pin accessibility scores obtained from four different runs, which are described in detail on pages 74 ff.



(a) A cell layout optimized for netlength, yielding a pin accessibility score of 1.167.



(b) A cell layout optimized lexicographically for pin accessibility and netlength, yielding a pin accessibility score of 2.

Figure 4.8.3: Two cell layouts realizing the instance AOI22_X1M, both using the same transistor positions. Depicted are M1 in blue, V1 in purple, and M2 in green. Connections from M1 to the transistors blow M1 are marked in dark gray. Pin access points are the M2 shape and the red squares. The M2 track positions are marked at the bottom. To improve the pin access score of layout (a), one could try to increase the size of the M1 pin shapes. However, this is not possible for the M1 pin in the lower left corner, because it is boxed in by the occupied M2 track and the lower right M1 metal shape and cannot easily be extended. BonnCell solves this situation by moving the M2 metal shape to the right and switching the tracks of several M1 shapes.

for which the *PA-routing* run encountered a timeout, BonnCell still managed to improve the pin accessibility score for 18 of those cells. This illustrates that our chosen approach is suitable for optimizing the pin accessibility score for almost all cells, for which BonnCell is normally able to find a solution.

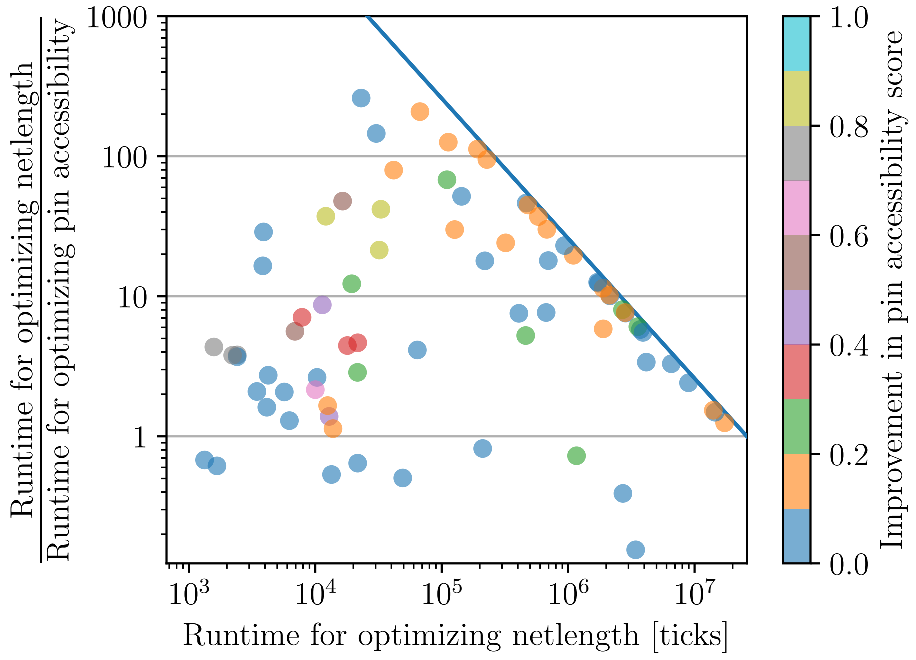


Figure 4.8.4: Comparison between the *netlength-routing* and *PA-routing* runs. Each dot represents both runs for one cell. The absolute improvements in pin accessibility score of the *PA-routing* over the *netlength-routing* runs are color-coded. The dots that touch the blue diagonal correspond to runs where the *PA-routing* run reached the runtime limit of 24 hours. For the other dots, the *PA-routing* run was solved to optimality, yielding the best possible pin accessibility score for the chosen transistor placement.

4.9 Conclusion

In this chapter, we have developed a pin accessibility score for a state-of-the-art technology node. We have embedded this in an abstract, technology independent setting that provides a guideline on how to define similar pin accessibility metrics in future technologies. Using a SAT formulation, we have shown how the score can be efficiently calculated for a given cell layout.

Furthermore, we have seen one possible way of modelling the pin accessibility score in BonnCell's CLRP ILP formulation. With this, we are able to compute layouts with the optimum pin accessibility score amongst all layouts with a certain cell width. To the best of our knowledge, BonnCell provides the first suite of algorithms to achieve this.

