Timing-Driven Macro Placement

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Vorgelegt von

Philipp Ochsendorf

geboren in

Frankfurt am Main

Bonn, Januar 2019

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

Erstgutachter:	Herr Professor Dr. Jens Vygen
Zweitgutachter:	Herr Professor Dr. Dr. h.c. Bernhard Korte
Tag der Promotion:	22. März 2019
Erscheinungsjahr:	2019

Acknowledgments

This dissertation would not have been possible without the support of many people. First and foremost I would like to express my gratitude to Professor Dr. Jens Vygen for the inspiring supervision of my doctoral studies.

I also wish to thank Professor Dr. Dr. h.c. Bernhard Korte and Professor Dr. Stefan Hougardy for providing outstanding working conditions at the Research Institute for Discrete Mathematics of the University of Bonn.

I would also like to thank the present and past members of the BonnPlace-Team, especially Dr. Ulrich Brenner, Dr. Jan Schneider and Jannik Silvanus. The extremely good atmosphere in this team, the support as well as the fruitful discussions and ideas proved to be invaluable. Special thanks also go to all students for their collaboration, especially Nils Hoppmann.

I also thank all other colleagues of the institute for inspiring discussions, in particular Anna Hermann. Special thanks go to Professor Dr. Stephan Held for his support with the IBM timing environment. Furthermore I thank Dr. Christoph Bartoschek for his influence on my programming attitude, which set me off on the right track.

But my biggest thanks go to Conni, my parents and my sister Lisa. Without their permanent support, encouragement and endless patience, I would not have been able to finish this thesis.

Contents

Chapter 1

Introduction

Computer chips are at the heart of most electronic devices in our daily lives: embedded systems, smartphones, personal computers and servers. Many applications need powerful chips that yet can be operated efficiently. At the same time processing ever increasing amounts of data constantly requires faster and faster microprocessors.

Creating the next generations of these miniature and almost magic devices is achieved in the physical design process. Here the most advanced manufacturing technologies meet with the fields of computer science and mathematics. Many practical challenges can be formulated as fascinating combinatorial optimization problems.

Designing chips with billions of transistors would not be possible without the help of automatic tools. To manage this complexity, transistors are partitioned into modules that either compute a Boolean function or store bits. These essential building blocks are called circuits or cells.

This thesis focuses on the placement problem, where circuits have to be arranged without overlaps on the chip image. As one of the very first stages of physical design, placement quality has far reaching effects on the overall chip performance. At the same time this stage is repeated frequently which particularly requires fast algorithms.

Favorable placements primarily need to enable good interconnections of the cells. On the one hand, these connections must be realized by wires in the subsequent routing design phase. On the other hand, all signals on a chip have to meet individual deadlines at which they must reach certain points on the design. Meeting these deadlines particularly requires short and thus fast connections. The deadlines define the contracts by which different cells are guaranteed to interoperate correctly. Especially for modern technology nodes these timing challenges require great attention already during the placement stage.

Algorithms for placement usually distinguish between two types of problems: Macro placement deals with comparably few but very large cells, the macros. Finding good positions for macros is challenging since packing macros disjointly is NP-hard and since the effect of macro positions on the overall placement objectives is hard to estimate. With fixed macro positions the remainder of cells, the standard cells, all share a common width or height. This property considerably simplifies finding feasible placements. But identifying good standard cell positions that can be routed easily and satisfy timing requirements remains challenging, both in theory and practice.

To the best of our knowledge, we address timing characteristics for the very first time directly during macro placement. This leads to new extensions of BonnPlace, the placement framework of the BonnTools suite (Korte, Rautenbach and Vygen 2007)

developed at the Research Institute for Discrete Mathematics of the University of Bonn in an industrial cooperation with International Business Machines Corporation (IBM).

For this purpose we propose a new timing model for macros and develop algorithms that compute macro placements with good timing traits in this model. For evaluating macro placements thoroughly we extend BonnPlace to become more efficient and yet more effective with respect to routing and timing objectives.

In Chapter 2 we formally introduce common chip design notation and outline general placement objectives and constraints, particularly arising from routing and timing optimization. Afterwards we focus on the two central problem variants, Macro Placement and Global Placement, before we review previous work.

Thereafter we explain how timing objectives for macros can be translated into geometric distance bounds. This leads to an extension of the classical Rectangle Packing Problem that is presented in Chapter 3. We analyze basic properties of this generalized packing problem before we develop a very efficient algorithm for a certain class of instances. We show in Theorem 3.34 that such instances can be solved optimally in $\mathcal{O}(nm)$ running time where n and m denote the number of rectangles and distance constraints. Moreover we devise strategies for computing equivalent and compact representations of our macro timing model. We prove that for any pair of rectangles a single (generalized) distance constraint suffices.

Chapter 4 is devoted to BonnMacro, the macro placement module of BonnPlace. We first revisit the central approach of this algorithm. Afterwards we elaborate how to optimize timing characteristics of macros in this framework. Therefore we develop the essential ingredient to enable this optimization: an efficient formulation as mixed-integer program.

In Chapter 5 we focus on global placement in order to evaluate timing characteristics of macro placements accurately. We develop a geometric pre-processing to significantly speed up partitioning during BonnPlaceGlobal, the global placement module of BonnPlace. The proposed model of partitioning as minimum-cost flow instance is proven to have minimum cardinality. Furthermore we propose Self-Stabilizing BonnPlace, an extension of this partitioning-based placer by a force-directed placement framework. This approach provides the necessary versatility for optimizing routing and timing objectives at the same time.

The quality and performance of all presented algorithms is analyzed and discussed in Chapter 6, both on benchmark instances and on challenging real-world designs provided by our industry partner IBM. We demonstrate the increased efficiency of partitioning, particularly for the largest and most complicated test cases. The quality of Self-Stabilizing BonnPlace is confirmed both for congestion mitigation and timing optimization on challenging designs. Thereafter we analyze our mixed-size placement flow that combines timing-driven BonnMacro as well as Self-Stabilizing BonnPlace.

Chapter 2

Placement Problems

Placement is one of the most challenging and at the same time important steps in the physical design of VLSI chips. The quality of the placement has far-reaching effects on the whole design process.

In this chapter we elaborate the context of placement. We start by fixing a common notation in Section 2.1. Section 2.2 is devoted to introduce chip design notation and placement problems in particular constraints (Section 2.2.1) and objectives (Section 2.2.2). Finally we introduce the two placement problems which are most important for this thesis: Macro Placement (Section 2.3) and Global Placement (Section 2.4). We explain further details and review previous work.

2.1 Basic Definitions

This thesis deals with a multitude of objects representable by rectangles. In order to establish a common understanding, we fix the following notation:

Definition 2.1. A **rectangle** $r \subseteq \mathbb{R}^2$ is a set of the form $r = [x_0, x_1] \times [y_0, y_1]$ where $x_i, y_i \in \mathbb{R}$ for $i \in [\![2]\!]$ and $z_0 < z_1$ for $z \in \{x, y\}$.

We emphasize that by Definition 2.1 rectangles are always closed, axis-parallel, non-empty and have non-zero area.

We mostly deal with configurations of rectangles with fixed width and height:

Definition 2.2. The (anchor) **position** of a rectangle $r = [x_0, x_1] \times [y_0, y_1]$ is the lower left corner $(x_0, y_0) \in \mathbb{R}^2$. Conversely **placing** r at $p \in \mathbb{R}^2$ means translating r by $p - (x_0, y_0)$.

Note that by Definition 2.2 placing rectangle r at $p \in \mathbb{R}^2$ refers to the (anchor) position of r. Thus clearly the position of r placed at p is p itself.

For shapes possibly composed of multiple rectangles, we use the following terminology:

Definition 2.3. A **rectilinear shape** is a set of points $S \subseteq \mathbb{R}^2$ for which $k \in \mathbb{N}$ and rectangles $r_i \subseteq \mathbb{R}^2$ for $i \in [\![k]\!]$ exist with $\bigcup_{i \in [\![k]\!]} r_i = S$. The set of rectangles $\{r_i : i \in [\![k]\!]\}$ is called a **representation** of S.

Clearly the representation of rectilinear shapes is not unique.

Definition 2.4. The **intersection** of rectangles a and b is $a \cap b := \overline{(a \cap b)^\circ}$. Furthermore we define the intersection of two rectilinear shapes R and S as $\bigcup_i \bigcup_j r_i \cap s_j$ where r_i and s_j are representations of R and S.

Note that the interior in Definition 2.4 refers to the interior in $\mathbb{R}^2$. Consequently by Definition 2.4 we disregard one-dimensional areas in the intersection of rectangles. Thus for rectangles a and b, $a \cap b$ is a rectangle if and only if $a \cap b \neq \varnothing$. Note that Definition 2.4 for rectilinear shapes does not depend on the representation and thus is well defined.

Definition 2.5. For a given finite set $P \subseteq \mathbb{R}^2$, the **bounding box BB(P)** is the smallest set of the form $\mathrm{BB}(P) := [x_0, x_1] \times [y_0, y_1] \subseteq \mathbb{R}^2$ containing P where $z_i \in \mathbb{R}$ and $z_0 \leq z_1$ for $i \in [\![2]\!]$, $z \in \{x, y\}$.

Note that $\mathrm{BB}(P)$ is not necessarily a rectangle according to Definition 2.1 since $\mathrm{BB}(P)$ may have zero area.

For graph theory we use the notation of Korte and Vygen (2018).

2.2 Placement

Now we turn our attention towards Chip Design and placement in particular. We start with a couple of introductory definitions:

Definition 2.6. A **netlist** is a tuple (C, P, γ, N) of finite sets C and P of **circuits** and **pins** where $\gamma \colon P \to C \cup \{\square\}$ maps pins to either a circuit or the **chip image** $\square$ and N is a partition of P denoting the **nets**, i.e. N is a family of non-empty disjoint subsets of P whose union is P.

The circuits are also commonly referred to as **cells** or **gates**. Pins $p \in P$ with $\gamma(p) = \square$ are **preplaced pins** (also **ports** or **primary in- or output pins**).

Cells of a netlist not only encode abstract connectivity information but also are physical objects:

Definition 2.7. Let (C, P, γ, N) be a netlist. The **cell shape A(c)** of a cell $c \in C$ is a rectangle. Furthermore each pin $p \in P$ has a designated **offset off (p)** where $\mathrm{off}(p) \in \mathbb{R}^2$.

In practice cells are actually not required to have a shape which is a rectangle. Theoretically it is possible that a cell's shape consists of multiple rectangles forming a connected shape with axis-parallel borders. Depending on their size it is reasonable to include such cells in the model of Definition 2.7 either by considering an enclosing rectangle shape instead or by splitting such a cell into multiple tightly connected cells. But such cell shapes hardly ever appear – in particular not in any practical instance considered in this thesis. Thus modelling cell shapes as rectangles is a minor simplification.

Moreover pins actually also have a physical shape. But pin shapes are rather small compared to cell shapes. This especially applies to large cells which are the main focus of this work. Thus it is a common assumption in placement to consider pins as points in $\mathbb{R}^2$.

The pin offset $\mathrm{off}(p) \in \mathbb{R}^2$ of a pin $p \in P$ denotes the relative position of this point on $\gamma(p) \in C \cup \{\square\}$. For a cell $c \in C$ we denote the reference point on $\mathrm{A}(c)$ for all $\mathrm{off}(p)$ of pins $p \in \gamma^{-1}(c)$ as **anchor**. By translating $\mathrm{off}(p)$ accordingly we further consider the anchor of cell $c \in C$ to be the lower left corner of $\mathrm{A}(c)$. For having a uniform notation, we select $0 \in \mathbb{R}^2$ as the anchor of $\square$.

With this understanding of the physical realization of a netlist, we can define our primary objective:

Definition 2.8. A **placement** for a netlist (C, P, γ, N) is a map $l \colon C \to \mathbb{R}^2$.

We interpret the position $l(c)$ of cell $c \in C$ and placement l according to Definition 2.8 as the location of the anchor of c. By $\mathrm{A}_l(c) \subseteq \mathbb{R}^2$ we denote the **placed shape** of cell $c \in C$ with anchor translated to $l(c)$. This aligns the notion of placement for c with placement for rectangle $\mathrm{A}(c)$ (cf. Definition 2.2). We extend any placement l to $\square$ by $l(\square) := 0 \in \mathbb{R}^2$. This choice naturally extends the notation of placement to pins $p \in P$ via $l(p) := l(\gamma(p)) + \mathrm{off}\,(p)$.

In practice it is additionally usually allowed to rotate a cell by integer multiples of $\frac{\pi}{2}$ or flip a cell along a vertical axis. (Note that flipping along a horizontal axis can be expressed by this.) Recall that both operations preserve rectangles. We refer to the state of a cell with respect to flipping and rotating it as **orientation** or **flip-code**.

Technically the orientation of a cell corresponds to one of eight linear maps applied to $\mathrm{A}(c)$ all preserving a cell's anchor. In that sense also $\mathrm{A}_l(c)$ depends on the orientation of $c \in C$. In our application rotations by odd multiples of $\frac{\pi}{2}$ are always prohibited. Consequently we can change a cell's orientation without changing its placed shaped (by adapting the placement accordingly). For the remainder of this thesis however it is sufficient to consider fixed cell orientations unless explicitly stated otherwise.

2.2.1 Placement Constraints

We now describe the most important practical constraints for a placement which will lead to the major problem definitions in the following sections.

Definition 2.9. A **chip image** is a pair (A, B) of a rectangle A, the **chip area**, and a rectilinear shape $B \subseteq A$ of **blockages**.

Blockages encode areas of the chip image which are prohibited to be occupied by cells of the netlist. These blocked areas are reserved for special purposes in the design process. Most commonly blockages preserve space for parts of a chip designed by a different team. Also design stages following after placement may require additional space in predetermined regions e.g. for inserting buffers. Such buffer bays cause blockages as well.

The minimum requirements for a reasonable placement of a netlist on a given chip image are the following:

Definition 2.10. Consider a netlist (C, P, γ, N) and a chip image (A, B). We call a placement l **legal** if all of the following conditions are satisfied:

- Distinct cells do not overlap, i.e. $\mathrm{A}_l(c_0) \cap \mathrm{A}_l(c_1) = \varnothing$ for $c_i \in C$ and $i \in [\![2]\!]$ with $c_0 \neq c_1$.
- Each cell is placed in the chip area, i.e. $\mathrm{A}_l(c) \subseteq A$ for all $c \in C$.
- Cells are placed disjointly from blockages, i.e. $\mathrm{A}_l(c) \cap B = \varnothing$ for all $c \in C$.

Recall that disjointness in Definition 2.10 always refers to the notion stated in Definition 2.4.

In addition to the constraints formulated in Definition 2.10 placements have to meet further important criteria:

Most importantly, each cell can not be placed in arbitrary positions but rather has to be placed on a grid. More precisely a predetermined **anchor** offset on the cell has to be aligned with a grid which depends on the cell's orientation.

Grid constraints originate from the power supply for cells. Power is distributed over the entire chip image in a regular grid. These power grids have differing granularity depending

on the routing layer. The necessity for cells to pick up the power supply is modeled by grid constraints for placement.

The granularity of these grids differs greatly depending on the cell. Most of the cells in a netlist, the **standard cells**, are very small. All standard cells have a common width or height. These cells contain very limited wiring resources internally and consequently only require alignment with a fine power grid on the lowest routing layers. This grid has the same delta as the common width or height due to which cells are required to be placed in columns or rows. Meeting such grid constraints can usually be achieved by moving cells locally which is subject to standard cell legalization (cf. Section 2.2.3).

But this is not the case for all cells, particularly not for **macros** which are the primary focus of this thesis. In contrast to standard cells, such cells are larger and contain more routing resources internally. Macros are designed independently by different teams or even bought entirely from contractors. For reasons of aligning to power supply on upper routing layers, macro grids are relatively coarse. Due to both size and grid granularity of macros, such cells require special algorithms (cf. Section 2.3).

Another important placement constraint is encoded by movebounds. Such constraints restrict feasible locations for cells to be not only within the chip image, but rather even within a smaller rectilinear shape.

Definition 2.11. A **movebound m** is a rectilinear shape. For a given netlist (C, P, γ, N) **assigned movebounds (M, μ)** are a set of movebounds M with a map $\mu\colon C \to M$.

Note that according to Definition 2.11 we consider unconnected movebounds in particular. Movebounds restrict feasible placements in the following sense:

Definition 2.12. Consider a netlist (C, P, γ, N) with assigned movebounds (M, μ). A placement l is **legal with respect to (M, μ)** if and only if $\mathrm{A}_l(c) \subseteq \mu(c)$ for all $c \in C$.

Movebound constraints do not arise for reasons of manufacturability. They rather serve as powerful tool for designers to control the placement solution space. There are many reasons why such influence is desirable: First and foremost, it allows tools to adapt to the concept of a designer. Although this is not a mathematical objective, it is an important goal in practice. Especially at early design stages of a chip, not all parts of the netlist are worked out yet. Using movebounds allows designers to anticipate future netlist changes for their placements. Furthermore the organization of the chip design process also imposes constraints that can be modeled by movebounds. There are certain parts of the netlist requiring to be placed in close proximity, e.g. parts designed by the same team or operated on the same power-level.

Note that theoretically the concept of movebounds makes blockages obsolete. Any blockage could be modeled by subtracting it from all movebounds. This is not done in practice as it increases the complexity of the representation of movebounds. As also commonly done in literature, we use both the notion of blockages and movebounds here.

In practice movebounds are used as optional constraints, i.e. there are cells without any movebound whatsoever. We incorporate this into the notion of Definition 2.11 by adding an artificial movebound m for which m is the complete chip area. Cells without actual movebound instead can equivalently be assigned to m. This simplifies movebound considerations in this thesis.

Practical movebound constraints are also applied in two different flavors: either as **inclusive** or as **exclusive movebounds**. Definition 2.12 refers to inclusive movebounds.

Exclusive movebounds m in contrast require $\mathrm{A}_l(c) \cap m = \varnothing$ for all cells c with $\mu(c) \neq m$. In other words m is limited to be used by cells c with $\mu(c) = m$.

We can avoid the special case of exclusive movebounds in Definition 2.12 using the following pre-processing: For any two exclusive movebounds m_0 and m_1 we introduce blockages $m_0 \cap m_1$. This is no restriction since this intersection must not be utilized by any cell whatsoever. Subsequently for an exclusive movebound m we subtract m from m' for all other (inclusive and exclusive) movebounds m'. After doing so, exclusive movebounds are disjoint from all other movebounds and thus impose no extra constraints any longer. Thus all movebounds can be treated as (inclusive) movebounds according to Definition 2.12 from now on.

This pre-processing is computable in terms of intersections and complements of rectangles. Thus it has polynomial running time in particular. In the following we always assume that this pre-processing has been applied. As a consequence we avoid the special case of exclusive movebounds throughout this thesis without loss of generality.

2.2.2 Placement Objectives

Placement is one of the earliest steps of the chip design flow. Thus placement has to satisfy all the requirements of the subsequent design stages. Since many of these stages are difficult optimization problems themselves, their needs are usually modeled as objectives.

A primary placement objective is netlength:

Definition 2.13. For a given netlist (C, P, γ, N) with placement l and net weights $w\colon N \to \mathbb{R}_+$, the **(weighted) bounding box netlength BBL** is

$$\mathrm{BBL}(l) := \sum_{n \in N} w(n) \cdot \mathrm{BBL}\Big(\big\{\, l(p) : p \in n \,\big\}\Big),$$

where $\mathrm{BBL}(P)$ for finite $P \subseteq \mathbb{R}^2$ denotes half the perimeter of $\mathrm{BB}(P)$.

Minimizing netlength indirectly models various other optimization goals which we will discuss shortly. Furthermore it is widely considered in literature on placement tools as well as in publicly available benchmarks, e.g. the ISPD placement contests (Nam 2006; Nam et al. 2005).

Another important objective is *routability*. All pins of the nets have to be interconnected by wires located on various routing layers. Thereby both intolerable detours have to be avoided and involved distance requirements have to be obeyed. On the one hand, this requires short interconnections while on the other hand avoiding too densely packed wires.

In addition to routing, placements have to satisfy strict *timing* requirements. Electrical signals have to arrive in time at specified points on the chip in order to guarantee various interdependent parts of the netlist operate correctly all together. This can only be achieved with sufficiently short nets but also requires additional optimizations for individual connections, e.g. buffering and gate sizing.