While finding optimum solutions is time-consuming, it is still a feasible technique for most smaller cells. These cells are typically the ones that have the highest pins per cell area ratio, which makes them an important use-case for pin accessibility optimization. For larger cells, we have seen that pin accessibility can still be optimized. By starting from a netlength optimized CLRP solution created by BonnCell, in most cases we can obtain a solution with improved pin-accessibility within reasonable runtime.

Summing up, BonnCell now provides a fully automated way of measuring pin accessibility and generating pin accessibility optimized layouts.

Summary

In this work, we have provided an in-depth description of BonnCell's routing engine. BonnCell models all design rules explicitly in its routing ILP formulation. This allows BonnCell to model the entire routing solution space exactly, so it can decide definitively which transistor placements are routable. It can provide optimum routing solutions in such cases, without artificially reducing the solution space or relying on heuristics, which sets BonnCell apart from other automatic cell layout tools.

Although this exact approach creates great layouts, it comes with the drawback that implementing and maintaining all design rules as ILP constraints takes a lot of effort. In chapter 3 we have addressed this by introducing a novel approach to model the CLRP as an ILP. We deduced that instead of implementing design rules directly as ILP constraints, it is beneficial to introduce an intermediate modelling layer. Through the notion of partial polygons, we have bundled ILP variables with their geometric meaning. This allowed us to propose a design rule framework that uses geometric algorithms to implement design rules. This, in conjunction with mimicking the structure observed in various design manuals, allows our framework to greatly reduce the effort needed to implement and maintain design rules in BonnCell.

Through these changes, BonnCell can now be adapted faster to new technology nodes. This early availability of BonnCell significantly increases its usefulness for our industry partner IBM, who is now able to utilize BonnCell even during the early exploration stages of new technology nodes.

Our second big contribution in this work was the addition of pin accessibility considerations to BonnCell. Based on pin accessibility scoring methods proposed in other works, we have developed a new scoring function that is both easy to compute in practice and can be concisely embedded into our routing ILP model. Previously, at our industry partner IBM, the accessibility of pins of cell layouts was assessed by human experts. Layouts without sufficient pin accessibility were manually adjusted. By offering an automated way to compute and optimize the pin accessibility score within BonnCell, this process now requires much less labor.

Overall, our contributions have significantly reduced the development times of BonnCell, automated pin accessibility considerations, and thus improved the availability and usefulness of the tool. In turn, this allows for the creation of better cell layouts with less human effort.

Bibliography

[22a] *Big M Method.* In: *Wikipedia.* 2022. URL: `https://en.wikipedia.org/w/index.php?title=Big_M_method&oldid=1111237191` (visited on 11/07/2022) (cit. on p. 12).

[22b] *IBM CPLEX Documentation.* 2022. URL: `https://www.ibm.com/docs/en/icos/22.1.0?topic=parameters-deterministic-time-limit` (visited on 04/11/2023) (cit. on p. 25).

[22c] *Quadtree.* In: *Wikipedia.* 2022. URL: `https://en.wikipedia.org/w/index.php?title=Quadtree&oldid=1115215433` (visited on 12/07/2022) (cit. on p. 30).

[23] *Fin Field-Effect Transistor.* In: *Wikipedia.* 2023. URL: `https://en.wikipedia.org/w/index.php?title=Fin_field-effect_transistor&oldid=1134478041` (visited on 04/06/2023) (cit. on p. 5).

[Ahr+15] M. Ahrens, M. Gester, N. Klewinghaus, D. Müller, S. Peyer, C. Schulte, and G. Tellez. "Detailed Routing Algorithms for Advanced Technology Nodes". In: *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 34.4 (2015), pp. 563–576. URL: `http://ieeexplore.ieee.org/document/6998035/` (visited on 02/06/2023) (cit. on p. 57).

[BFFH20] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger. "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020". In: *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions.* Ed. by T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda. Vol. B-2020-1. Department of Computer Science Report Series B. University of Helsinki, 2020, pp. 51–53 (cit. on pp. 41, 67).

[Bib22] J. F. Biburger. "MIP-Formulierungen im Detailed Routing". BA thesis. University of Bonn, 2022 (cit. on p. 53).