A placement not only has to enable such optimizations, i.e. avoid packing cells too densely, but also should require as few as possible of these operations. Additional gates as well as faster gates consume more *power* which is an important objective to minimize.

To make things even more complicated, also manufacturing costs should be taken into account. Theoretically this can be achieved by facilitating placement on smaller chip areas by which more processors could be produced on a single wafer. But for the most part of

the design process the chip area is considered to be fixed. In practice the **yield** is a main focus, i.e. the fraction of produced chips that actually work as designed. This is primarily achieved by avoiding local routing configurations that can not be manufactured reliably.

All these objectives correlate to some extent to placements with small netlength. But considering this as the only optimization goal is not sufficient for the placement of modern microprocessors any longer.

In the following we explain routing and timing objectives in further detail. Both are of primary focus throughout this thesis and will subsequently be targeted directly.

Routing Objectives

Any placement defines a routing instance for which all pins of the nets have to be interconnected by wires located on various routing layers. Mathematically such a routing can be seen as a packing of 3-dimensional Steiner trees. Unfortunately the Steiner Tree Problem is known to be difficult (NP-hard due to Karp (1972), MaxSNP-hard due to Bern and Plassmann (1989)), which is why even routing a single net is complicated. In addition routing both has to obey involved distance requirements and avoid intolerable detours. Extensive information on routing can be found in Alpert, Mehta and Sapatnekar (2008) as well as Gester et al. (2013).

Nevertheless placements have to be routed eventually. In practice this not only requires a theoretical guarantee of routability, but rather a concrete routing algorithm implementation capable of finding the desired routing in reasonable time.

Such algorithms usually distinguish between **global** and **detailed** routing. During global routing a rough outline of the wiring is determined. Thereby density constraints for wiring capacities on all layers have to be obeyed and detours should be avoided while connections are allowed to overlap. The task of detailed routing is completing such a global wiring to an actual routing. This involves complicated distance requirements arising from the manufacturing process. Detailed routing is of subordinate interest for this thesis as placements usually can be optimized locally for detailed routability (e.g. during detailed placement).

Therefore the central routing objective during placement is **routing congestion**, which is an estimate for the wire density required for routing the placement (determined by a global routing algorithm). The underlying assumption that any placement with low congestion can actually be routed is met in practice.

From the perspective of placement there are two different types of congestion: If cells have been placed too far apart, nets become too long to be able to pack all needed wires on the chip. This type of **global congestion** is somewhat attacked by netlength minimization. But in addition placements have to avoid **local congestion** caused by tightly entangled circuits in close proximity. This effect will require special treatment and extra optimization steps.

Timing Objectives

Before we elaborate how to optimize for good timing characteristics, we need to explain how timing is actually modeled and evaluated.

Signal propagation times are computed within a connected, acyclic, directed graph on the pins of the netlist, the so called **timing graph**. Signals are propagated along the directed edges in the timing graph.

In practice, the timing graph actually is more complicated in order to be more efficient: It contains multiple nodes for each pin (corresponding to different types of timing analysis).

In addition the timing graph de facto also contains artificial vertices in order to avoid complete bipartite subgraphs. Since the efficiency of the timing engine is not of concern for this thesis, we may assume no such vertices have been added. Moreover duplicate nodes for the same pins can be handled as virtual pins in the netlist. In order to simplify notation, we thus assume that the vertices of the timing graph are exactly the pins of the netlist.

Constraints arise when signals need to have reached certain nodes in the timing graph. These ensure various parts of the netlist are able to interoperate jointly. Particular care is needed for storage elements which save and emit formerly computed bits. Such storage elements fall into two categories, those storing multiple bits (usually modeled by some macro) and those storing single bits only. Circuits of the latter category are called **latches** and appear frequently in any netlist. Non-memory circuits on the chip are also referred to as **logic**.

Saving and emission of storage elements takes place synchronously and in cycles, i.e. all those gates periodically start picking up the current signal and emitting a former one. This time interval is called **clock cycle** and its length is the **cycle time**. The timing graph expresses the signal propagation that is required to take place within any single clock cycle.

The cycle time directly affects the overall performance of the chip. Executing the same computations within shorter clock cycles results in faster chips. In practice an ambitious cycle time is fixed upfront and most part of the design process aims at making the set goal possible.

The time in each clock cycle at which a signal reaches a given node v in the timing graph is the **arrival time at(v)**. Similarly the latest possible arrival time at node v by which the cycle time is not exceeded is the **required arrival time rat(v)**. The amount of time left between required and actual arrival time is denoted by **slack(v)**, i.e. $\operatorname{slack}(v) := \operatorname{rat}(v) - \operatorname{at}(v)$.

Those vertices v in the timing graph with predefined arrival times $\operatorname{at}(v)$ or required arrival times $\operatorname{rat}(v)$ are called **timing startpoints** or **timing endpoints**. Not all nodes are start- or endpoints, but e.g. those corresponding to memory elements usually are of this kind, which particularly applies to latches.

The constant (required) arrival times for timing start- and endpoints are known as **assertions**. Note that by definition assertions are independent from the placement. Timing assertions define the contracts by which all gates of the chip are able to interoperate.

For any fixed placement all values $\operatorname{at}(v)$ and $\operatorname{rat}(v)$ can be computed by **timing engines** based on the assertions. The resulting (required) arrival times crucially depend on the used **delay function** which denotes the time needed for a signal to traverse an edge in the timing graph. This computation considers further properties of the nodes in the timing graph which are not important for our purpose. We will refer to a fixed timing engine with fixed delay model as **timing model**.

There are multiple delay models varying in accuracy, complexity and underlying assumptions. The presented evaluation context is known as static timing analysis and goes back to Hitchcock, Smith and Cheng (1982) and Kirkpatrick and Clark (1966). The simplest models therein neglect any delay resulting from wires entirely or assume a linear delay proportional to the distance between connected pins (e.g. Otten 1998). More accurate models view the timing graph as electrical network of resistances and capacitances resulting in quadratic delay functions (Elmore 1948). With more computational effort

even preciser models can be derived, e.g. RICE (Ratzlaff and Pillage 1994), MAISE (Liu and Feldmann 2008–2008) or statistical timing (Visweswariah et al. 2006).

Depending on the stage in the chip design process and consequently the required accuracy either of the mentioned delay models is used in practice. For more details on timing analysis in general, the aforementioned delay models in particular and also further higher order delay models we refer to Sapatnekar (2004) and Alpert, Mehta and Sapatnekar (2008, 546 ff.).

For this thesis, we focus on the **virtual timing model** (cf. Alpert et al. 2006; Otten 1998). This model assumes that all nets can be buffered optimally and therefore the delay along any net is linear in the distance of the endpoints. The coefficient involved depends on the layer the net is predominantly assigned to. This model copes well with various uncertainties during placement e.g. topologies remaining to be selected for multi-terminal nets or buffers to be inserted. Virtual timing is an optimistic model that bounds actual delay from below and estimates it reasonably well in practice, particularly for the most critical nets (Alpert et al. 2006). In addition it can be computed and updated in linear time by propagation in the timing graph in topological order. This is why virtual timing is a good model to consider during placement.

For any path P in the timing graph, $\mathrm{E_c}(P)$ denotes the **circuit internal edges** of P i.e. those connecting vertices corresponding to the same circuit. Analogously $\mathrm{E_w}(P) := \mathrm{E}(P) \setminus \mathrm{E_c}(P)$ stands for the **wiring edges** whose endpoints eventually have to be connected by actual wires.

Furthermore for given path P in the timing graph and placement l of the netlist, $\mathbf{d}(P, l)$ denotes the **virtual timing model delay**. By assumption on this delay function, it is composed of two components: $\mathrm{d_c}(P) := \sum_{e \in \mathrm{E_c}(P)} \mathrm{d}(e, l)$ stands for the **circuit internal delay** of P. Note that by assumption on d, $\mathrm{d_c}$ is independent from the placement l. Furthermore

$$\mathrm{d_w}(P) := \sum_{e \in \mathrm{E_w}(P)} \mathrm{d}(e, l) = \sum_{(v,w) \in \mathrm{E_w}(P)} \alpha_{(v,w)} \cdot \|l(w) - l(v)\|_1$$

denotes the **wiring delay** where $\alpha_e \in \mathbb{R}$ is the **coefficient** depending on the assignment of wiring edge e.

The primary timing objective during placement is maximization of the **worst slack**, i.e. the minimum slack of any node in the timing graph. The worst slack denotes how much later all signals could arrive without being too late. In practice assertions are often ambitious, in which case positive slack often can not be achieved.

Only a small fraction of nodes in the timing graph contributes to the worst slack. Thus usually secondary objectives, to which larger parts of the netlist contribute, are considered in addition to worst slack. The most prominent among these is the **figure of merit (FOM)** which specifies the sum of slacks at all endpoints in the timing graph. There is also a variant of the FOM, the so called **pFOM**, for which the slack of even more nodes is taken into account. For the pFOM all slacks of endpoints and those nodes v with $|\delta^+(v)| > 2$ are summed up.

2.2.3 Placement Problem Variants

For various stages in the chip design process, different types of placement problems arise in practice and have been studied in literature.

Chronologically first considered in the order of a hierarchical design process is the **FLOORPLANNING PROBLEM**. Here only few but large circuits are placed which corre-

spond to macros or units that will be designed separately. For such units, only a very rough sketch is known. Often neither the concrete size, nor the precise aspect ratio nor detailed pin positions have been determined yet. All these details are subject to optimization under the constraint that copies of the same unit share identical characteristics. With so much uncertainty only netlength can be optimized during floorplanning.

Once concrete shapes for all cells in the netlist have been settled upon, placement requirements resulting from the hierarchical design process are usually formulated as movebound constraints (cf. Definitions 2.11 and 2.12) for all further placement steps. The first of these placement stages is the **Macro Placement Problem**. At this stage positions for all large cells in the netlist, so called **macros**, have to be determined, which subsequently remain more or less fixed for the remainder of the flow.

The involved constraint of placing macros disjointly is challenging in practice since large fractions of the chip can be occupied by few macros. Macro Placement focuses on positioning macros in a way that allows extending the placement to the complete netlist nicely. This primarily targets approximations of netlength since all actual objectives are very hard to estimate at this point.

Further placement goals can only be incorporated by reiterating the complete design flow for few, manually selected promising placement changes. Consequently at this point large safety margins are required for the actual placement of logic. This allows reacting to any potential issues later in the flow without the need of redesigning the whole chip. On the other hand, it likely is overly conservative for the most part.

For fixed macro positions, the next step is placing the remainder of the netlist. In the **Global Placement Problem** this is addressed while neglecting local grid constraints. Instead of grids Global Placement considers density constraints to ensure finding legal on-grid positions afterwards can be achieved easily. While netlength has been the predominant objective of Global Placement for many years, routability and timing characteristics are becoming increasingly important in recent technology nodes.

In the final placement stage, the **Detailed Placement Problem**, local placement issues are addressed. For this purpose all non-standard circuits are assumed to be already fixed and only considered as blockages. Input for Detailed Placement usually is a feasible global placement. The task of this placement step is finding a placement satisfying local constraints (e.g. grids or additional constraints for pin accessibility) while perturbing the input as little as possible. A common objective is total squared movement.

The problem of snapping standard circuits into their respective grids imposed by the circuit rows is also referred to as **Legalization Problem**. In contrast to the Macro Placement Problem, finding any legal solution is usually easy. This is due to the facts that in this case cells share a common width or height and sufficient whitespace is guaranteed via Global Placement's density constraints.

We refer to the combination of macro and global placement followed by legalization as **Mixed-Size Placement Problem**.

With this preliminary introduction to placement in general, in the following two sections we will elaborate details of two components of mixed-size placement centrally considered in this thesis.

2.3 Macro Placement

The Macro Placement Problem is central to this thesis. A solution to this problem, a **macro placement**, is a legal placement map restricted to macro cells. Good macro placements can easily be extended to good placements for the entire netlist.

Consequently it is reasonable to incorporate special macro handling into existing global placement algorithms. This has been done for min-cut based Capo (Roy et al. 2006), force-directed FastPlace (Viswanathan, Pan and Chu 2006, 2007) and Kraftwerk (Eisenmann and Johannes 1998; Spindler, Schlichtmann and Johannes 2008), as well as non-linear, analytical APlace (Kahng and Wang 2005), mPL6 (Chan et al. 2006, 2007), ePlace (Lu, Zhuang et al. 2015; Lu, Chen et al. 2015), NTUPlace3 (Chen, Jiang et al. 2008) and Maple (Kim et al. 2012). Further details of these and other global placement algorithms are presented in Section 2.4.

Placing standard cells and macros simultaneously allows to model objectives accurately. Since all placeable objects are considered at the same time, such algorithms always have a global view. But on the flipside, achieving legal macro positions is complicated.

Some of the mentioned approaches, e.g. FastPlace, mPL6 and Maple, handle this requirement in a rough pre-legalization estimate. While this guarantees legal positions, these algorithms suffer from suboptimal relative positions that need to be fixed early. Analytical placers incorporate disjointness requirements into the objective. This often fails to find an actual legal placement, particularly on instances where macros occupy the majority of the chip area. In this case additional techniques are required.

Partitioning based placement algorithms on the other hand can easily guarantee legality by assigning macros to different bins of the partitioning grid. The central challenge thereby is to support fine grids with bins smaller than macros. This has been tackled by tentatively blocking multiple bins for macros, which later can be revised (Adya et al. 2004) or even completely undone (PolarBear: Cong, Romesis and Shinnerl 2005). Khatkhate et al. (2004) instead propose to enhance the partitioning paradigm and consider fractional cuts in their placer FengShui.

There is also a different approach for deriving initial macro positions with potential overlaps. Instead of working on the netlist with macros, it is possible to consider an artificial netlist without such large cells.

Adya and Markov (2005) and Doll, Johannes and Antreich (1994) first proposed replacing macros with fixed outlines by many small and tightly interconnected artificial cells. This technique is commonly referred to as **macro shredding** and the artificial cells are called **macro fragments**. Based on a placement of the shredded netlist macro positions can finally be inferred from their respective fragments, e.g. their center of gravity. This step is known as **macro reassembly**.

It is possible to apply all existing algorithms for the Global Placement Problem to the netlist with shredded macros (cf. Section 2.4). With good global placement algorithms and accordingly adapted shredded netlist (e.g. fragment sizes and interconnections) the resulting macro positions only overlap locally but are well spread globally. This concept has also been used in BonnMacro (Brenner 2007), the macro placement component of BonnPlace (Brenner, Struzyna and Vygen 2008), as well as in ComPLx (Kim and Markov 2012).

In a second step any potential macro overlaps need to be resolved while perturbing the input as little as possible. This is also referred to as **Macro Legalization Problem**. Recall that all cell shapes in fact are modeled as rectangles by Definition 2.7. Consequently Macro Legalization leads to the geometric problem of *packing* rectangles.

If two rectangles are placed without overlaps their disjointness can be certified by a spatial relation:

Definition 2.14. There are four **spatial relations**: $\lhd$ (left) , $\rhd$ (right) , $\triangledown$ (below) and $\triangle$ (above) . For given rectangles r_i where $i \in [\![2]\!]$ and $\sim \in \{\lhd, \rhd, \triangledown, \triangle\}$ we write $r \sim r'$ if "$\sim$" holds for the positions of r_i, e.g. $r_0 \lhd r_1$ in case of $\mathrm{xmax}(r_0) \leq \mathrm{xmin}(r_1)$.

The spatial relation of two rectangles r_i depends on the placement of both r_i or more precisely on the relative placement of them. Finding a disjoint placement in particular includes selecting one spatial relation for each pair of rectangles that holds in the placement.

For simply packing rectangles disjointly, any spatial relation is sufficient for any pair of rectangles. In our application though, it is valuable to restrict the available choices of relations. Therefore we consider **allowed relations** $\varphi\colon R \times R \to \mathcal{P}(\{\lhd, \rhd, \triangledown, \triangle\})$. We always assume φ is anti-symmetric, i.e. $x \in \varphi(r, r')$ if and only if $\overline{x} \in \varphi(r', r)$ where $\overline{\lhd} := \rhd$, $\overline{\triangledown} := \triangle$ and vice versa.

Moreover we even require relation dependent minimum distance constraints for all rectangle pairs (in order to model macro grid constraints less pessimistically). More precisely, a relation $\sim \in \{\lhd, \rhd, \triangledown, \triangle\}$ holds with **additional space** $s \in \mathbb{R}$ if and only if the $\sim$-defining inequality holds with slack at least s.

With the introduced additional notation we formalize the standard problem of packing rectangles as follows:

Rectangle Packing Problem

Instance: A finite set of rectangles R, allowed relations $\varphi\colon R \times R \to \mathcal{P}(\{\lhd, \rhd, \triangledown, \triangle\})$, required spacing $\psi\colon R \times \{\lhd, \rhd, \triangledown, \triangle\} \times R \to \mathbb{R}_+$, feasible area rectangles $A(r)$ for $r \in R$.

Task: Find placement $l\colon R \to \mathbb{R}^2$ s.t.

- $\forall r_0, r_1 \in R$ placed according to l with $\varphi(r_0, r_1) \neq \varnothing$:
  $\exists \sim \in \varphi(r_0, r_1)$ s.t. $r_0 \sim r_1$ with additional space $\psi(r_0 \sim r_1)$
- $\forall r \in R$: r placed at $l(r)$ is contained in $A(r)$

or decide that no such locations l exist at all.

Recall that the Rectangle Packing Problem refers to the placement of rectangles introduced in Definition 2.2.

Note that the presented formulation of the Rectangle Packing Problem also includes rectangles with prescribed positions. Those can be modeled by appropriate feasible areas allowing the desired placement only, i.e. $A(r) := r$. We often call rectangles with such restrictive feasible areas **fixed**.

Clearly Rectangle Packing contains the problem of placing rectangles disjointly via $\varphi :\equiv \{\lhd, \rhd, \triangledown, \triangle\}$. As part of the extension with restricted relations, in particular no required relation is possible, i.e. $\varphi(r_0, r_1) = \varnothing$. In this case r_0 and r_1 may overlap in a feasible solution l. We will explicitly use this possibility in Chapters 3 and 4.

The presented form of the Rectangle Packing Problem is a decision problem only. There are many variants of this problem studied in literature with different objectives. Motivated by the chip design application, minimizing netlength and particularly movement from an illegal placement is of special interest. Moreover Rectangle Packing is a special case of Floorplanning with fixed outlines. This is why minimizing the area of

an enclosing rectangle also has been studied. We will generalize the Rectangle Packing Problem in order to model timing on macros in Chapter 3.

The Rectangle Packing Problem is a notoriously difficult optimization problem (cf. Theorem 3.16). Consequently large instance can not be solved to optimality in practice.

Many practical approaches start with representations of possible placement topologies, e.g. sequence pairs (Jerrum 1985; Murata et al. 1996), bounded sliceline grids (Nakatake et al. 1996), Q-sequences (Zhuang et al. 2002–2002), O-trees (Guo, Cheng and Yoshimura 1999; Takahashi 2000), B*-trees (Chang et al. 2000), corner sequences (Lin, Chang and Lin 2003), transitive closure graphs (Lin and Chang 2005), or adjacent constraint graphs (Zhou and Wang 2004). These and further representations have been surveyed in Chen and Chang (2008) and Young (2008). For all schemes there are exponentially many representations for solutions to Rectangle Packing. The tightest bounds on the cardinality of representations have recently been shown by Silvanus and Vygen (2017).

Naturally an optimum packing representation can not be enumerated in practice. To circumvent this many heuristics have been applied: Almost all authors of previously mentioned representations apply simulated annealing, but also dedicated local search algorithms (e.g. Adya and Markov (2003), Janiak, Kozik and Lichtenstein (2010) and Moffitt et al. (2006) on constraint graphs or Guo, Cheng and Yoshimura (1999) on O-trees), genetic algorithms (e.g. Tang and Sebastian (2005) on O-trees or Fernando and Katkoori (2008) on sequence pairs) as well as particle swarm optimization (e.g. Sun et al. (2006) on B*-trees) have already been studied. Various heuristics for area minimization were surveyed by Bortfeldt (2013). Other (unsurprisingly popular) heuristics simply mimic the common designer's approach of moving macros to the corners (e.g. MP-trees by Chen, Yuh et al. (2008)).