[Che+21] C.-K. Cheng, C.-T. Ho, D. Lee, B. Lin, and D. Park. “Complementary-FET (CFET) Standard Cell Synthesis Framework for Design and System Technology Co-Optimization Using SMT”. In: *IEEE Trans. VLSI Syst.* 29.6 (2021), pp. 1178–1191. URL: `https://ieeexplore.ieee.org/document/9390403/` (visited on 04/05/2023) (cit. on pp. 5, 7).

[Che+22] C.-K. Cheng, A. B. Kahng, H. Kim, M. Kim, D. Lee, D. Park, and M. Woo. “PROBE2.0: A Systematic Framework for Routability Assessment From Technology to Design in Advanced Nodes”. In: *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 41.5 (2022), pp. 1495–1508. URL: `https://ieeexplore.ieee.org/document/9467334/` (visited on 02/06/2023) (cit. on p. 58).

[CHLL21] C.-K. Cheng, C.-T. Ho, D. Lee, and B. Lin. “Multirow Complementary-FET (CFET) Standard Cell Synthesis Framework Using Satisfiability Modulo Theories (SMTs)”. In: *IEEE J. Explor. Solid-State Comput. Devices Circuits* 7.1 (2021), pp. 43–51. URL: `https://ieeexplore.ieee.org/document/9466133/` (visited on 04/06/2023) (cit. on p. 5).

[Cho94] S. Chopra. “Comparison of Formulations and a Heuristic for Packing Steiner Trees in a Graph”. In: *Ann Oper Res* 50.1 (1994), pp. 143–171. URL: `http://link.springer.com/10.1007/BF02085638` (visited on 11/21/2022) (cit. on p. 14).

[Cre19] P. Cremer. “Algorithms for Cell Layout”. PhD thesis. University of Bonn, 2019 (cit. on pp. 6, 7, 12, 16, 24, 76–78).

[DCM17] Y. Ding, C. Chu, and W.-K. Mak. “Pin Accessibility-Driven Detailed Placement Refinement”. In: *Proceedings of the 2017 ACM on International Symposium on Physical Design.* ISPD ’17: International Symposium on Physical Design. Portland Oregon USA: ACM, 2017, pp. 133–140. URL: `https://dl.acm.org/doi/10.1145/3036669.3036679` (visited on 03/09/2021) (cit. on pp. 57, 59).

[GM93] M. X. Goemans and Y.-S. Myung. “A Catalog of Steiner Tree Formulations”. In: *Networks* 23.1 (1993), pp. 19–28. URL: `https://onlinelibrary.wiley.com/doi/10.1002/net.3230230104` (visited on 11/21/2022) (cit. on pp. 14, 22).

[HCN09] C.-H. Hsu, Y.-W. Chang, and S. R. Nassif. “Simultaneous layout migration and decomposition for double patterning technology”. In: *Proceedings of the 2009 International Conference on Computer-Aided Design.* 2009, pp. 595–600 (cit. on p. 3).

[HK12] N.-D. Hoàng and T. Koch. “Steiner Tree Packing Revisited”. In: *Math Meth Oper Res* 76.1 (2012), pp. 95–123. URL: `http://link.springer.com/10.1007/s00186-012-0391-8` (visited on 11/21/2022) (cit. on p. 14).

[Ho+23] C.-T. Ho, A. Ho, M. Fojtik, M. Kim, S. Wei, Y. Li, B. Khailany, and H. Ren. “NVCell 2: Routability-Driven Standard Cell Layout in Advanced Nodes with Lattice Graph Routability Model”. In: *Proceedings of the 2023 International Symposium on Physical Design.* ISPD ’23: International Symposium on Physical Design. Virtual Event USA: ACM, 2023, pp. 44–52. URL: `https://dl.acm.org/doi/10.1145/3569052.3578920` (visited on 04/05/2023) (cit. on p. 7).

[KJK21] S. Kim, K. Jo, and T. Kim. “Boosting Pin Accessibility Through Cell Layout Topology Diversification”. In: *Proceedings of the 26th Asia and South Pacific Design Automation Conference.* ASPDAC ’21: 26th Asia and South Pacific Design Automation Conference. Tokyo Japan: ACM, 2021, pp. 183–188. URL: `https://dl.acm.org/doi/10.1145/3394885.3431567` (visited on 01/19/2023) (cit. on p. 62).