A different approach is heuristically selecting small packing sub-instances whose optimum solutions can be combined to an overall solution. Onodera, Taniguchi and Tamaru (1991) started this using a branch-and-bound algorithm for area minimization. This was subsequently improved by Korf, Moffitt and Pollack (2010) via a formulation as meta constraint satisfaction problem which improved pruning. Similar strategies have also been generalized and improved by Funke, Hougardy and Schneider (2016) for the version of netlength minimization. An evolution of their Spark implementation is a fundamental component of today's BonnMacro.

Depending on input placement quality, minimizing movement from an initial placement during Macro Legalization is not sufficient. Yan, Viswanathan and Chu (2014) proposed to include clusters of standard cells with variable outline in legalization with their placer Flop. This results in a problem variant very similar to actual Floorplanning.

All mentioned macro placement approaches primarily target netlength as objective. Timing requirements for non-standard cells have been translated to netweighting heuristics by Gao, Vaidya and Liu (1992), but only in case the packing problem actually is easy. This concept is very similar to common timing-driven standard cell placement paradigms (cf. Section 2.4).

2.4 Standard Cell Placement

Solving the Global Placement Problem is a crucial step in the physical design of VLSI chips. The quality of the placement has far-reaching effects on the whole design process.

For Global Placement macro positions are fixed and the remaining standard cells of roughly equal height have to be arranged on the chip. Thereby cell grids are ignored.

Moreover small overlaps are also allowed, which expectedly can be easily resolved by subsequent Legalization of standard cells (cf. Section 2.2.3). Nevertheless movebound and density constraints (e.g. measured on a regular grid across the chip) have to be respected.

Finding any feasible global placement satisfying these constraints can usually be accomplished easily. Since chips contain sufficient unused space, the packing problem becomes simple. But nevertheless optimizing for good placements is challenging, in particular on modern designs with multiple millions of cells.

Global placement tools typically minimize the total interconnect length. In addition, they have to guarantee routability of their placements while meeting tight timing constraints on individual signals.

Most state-of-the-art placement tools are analytical algorithms. They start off with a placement minimizing a smoothed approximation of the half-perimeter wirelength, but allowing cells to overlap arbitrarily (Brenner and Vygen 2008). Afterwards the task is to reduce the overlapping of cells.

There are different ways to work towards an overlap-free placement. In principle, two major approaches have been applied in practice. Actual implementations often combine various methods of either paradigm.

In **partitioning-based** algorithms, the chip area is divided into bins and the algorithm ensures that no bin contains too many cells. Those approaches often are highly efficient and meet density constraints very accurately. The intrinsic challenge for these approaches is the partitioning of cells into bins during which original objectives can only be considered indirectly.

Other approaches iteratively apply artificial forces pulling cells from crowded areas of the chip towards free space. Forces can be seen as punishments for the illegality of an analytical placement. Those **force-directed** placers vary widely in their force computation and modulation. Usually they require many iterations in order to obey density constraints. In contrast to partitioning-based algorithms though, each iteration has a global view on the actual objective.

Analytical placement tools start with an arrangement of cells optimizing a smooth approximation of the resulting interconnect netlength if all constraints are ignored. In particular, cells will overlap in the obtained initial solution. Objectives thereby considered range from super-linear (e.g. quadratic by Kleinhans et al. 1991), over log-sum-exp (Naylor, Donelly and Sha 2001) to the weighted-average netlength model (Hsu, Balabanov and Chang 2013). With such a solution, the goal is to find a placement subject to density constraints which is almost as good as the initial arrangement.

Force-directed placers perform multiple placement iterations. Successively increasing forces pull cells out of overfull areas of the chip in order to achieve legality. This was first applied by Kraftwerk (Eisenmann and Johannes 1998; Spindler, Schlichtmann and Johannes 2008) successfully. Here forces are computed locally as gradient of the density violation.

Forces are computed differently in RQL (Viswanathan et al. 2007), FastPlace (Viswanathan, Pan and Chu 2006, 2007) as well as in SimPL (Kim, Lee and Markov 2012, 2013) including its subsequent evolutions SimPLR (Kim et al. 2011), Ripple (He et al. 2011), ComPLx (Kim and Markov 2012), Maple (Kim et al. 2012) and Polar (Lin and Chu 2014; Lin et al. 2015). Here a rough legalization is computed and each cell is pulled towards its legalized position. Different to Lagrangian multipliers, forces do not penalize illegality directly but punish the difference to one particular legal placement

which has been computed previously. Hence the quality of the overall algorithm heavily depends on the legalization itself. The legalization applied in SimPL-based papers is fast but not very sophisticated. As a result cells can be pulled only slightly towards their former legal positions. Consequently a large number of iterations is necessary.

All algorithms mentioned so far introduce forces as artificial connections subsequently smoothed by the respective analytical interconnect minimizer. Other analytical approaches directly incorporate smooth approximations for the opposing placement goals density and legality. APlace (Kahng and Wang 2005) does this based on the patent of Naylor, Donelly and Sha (2001). In mPL (Chan et al. 2006, 2007) density constraints are globally smoothed using the Helmholtz equation. NTUPlace4 (Hsu et al. 2011) uses quadratic functions to penalize violations of a locally smoothed density function and of a smoothed congestion estimation. More recently, ePlace (Lu, Zhuang et al. 2015; Lu, Chen et al. 2015) successfully models placement instances as electrostatic systems translating density violations to the system's potential energy.

In contrast to those ideas, partitioning-based algorithms recursively subdivide the chip area while assigning circuits to the respective regions. Capo 10.5 (Roy et al. 2006) does this while minimizing the induced cut in the netlist hypergraph. Xiu and Rutenbar (2007) iteratively determine a non-uniform grid that optimizes density violations and scale these assignments into a regular grid. Also BonnPlace (Brenner, Struzyna and Vygen 2008; Struzyna 2013) contains a partitioning-based global placement algorithm. BonnPlace uses flow-based partitioning which efficiently solves the global partitioning problem via a series of min-cost flow problems.

In order to improve running time and solution quality many global placers use clustering. A **cluster** is a group of cells that will be handled as single, artificial object by the algorithm. Clustering can be integrated well into iterative placement paradigms (cf. Chen et al. 2003; mFAR: Hu and Marek-Sadowska 2005; FastPlace 3.0: Viswanathan, Pan and Chu 2007).

Above mentioned and many further global placement algorithms have been surveyed by Nam and Cong (2007), Markov, Hu and Kim (2015) and Alpert, Mehta and Sapatnekar (2008, Chapters 15 – 18).

With ever increasing design complexity, optimizing netlength is insufficient for successful routing. In order to mitigate routing failures, congestion-driven global placement algorithms require additional techniques.

Placements with small netlength are a good starting point that globally honors routing resources. Most placer can reliably avoid local routing congestion by spreading cells apart that contribute to routing hotspots. A placement of locally lower density provides more routing resources for fewer nets and hence is easier to route. At the same time density reduction needs to be traded carefully against congestion mitigation: longer nets require more routing resources which needs to be prevented on a global scale. Spreading cells can either be achieved by adjusting the target density locally or by artificially inflating cells (Hou et al. 2001).

A crucial difference between various congestion-driven placement paradigms is the actual congestion estimation. The earliest approaches by Cheng (1994) use a probabilistic routing estimation. This approach has been refined and extended to account for pin densities in a former version of BonnPlace (Brenner and Rohe 2003).

Modern approaches rely on fast modes of actual global routing engines: This is the case for many routability-driven evolutions of SimPL (Kim, Lee and Markov 2012) including SimPLR (Kim et al. 2011) based on BFG-R (Hu, Roy and Markov 2010) as well as

POLAR2 (Lin and Chu 2014) based on FASTROUTE4 (Xu, Zhang and Chu 2009). Recent versions of BONNPLACE are based on BONNROUTEGLOBAL (Müller, Radke and Vygen 2011).

There are also different approaches which incorporate routing estimates more directly: The congestion-driven version ROOSTER (Roy and Markov 2007) of CAPO (Roy et al. 2006) considers cuts in the netlist based on Steiner tree estimates in order to avoid congestion. NTUPLACE4 (Hsu et al. 2011), the congestion-driven NTUPLACE3 (Chen, Jiang et al. 2008), incorporates congestion penalties into the overall smoothed objective function.

Further literature on congestion-driven global placement has been surveyed by Adya and Yang (2008) and Markov, Hu and Kim (2015).

But even a placement that can be routed easily is not satisfactory for modern technology designs. In addition placements need to ensure that various parts of the netlist can interoperate jointly. To meet these timing requirements, certain connections must not be too long.

A very common approach is iteratively computing multiple placements while incentivizing the placer to shorten nets with negative slack. This is primarily done using **net weights**, i.e. coefficients of the nets in the objective.

Effective net weighting schemes have already been proposed by Burstein and Youssef (1985) and Dunlop et al. (1984). Kong (2002) extends this idea and emphasizes shared critical nets by path counting. Weights can also be inferred as Lagrangian multipliers for relaxed timing constraints (Hamada, Cheng and Chau 1993; Srinivasan, Chaudhary and Kuh 1992; Szegedy 2005; Wu and Chu 2017).

Other approaches restrict the length of critical nets (Zhong and Dutt 2002). For small instances it is also possible to replace an analytical placement with timing-driven input locations as determined by a linear program. This has been done in ALLEGRO (Jackson and Kuh 1989). In order to apply this to larger instances, Luo, Newmark and Pan (2006) pursue a hybrid approach: They use net weights as well as local linear programs ensuring good timing characteristics.

Similar ideas have been used in many of the aforementioned global placement algorithms. KRAFTWERK (Eisenmann and Johannes 1998) even use net weights based on Elmore delay.

More literature on timing-driven global placement has been summarized by Markov, Hu and Kim (2015) and Pan, Halpin and Ren (2008).

Chapter 3

Timing-Driven Rectangle Packing

In this thesis we develop a tool to optimize timing characteristics of macro placements. This chapter lays the theoretical foundation for this purpose. We introduce an extension of the classical Rectangle Packing Problem motivated by our application in Macro Placement.

In Section 3.1 we first translate timing information into geometric constraints which leads to the Timing-Driven Rectangle Packing Problem. Section 3.2 elaborates how to compute this model. We analyze basic properties of the generalized packing problem in Section 3.3, before we focus on certain instances in Section 3.4. Here we prove the essential Theorem 3.34, an efficient, exact algorithm for this subclass of instances.

Afterwards in Section 3.5, we discuss techniques for representing our geometric constraints more efficiently. Corollary 3.42 proves that for any pair of rectangles a single generalized distance constraint suffices. We close this chapter by a comparison of classical and generalized Rectangle Packing for fixed spatial relations in Section 3.6.

3.1 Distance-Bound Model

Macro Placement is an important problem in the chip design flow, but at the same time a very difficult one (cf. Section 2.3). This particularly holds with regard to the timing objective. Even simply evaluating timing properties of a given macro placement involves solving a timing aware Global Placement Problem. In order to still optimize timing characteristics of macro placements, we will make some simplifying considerations.

Recall that we are assuming a virtual timing environment which has been explained in Section 2.2.2 (page 8 ff.). In this model we distinguish between gate-internal (E_c, d_c) and wiring (E_w, d_w) edges and delay. The gate-internal delay thereby is constant, while the correlation of distance and wiring delay is linear and layer-dependent.

Throughout this section we consider a fixed instance of the Macro Placement Problem with arbitrary but also fixed placement l. More precisely this includes a netlist with dedicated macro cells, blockages and movebounds as well as a timing environment, particularly a timing graph. Recall that we assume without loss of generality the nodes of the timing graph are exactly the pins of the netlist.

Consider two pins s and t of macros as vertices in the timing graph. In the following we denote by α the (smallest) delay-per-distance coefficient of the fastest layer. Consequently for any s-t-path P in the timing graph

$$\begin{aligned}\mathrm{d}(P,l) = \mathrm{d_w}(P,l) + \mathrm{d_c}(P) &= \sum_{e \in \mathrm{E_c}(P)} \mathrm{d_c}(e) + \sum_{(u,v) \in \mathrm{E_w}(P)} \alpha_{(u,v)} \cdot \|l(v) - l(u)\|_1 \\ &\geq \sum_{e \in \mathrm{E_c}(P)} \mathrm{d_c}(e) + \sum_{(u,v) \in \mathrm{E_w}(P)} \alpha \cdot \|l(v) - l(u)\|_1.\end{aligned}$$

Using $l(t) - l(s) = \sum_{(u,v)\in \mathrm{E}(P)} (l(v) - l(u))$ we derive

$$\sum_{(u,v) \in \mathrm{E_w}(P)} \|l(v) - l(u)\|_1 \geq \|l(t) - l(s)\|_1 - \left\| \sum_{(u,v) \in \mathrm{E_c}(P)} l(v) - l(u) \right\|_1 \tag{3.1}$$

$$\geq \|l(t) - l(s)\|_1 - \sum_{(u,v) \in \mathrm{E_c}(P)} \|l(v) - l(u)\|_1 \tag{3.2}$$

by the triangle inequality. Thus we see the following lower bound on $\mathrm{d}(P,l)$

$$\mathrm{d}(P,l) \geq \alpha \|l(t) - l(s)\|_1 + \sum_{(u,v) \in \mathrm{E_c}(P)} \Big[\mathrm{d_c}(u,v) - \alpha \|l(v) - l(u)\|_1 \Big]. \tag{3.3}$$

Most importantly, the delay bound Equation (3.3) only depends on the placement l of the endpoints s and t of P. Both $\mathrm{d_c}$ and $\|l(v) - l(u)\|_1$ are constant for circuit internal edges of the timing graph. To emphasize this, we further use the notation $\|u,v\|_1 := \|l(v) - l(u)\|_1$.

Since this lower bound holds for any s-t-path P, we are particularly interested in the most restrictive bound. This motivates the following definition:

Definition 3.1. Consider an instance of Macro Placement with placement l and two pins s and t in the timing graph. We denote the **maximum path delay** $\mathrm{d}_l(s,t)$ between s and t as

$$\mathrm{d}_l(s,t) := \max\{\, \mathrm{d}(P,l) : P \text{ is an } s\text{-}t\text{-path} \,\}.$$

We furthermore define the **circuit internal delay estimate** $\mathrm{cd}(s,t)$ between s and t as

$$\mathrm{cd}(s,t) := \max \left\{ \sum_{(u,v) \in \mathrm{E_c}(P)} \Big[\mathrm{d_c}(u,v) - \alpha \|u,v\|_1 \Big] : P \text{ is an } s\text{-}t\text{-path} \right\}.$$

We abbreviate $\max \varnothing = -\infty$.

Please note that Definition 3.1 deliberately makes use of the weaker bound Equation (3.2). This is done as Corollary 3.13 proves that the stronger bound Equation (3.1) would lead to an alternative version of $\mathrm{cd}(s,t)$ that is NP-hard to compute. This will be discussed in Section 3.2 in detail.

We summarize the previous motivation in the following lemma:

Lemma 3.2. *For any two pins s, t in an instance of* Macro Placement *and arbitrary placement l*

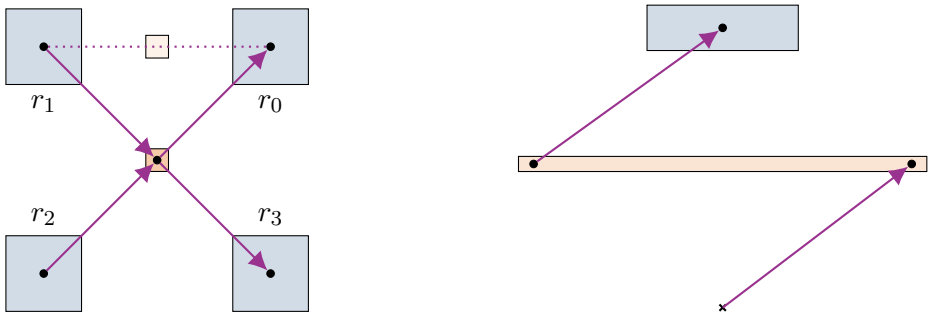
$$\mathrm{d}_l(s,t) \geq \alpha \|l(t) - l(s)\|_1 + \mathrm{cd}(s,t).$$

Proof. If t is not reachable from s, there is nothing to show since $\mathrm{cd}(s,t) = \mathrm{d}_l(s,t) = -\infty$. Otherwise, we consider any s-t-path P certifying the value of $\mathrm{cd}(s,t)$, i.e. maximizing $\sum_{(u,v)\in \mathrm{E_c}(P)} \big[\mathrm{d_c}(u,v) - \alpha \|l(v) - l(u)\|_1\big]$. The stated bound on $\mathrm{d}_l(s,t)$ follows from Equation (3.3) applied to P. ■

This lower bound contains several simplifying assumptions, i.e. it is not always tight even for slack-optimum placements. To start with, it assumes that *simultaneously* all paths are shortest. For an example where this may not be achievable see Figure 3.1(a). Here we assume the blue macros r_i for $i \in [\![4]\!]$ are de facto fixed in their position e.g. for reasons of other connections omitted for simplicity. For appropriate at- and rat-values the depicted solution for the orange cell in the middle is the unique optimum regarding slack. Despite that e.g. the r_1-r_0-path is not shortest as suggested by the lightly shaded alternative position at the top. But this is what our lower bound assumes.

Additionally Lemma 3.2 expects no detours from circuit internal edges. But also this aspect can be violated as depicted in Figure 3.1(b). In this example the orange circuit spans a large distance, but thereby makes the highlighted path longer than estimated. If it was rotated by $\frac{1}{2}\pi$, our lower bound would be accurate. In practice however, circuits that are not selected as macros will be small. This is why this effect only plays a minor role in practice.

Lastly we assumed a uniform layer assignment to the fastest layer. This will hardly be realizable for reasons of routing congestion. Consequently the actual delay will often be longer than estimated as slower layers have to be utilized as well.



(a) An instance with a worst r_0-r_1-path that is not shortest.

(b) An instance where circuit offsets increase delay on the critical path.

Figure 3.1: Two examples visualizing that Lemma 3.2 only defines a lower bound on delay (macros in blue; non-macros in orange).

Using the lower bound of Lemma 3.2 we can state the following necessary condition for achieving a certain slack:

Proposition 3.3. *Consider any instance of* Macro Placement. *If there is a placement l with slack $r \in \mathbb{R}$, then*

$$\|l(t) - l(s)\|_1 \leq \alpha^{-1}\big[\mathrm{rat}(t) - \mathrm{at}(s) - r - \mathrm{cd}(s,t)\big]$$

for all startpoints s and endpoints t in the timing graph.

Proof. Since l has slack r, in particular $\mathrm{d}_l(s,t) \leq \mathrm{rat}(t) - \mathrm{at}(s) - r$. Using Lemma 3.2 concludes the proof. ■

But clearly, the inverse implication is not necessarily true. As already discussed in Figures 3.1(a) and 3.1(b), Lemma 3.2 is only a necessary condition.

In the following, we want to translate distance requirements as considered in Proposition 3.3 to pure rectangles. Therefore we make the following definitions:

Definition 3.4. Consider a finite set of rectangles R. We define **distance bounds** as an undirected graph $G = (V, E, \Psi)$ on vertices $V = R \dot\cup \{\square\}$ with associated functions $\mathrm{b}\colon E \to \mathbb{R}$ and $\mathrm{o}_e\colon [\![2]\!] \to \mathbb{R}^2$ for $e \in E$. For brevity we also write G_b for such $(G, \mathrm{b}, \mathrm{o})$.

Distance bounds G_b are defined as general undirected graphs in the notation due to Korte and Vygen (2018, Chapter 2.1): $\Psi\colon \mathrm{E}(G) \to \mathrm{V}(G)^2$ denotes the endpoints of the edges. Since G is undirected, the ordering of $\Psi(e)$ is insignificant for G but rather used to distinguish different endpoints of e for o.

The b-function describes the **distance bound length** encoded by the edges $\mathrm{E}(G)$. We use $\square$ as an artificial rectangle representing the chip area. For any edge $e \in \mathrm{E}(G)$ and $i \in [\![2]\!]$, $\mathrm{o}_e(i)$ specifies the **offset** of the i'th endpoint of e w.r.t. Ψ (i.e. either a rectangle $v \in R$ or the artificial $\square$).

Please note that Definition 3.4 requires no properties of G, b or o. We stress that G may in particular contain loops, parallel edges as well as negative cycles with respect to b. This is why we defined G_b in the rarely used notion of graphs. This generality is necessary in order to encode arbitrary distance requirements as considered in Proposition 3.3.