[Klo18] B. Klotz. “Faster Leaf Cell Placement Algorithms”. MA thesis. University of Bonn, 2018 (cit. on p. 42).

[KV18] B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms.* Vol. 6. Algorithms and Combinatorics. Springer, 2018. URL: `https://link.springer.com/10.1007/978-3-662-56039-6` (cit. on p. 73).

[KWX20] A. B. Kahng, L. Wang, and B. Xu. “The Tao of PAO: Anatomy of a Pin Access Oracle for Detailed Routing”. In: *2020 57th ACM/IEEE Design Automation Conference (DAC).* 2020 57th ACM/IEEE Design Automation Conference (DAC). San Francisco, CA, USA: IEEE, 2020, pp. 1–6. URL: `https://ieeexplore.ieee.org/document/9218532/` (visited on 01/19/2023) (cit. on p. 57).

[Lee+21] D. Lee, D. Park, C.-T. Ho, I. Kang, H. Kim, S. Gao, B. Lin, and C.-K. Cheng. “SP&R: SMT-Based Simultaneous Place-and-Route for Standard Cell Synthesis of Advanced Nodes”. In: *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 40.10 (2021), pp. 2142–2155. URL: `https://ieeexplore.ieee.org/document/9259080/` (visited on 04/05/2023) (cit. on p. 5).

[Li+19] Y.-L. Li, S.-T. Lin, S. Nishizawa, H.-Y. Su, M.-J. Fong, O. Chen, and H. Onodera. “NCTUcell: A DDA-Aware Cell Library Generator for FinFET Structure with Implicitly Adjustable Grid Map”. In: *Proceedings of the 56th Annual Design Automation Conference 2019.* DAC ’19: The 56th Annual Design Automation Conference 2019. Las Vegas NV USA: ACM, 2019, pp. 1–6. URL: `https://dl.acm.org/doi/10.1145/3316781.3317868` (visited on 04/05/2023) (cit. on pp. 6, 7).

[Lu+15] A. Lu, H.-J. Lu, E.-J. Jang, Y.-P. Lin, C.-H. Hung, C.-C. Chuang, and R.-B. Lin. “Simultaneous Transistor Pairing and Placement for CMOS Standard Cells”. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015.* Design, Automation and Test in Europe. Grenoble, France: IEEE Conference Publications, 2015, pp. 1647–1652. URL: `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7092657` (visited on 04/06/2023) (cit. on p. 6).

[Par+20] D. Park, D. Lee, I. Kang, C. Holtz, S. Gao, B. Lin, and C.-K. Cheng. “Grid-Based Framework for Routability Analysis and Diagnosis With Conditional Design Rules”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.12 (2020), pp. 5097–5110 (cit. on p. 41).

[Pol03] T. Polzin. “Algorithms for the Steiner Problem in Networks”. PhD thesis. Universität des Saarlandes, 2003 (cit. on pp. 14, 19).

[RFK21] H. Ren, M. Fojtik, and B. Khailany. *NVCell: Standard Cell Layout in Advanced Technology Nodes with Reinforcement Learning.* 2021. arXiv: `arXiv:2107.07044`. URL: `http://arxiv.org/abs/2107.07044` (visited on 04/05/2023). preprint (cit. on p. 7).

[Rub11] P. A. Rubin. *Perils of ”Big M”.* OR in an OB World. 2011. URL: `https://orinanobworld.blogspot.com/2011/07/perils-of-big-m.html` (visited on 11/07/2022) (cit. on p. 12).

[Sch23] J. M. Schürks. “SAT-based Algorithms for Leafcell Layout”. MA thesis. University of Bonn, 2023 (cit. on p. 41).

[SJKS17] J. Seo, J. Jung, S. Kim, and Y. Shin. “Pin Accessibility-Driven Cell Layout Redesign and Placement Optimization”. In: *Proceedings of the 54th Annual Design Automation Conference 2017.* DAC ’17: The 54th Annual Design Automation Conference 2017. Austin TX USA: ACM, 2017, pp. 1–6. URL: `https:`

//dl.acm.org/doi/10.1145/3061639.3062302 (visited on 03/09/2021) (cit. on pp. 3, 60, 61).