A distance bound itself might seem like an odd concept. But before we can demonstrate its intention and relation to Proposition 3.3, we need to extend the notion of slack to distance bounds:

Definition 3.5. Consider a finite set of rectangles R with distance bounds G_b and arbitrary placement $l\colon R \to \mathbb{R}^2$. We define the **distance bound slack** $\mathbf{slack}(l, G_\mathrm{b})$ as

$$\mathrm{slack}(l, G_\mathrm{b}) := \min\Big\{ \mathrm{b}(e) - \big\|l(e,0) - l(e,1)\big\|_1 : e \in \mathrm{E}(G) \Big\}$$

where $l(e,i) := l\big(\Psi(e)_i\big) + \mathrm{o}_e(i)$ is a shorthand for the location of the i'th endpoint of e and $l(\square) := (0,0) \in \mathbb{R}^2$. A bound $\{v, w\} = e \in \mathrm{E}(G)$ is **critical** if e defines the slack of l, i.e. $\mathrm{slack}(l, G_\mathrm{b}) = \mathrm{b}(e) - \big\|l(e,0) - l(e,1)\big\|_1$.

This definition of (distance bound) slack is very analogous to the definition of $\mathrm{slack}(l)$ with respect to the underlying timing graph: If l meets all distance bounds, $\mathrm{slack}(l, G_\mathrm{b}) \geq 0$ defines the maximum threshold by which b could be tightened uniformly while preserving $\big\|l(v_e) - l(w_e)\big\|_1 \leq \mathrm{b}(e)$ for all $\{v, w\} = e \in \mathrm{E}(G)$. If this is not the case, $-\mathrm{slack}(l, G_\mathrm{b}) > 0$ defines the minimum relaxation necessary for b s.t. that this inequality holds for l.

Clearly the definition of slack does not depend on the ordering of endpoints for $e \in \mathrm{E}(G)$ w.r.t. Ψ. To emphasize this we usually write $\big\|l(e)\big\|_1$ for $\big\|l(e,0) - l(e,1)\big\|_1$.

Up to this point distance bounds have not been related to timing in Macro Placement. A priori it is not clear how to infer geometric distance bounds based on Proposition 3.3. This is due to the fact that both rat- and at-values of macro pins depend on the placement of successors and predecessors in the timing graph and thus are not known in advance. Without reliable timing assertions, it is not apparent how to infer distance bounds.

In the following, we want to use practical properties of macros to overcome this problem: Macros mostly define start- and endpoints in the timing graph (cf. Section 2.2.2, page 8 ff.). There are cases of macros requiring signal propagation through them in a

single clock cycle, but these cases are extremely rare. We will later revisit handling such special macros. For the reference, we state this assumption explicitly:

Assumption 3.6. We assume all pins of macros are start- or endpoints in the timing graph.

Assumption 3.6 equivalently assumes fixed at- or rat-values for all macro pins. Note that both thereby are in particular independent from the placement.

Based on Assumption 3.6 we can consider timing in Macro Placement using the following distance bounds for a superset of the macros:

Definition 3.7. Consider an instance I of Macro Placement. Let $S \subseteq P$ denote timing startpoints (i.e. pins with fixed at); analogously define $T \subseteq P$ as the timing endpoints (i.e. with fixed rat). Let R be the rectangles for all shapes of cells $\gamma(S \cup T) \cap C$. We thereby define the **cannonical distance bounds G_b** for I on $\mathrm{V}(G) := R \dot\cup \{\square\}$ as $\mathrm{E}(G) := \{\,(s,t) : s \in S,\, t \in T\,\}$, $\Psi\big((s,t)\big) := \big(\gamma(s), \gamma(t)\big)$,

$$\mathrm{b}\big((s,t)\big) := \frac{\mathrm{rat}(t) - \mathrm{at}(s) - \mathrm{cd}(s,t)}{\alpha}$$

and $\mathrm{o}_e(i)$ to be the offset of e_i on $\gamma(e_i) = \Psi(e)_i$ for $e \in \mathrm{E}(G)$.

Consider a macro $m \in M$ with any pin, i.e. $\gamma^{-1}(m) \neq \varnothing$. By Assumption 3.6 on the set of macros M, we have $m \in \gamma(S \cup T)$. Consequently the canonical distance bounds G_b of Definition 3.7 contains a rectangle $r_m \in \mathrm{V}(G_\mathrm{b})$ for each such macro m. Moreover edges $\delta_{G_\mathrm{b}}(r_m)$ represent distance requirements for all timing startpoints that m can be reached from and endpoints reachable from m.

The series of previous definitions is motivated by the following correspondence of the distance bound slack:

Proposition 3.8. *Consider an instance of* Macro Placement *with corresponding canonical distance bounds G_b. For every placement l of the netlist $\alpha\,\mathrm{slack}(l, G_\mathrm{b}) \geq \mathrm{slack}(l)$.*

Proof. If $\mathrm{E}(G_\mathrm{b}) = \varnothing$ there is nothing to show.

Otherwise by Definition 3.7 of canonical bounds G_b, we have $l(p) = l(e,i)$ for any $p \in S \dot\cup T$, $i \in [\![2]\!]$ and $e \in \mathrm{E}(G)$ with $e_i = p$. Consider any placement l with arbitrary critical bound $e = (s,t) \in \mathrm{E}(G)$. Since $s \in S$ and $t \in T$, we can apply Proposition 3.3 in order to conclude

$$\begin{aligned}\mathrm{slack}(l, G_\mathrm{b}) &= \mathrm{b}(e) - \big\|l(e,0) - l(e,1)\big\|_1 \\ &= \frac{\mathrm{rat}(t) - \mathrm{at}(s) - \mathrm{cd}(s,t)}{\alpha} - \big\|l(t) - l(s)\big\|_1 \overset{\text{Prop. 3.3}}{\geq} \frac{\mathrm{slack}(l)}{\alpha}.\end{aligned}$$

■

Please note that Proposition 3.8 holds not only, but also in particular for feasible placements. Moreover it is applicable to any placement obtained from a rectangle placement extended arbitrarily to non-macro cells. Further note that this proposition is independent from Assumption 3.6. In case this assumption is not met, $\mathrm{slack}(l, G_\mathrm{b})$ only implies a weaker upper bound for $\mathrm{slack}(l)$.

We already emphasized that Definition 3.4 imposes no requirements on b. In particular b may be negative and not necessarily allows $\mathrm{slack}(\,\cdot\,, G_\mathrm{b}) \geq 0$. In fact it is very

common that distance bounds in practice are even negative. This stems from the fact that some timing assertions are overly ambitious and hence lead to those bounds. We defer normalization of those bounds to Section 3.3.

Already at this point we notice that the contribution of loops $e \in \mathrm{E}(G)$ to $\mathrm{slack}(l, G_\mathrm{b})$ doesn't depend on l. Such edges merely impose a priori upper bounds on the best possible slack w.r.t. G_b.

Lemma 3.9. *Consider an instance of* Rectangle Packing *with distance bounds* G_b. *Let* $L \subseteq \mathrm{E}(G)$ *be the set of loops*

$$L := \{\, e \in \mathrm{E}(G) : \Psi(e) = (v, v) \text{ for some } v \in \mathrm{V}(G) \,\},$$

$G' := (\mathrm{V}(G), \mathrm{E}(G) \setminus L)$ *and* $x_L := \min_{e \in L} \{\, \mathrm{b}(e) - \|\mathrm{o}_e(0) - \mathrm{o}_e(1)\|_1 \,\}$. *Here* $\mathrm{slack}(l, G_\mathrm{b}) = \min\{\, x_L, \mathrm{slack}(l, G'_\mathrm{b}) \,\}$ *for any placement* l.

Proof. By definition of $\mathrm{slack}(l, G_\mathrm{b})$

$$\begin{aligned}\mathrm{slack}(l, G_\mathrm{b}) &= \min\left\{ \begin{array}{l} \min\{\, b(e) - \|l(e,0) - l(e,1)\|_1 : e \in L \,\}, \\ \min\{\, b(e) - \|l(e,0) - l(e,1)\|_1 : e \in \mathrm{E}(G) \,\} \end{array} \right\} \\ &= \min\{\, x_L, \mathrm{slack}(l, G'_\mathrm{b}) \,\}.\end{aligned}$$

■

By Lemma 3.9 it is no loss of generality to consider only distance bounds G_b without loops from now on. Disregarding loops is helpful in order to simplify notation:

Firstly we commonly identify an edge $e \in \mathrm{E}(G)$ with the set $\{v, w\}$ where $\Psi(e) = (v, w)$. But inevitably the $\{v, w\}$ shorthand is ambiguous for parallel edges. We will make sure that the concrete edge e with $e = \{v, w\}$ is always apparent from the context. This is why we drop Ψ entirely from here on.

Secondly since each $\{v, w\} = e \in \mathrm{E}(G)$ connects distinct vertices now, we will commonly denote those by v_e and w_e respectively. Formally this defines a shorthand for a tuple (e, i) where $e \in \mathrm{E}(G)$, $i \in [\![2]\!]$. We extend o-functions to this notation, i.e. define $\mathrm{o}(v_e) := \mathrm{o}_e(v)$. Consequently we can also define $l(v_e) := l(v) + \mathrm{o}(v_e)$ for a placement l. We summarize the previously introduced notation with the following central problem:

Timing-Driven Rectangle Packing Problem

Instance: An instance $I = (R, \varphi, \psi, A)$ of the Rectangle Packing Problem (cf. page 13) together with distance bounds G_b without loops.

Task: Find a solution l of I with maximum bound slack $\mathrm{slack}(l, G_\mathrm{b})$ or decide that no feasible solution exists at all.

Instances of the Timing-Driven Rectangle Packing Problem with simple packing constraints will be discussed later on. In order to refer to those concisely, we make the following definition:

Definition 3.10. Let $I = (R, \varphi, \psi, A, G_\mathrm{b})$ be an instance of the Timing-Driven Rectangle Packing Problem. I is said to be an **elementary instance** if I has no feasible area constraints and no disjointness constraints, i.e. $\varphi :\equiv \varnothing$.

3.2 Computing Distance-Bounds

Definition 3.1 of $\mathrm{cd}(s,t)$ contains a choice between Equations (3.1) and (3.2) as underlying inequality. We start this section by justifying the deliberate choice for the weaker Equation (3.2): Unless P = NP, the stronger version can not be computed in polynomial time.