[SR19] A. Sorokin and N. Ryzhenko. “SAT-Based Placement Adjustment of FinFETs inside Unroutable Standard Cells Targeting Feasible DRC-Clean Routing”. In: *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. GLSVLSI ’19: Great Lakes Symposium on VLSI 2019. Tysons Corner VA USA: ACM, 2019, pp. 159–164. URL: https://dl.acm.org/doi/10.1145/3299874.3317965 (visited on 04/11/2023) (cit. on p. 6).

[Tag+10] T. Taghavi, Z. Li, C. Alpert, G.-J. Nam, A. Huber, and S. Ramji. “New Placement Prediction and Mitigation Techniques for Local Routing Congestion”. In: *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). San Jose, CA, USA: IEEE, 2010, pp. 621–624. URL: http://ieeexplore.ieee.org/document/5654225/ (visited on 01/18/2023) (cit. on pp. 57, 60).

[Tho19] S. Thomä. “Purely MIP-based Cell Layout”. MA thesis. University of Bonn, 2019 (cit. on p. 5).

[Tse+19] I.-L. Tseng, Z. C. Lee, V. Tripathi, C. M. Tommy Yip, Z. Chen, and J. Ong. “A System for Standard Cell Routability Checking and Placement Routability Improvements”. In: *2019 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. 2019 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS). Bangkok, Thailand: IEEE, 2019, pp. 125–128. URL: https://ieeexplore.ieee.org/document/8953119/ (visited on 01/18/2023) (cit. on pp. 57, 58).

[vCHSW19] P. v. Cleeff, S. Hougardy, J. Silvanus, and T. Werner. “BonnCell: Automatic Cell Layout in the 7-Nm Era”. In: *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 39.10 (2019), pp. 2872–2885. URL: https://ieeexplore.ieee.org/document/8945429/ (visited on 12/05/2022) (cit. on p. 14).

[Vic18] R. Vicari. “Simplex Based Graphs Yield Large Integrality Gaps for the Bidirected Cut Relaxation”. MA thesis. University of Bonn, 2018 (cit. on p. 14).

[Wit19] J. K. Witt. “Introducing Hat Graphs”. In: *Cogn. Research* 4.1 (2019), p. 31. URL: https://cognitiveresearchjournal.springeropen.com/articles/10.1186/s41235-019-0182-3 (visited on 01/31/2023) (cit. on pp. 41, 77).

[Won84] R. T. Wong. “A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph”. In: *Mathematical Programming* 28.3 (1984), pp. 271–287. URL: `http://link.springer.com/10.1007/BF02612335` (visited on 11/21/2022) (cit. on p. 14).

[Wu+13] P.-H. Wu, M. P.-H. Lin, T.-C. Chen, T.-Y. Ho, Y.-C. Chen, S.-R. Siao, and S.-H. Lin. “1-D Cell Generation With Printability Enhancement”. In: *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 32.3 (2013), pp. 419–432. URL: `http://ieeexplore.ieee.org/document/6461981/` (visited on 04/11/2023) (cit. on p. 6).

[XLLP17] X. Xu, Y. Lin, V. Livramento, and D. Z. Pan. “Concurrent Pin Access Optimization for Unidirectional Routing”. In: *Proceedings of the 54th Annual Design Automation Conference 2017*. DAC ’17: The 54th Annual Design Automation Conference 2017. Austin TX USA: ACM, 2017, pp. 1–6. URL: `https://dl.acm.org/doi/10.1145/3061639.3062214` (visited on 03/09/2021) (cit. on p. 59).

[Xu+15] X. Xu, B. Cline, G. Yeric, B. Yu, and D. Z. Pan. “Self-Aligned Double Patterning Aware Pin Access and Standard Cell Layout Co-Optimization”. In: *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 34.5 (2015), pp. 699–712. URL: `http://ieeexplore.ieee.org/document/7031419/` (visited on 03/09/2021) (cit. on pp. 7, 59, 61).

[Xu+16] X. Xu, B. Yu, J.-R. Gao, C.-L. Hsu, and D. Z. Pan. “PARR: Pin-Access Planning and Regular Routing for Self-Aligned Double Patterning”. In: *ACM Trans. Des. Autom. Electron. Syst.* 21.3 (2016), pp. 1–21. URL: `https://dl.acm.org/doi/10.1145/2842612` (visited on 03/09/2021) (cit. on p. 59).