In order to elaborate this, we consider the following problem:

> **Normed Shortest Path Problem**
>
> *Instance:* $d \in \mathbb{N}$, a norm $\|\cdot\|\colon \mathbb{R}^d \to \mathbb{R}$, a directed graph G with weights $w\colon \mathrm{E}(G) \to \mathbb{R}^d$ and two distinct nodes $s,t \in \mathrm{V}(G)$.
>
> *Task:* Find an s-t-path P with minimum $\|w(P)\| := \|\sum_{e\in\mathrm{E}(P)} w(e)\|$ or decide that there is no s-t-path at all.

Theorem 3.11. *The* Normed Shortest Path Problem *is NP-hard even if G is acyclic and $d = 1$.*

Proof. Transformation from Partition, which is well known to be NP-complete (Karp 1972):

Consider an instance of Partition, i.e. numbers $n, c_0, \dots, c_{n-1} \in \mathbb{N}$. We define a graph $G := (V,E)$ with $V := \{v_0, \dots, v_n\}$ and $E := \{e_i, f_i : i \in [\![n]\!]\}$ where $f_i := e_i := (v_i, v_{i+1})$. Let $w\colon \mathrm{E}(G) \to \mathbb{R}$ be defined as $w(e_i) := +c_i$ and $w(f_i) := -c_i$. We further select $s := v_0$, $t := v_n$. The graph G is visualized in Figure 3.2.

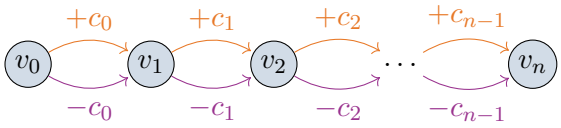


Figure 3.2: Constructed graph G for reducing Partition to Normed Shortest Path.

By construction of G, any s-t-path contains for each i either e_i or f_i. As $\|\cdot\|$ is a norm, $\|x\| = 0$ if and only if $x = 0$. Thus G contains an s-t-path P with $\|w(P)\| = 0$ if and only if $\sum_{i:e_i\in\mathrm{E}(P)} c_i = \sum_{i:f_i\in\mathrm{E}(P)} c_i$, i.e. $\{c_i : e_i \in \mathrm{E}(P)\}$ is a feasible partition. ∎

For general dimensions d the Normed Shortest Path Problem is even strongly NP-hard:

Theorem 3.12. *The* Normed Shortest Path Problem *is strongly NP-hard even if G is acyclic and $\|\cdot\| \equiv \|\cdot\|_\infty$.*

Proof. Transformation from p-Partition, which is strongly NP-hard due to Garey and Johnson (1975, 1978):

Consider an instance of p-Partition consisting of $p, n, c_0, \dots, c_{n-1} \in \mathbb{N}$. Define $C := \sum_{i\in[\![n]\!]} c_i$. Then p-Partition is the problem to decide whether $g\colon [\![n]\!] \to [\![p]\!]$ with $\sum_{j\in g^{-1}(k)} c_j = C/p$ for all $k \in [\![p]\!]$ exists.

We consider a graph $G := (V,E)$ similar to the one constructed in the proof of Theorem 3.11: $V := \{v_0, \dots, v_n\}$ and $E := \{e_i^k : k \in [\![p]\!], i \in [\![n]\!]\}$ where $e_i^k := (v_i, v_{i+1})$. We

further define $d := p$, $s := v_0$, $t := v_n$ and weights $w\colon E \to \mathbb{R}^p$ as $w(e_i^k) := \chi_k c_i$ where χ_k is the characteristic unit vector in dimension k.

By construction any s-t-path P in G uses exactly one edge of $\{ e_i^k : k \in [\![p]\!] \}$ for each $i \in [\![n]\!]$, thus $\|w(P)\|_\infty \geq C/p$ in general as $\|w(P)\|_1 = c$.

We claim that there is an s-t-path P in G with $\|w(P)\|_\infty = C/p$ if and only if the p-Partition instance is a yes-instance: Given a certificate g as above for the instance of p-Partition, the path P with $\mathrm{E}(P) := \{ e_i^{g(i)} : 0 \leq i < n \}$ satisfies $\|w(P)\|_\infty = C/p$. If on the other hand any s-t-path P in G satisfies $\|w(P)\|_\infty = C/p$, the partition g defined as $g(i) := k$ where $e_i^k \in \mathrm{E}(P)$ certifies the original instance of p-Partition as yes-instance. ■

We have seen that the Normed Shortest Path Problem in general is strongly NP-hard and remains NP-hard for $d = 1$. This is important, as it implies the following corollary:

Corollary 3.13. *The following problem is NP-hard: Given an instance of* Macro Placement *with two pins s and t, compute*

$$\max \left\{ \sum_{e \in \mathrm{E_c}(P)} \mathrm{d_c}(e) - \alpha \left\| \sum_{e \in \mathrm{E_c}(P)} \delta_e \right\|_1 : P \text{ is an } s\text{-}t\text{-path in the timing graph} \right\}$$

where $\delta_e \in \mathbb{R}^2$ is the offset of the pins of a circuit internal edge e.

Proof. Implied by Theorem 3.11 as this problem contains the Normed Shortest Path Problem for acyclic G and $d = 1$ in the special case of $\mathrm{d_c} \equiv 0$. ■

Due to Corollary 3.13 it is impossible to compute a variant of $\mathrm{cd}(s,t)$ based on Equation (3.1) in polynomial time, unless P = NP. This is why Definition 3.1 makes use of the inferior Equation (3.2).

But this definition of $\mathrm{cd}(s,t)$ still requires to consider all s-t-paths. Although there may be exponentially many such paths, we can compute $\mathrm{cd}(s,t)$ efficiently.

Lemma 3.14. *Consider an instance of* Macro Placement. *We can compute $\mathrm{cd}(s,t)$ for $s \in S \subseteq V$, $t \in T \subseteq V$ in $\mathcal{O}(k(n+m))$ time and $\mathcal{O}(kn)$ space where n and m refer to the timing graph and $k := \min\{ |S|, |T| \}$.*

Proof. We compute $\mathrm{cd}(s,v)$ for all $s \in S$ and $v \in V$; in particular for $v \in T$. For any $s \in S$, $v \in V$ we have the recursion formula

$$\mathrm{cd}(s,v) = \max \Big\{ \mathrm{cd}(s,u) + \mathrm{cd}(u,v) : (u,v) \in \delta^-(v) \Big\},$$

where $-\infty + x := -\infty$ for $x \in \mathbb{R}$. Please note that $\mathrm{cd}(u,v)$ is either 0 for wiring edges (u,v) or otherwise the (constant) circuit-internal delay contribution $\mathrm{d_c}(u,v) - \alpha \|u,v\|_1$ from u to v.

We thus can compute $\mathrm{cd}(s,v_i)$ for each $s \in S$ by the recursion formula using a topological ordering $\{ v_0, \ldots, v_{n-1} \} = V$ of the nodes. This takes $\mathcal{O}(n+m)$ time and $\mathcal{O}(n)$ space for each $s \in S$. Analogously we can work on the reversed timing graph in case $|T| < |S|$ which concludes the proof. ■

The complexity of Lemma 3.14 is reasonable in cases where we can reach most of the timing graph from $s \in S$. This is not the case in practice however. There we are confronted with timing graphs where we can only reach very small fractions of the nodes from a fixed $s \in S$. Consequently accounting for $\mathcal{O}(n)$ space and $\mathcal{O}(n+m)$ time for such a node s is a huge overhead.

But we can improve Lemma 3.14 for those kinds of instances.

Proposition 3.15. *Consider an instance of* Macro Placement. *We can compute* $\mathrm{cd}(s,t)$ *for* $s \in S \subseteq V$ *and* $t \in T \subseteq V$ *in* $\mathcal{O}(\sum_{s \in S} n_s + m_s)$ *time and* $\mathcal{O}(\max\{ n_s : s \in S \})$ *space where we define* $n_s := |\{ v \in V : v \text{ reachable from } s \}|$ *as well as* $m_s := |\{(v,w) \in E : v \text{ reachable from } s\}|$ *for* $s \in S$.

Proof. We proceed for each $s \in S$ individually: We compute the subgraph $H_s \subseteq G$ reachable from s in $\mathcal{O}(n_s + m_s)$ time. In the same running time, we can compute $\mathrm{cd}(s,t)$ for all $t \in T$ by Lemma 3.14 applied to H_s. This implies the stated complexities for both running time and space. ■

The running time of Proposition 3.15 is clearly superior to the one of Lemma 3.14 since $n_s + m_s \leq n + m$. For instances in practice this difference can be drastic. In addition the space complexity is output sensitive.

Please note we analogously can propagate $\mathrm{cd}(\,\cdot\,, t)$ backwards in topological order of the timing graph. This yields a similar result to Proposition 3.15 with analogous definitions of $n_t, m_t \in \mathbb{N}$ for $t \in T$.

3.3 Basic Properties

Up to now we have defined the Timing-Driven Rectangle Packing Problem and recognized its practical relevance. In the following we want to gain a deeper understanding of this problem. We start with an obvious negative observation:

Theorem 3.16. *The* Timing-Driven Rectangle Packing Problem *is strongly NP-hard.*

Proof. It contains the Rectangle Packing Problem as the special case where there are no bounds at all. This problem in turn contains p-Partition, which is known to be strongly NP-hard due to Garey and Johnson (1975). ■

Although the Timing-Driven Rectangle Packing Problem is computationally difficult in general, it has interesting properties:

Lemma 3.17. *Consider an instance* (R, G_{b}) *of* Timing-Driven Rectangle Packing *and* $\delta \in \mathbb{R}$. *There is a solution for* (R, G_{b}) *with slack* $r \in \mathbb{R}$ *if and only if there is a solution for* $(R, G_{\mathrm{b}+\delta})$ *with slack* $r + \delta$.

Proof. For any placement l we observe for each $\{v,w\} = e \in \mathrm{E}(G)$ individually

$$\mathrm{b}(e) - \|l(w_e) - l(v_e)\|_1 = \Big[(\mathrm{b}+\delta) - \|l(w_e) - l(v_e)\|_1\Big] - \delta.$$

Consequently $\mathrm{slack}(l, G_{\mathrm{b}}) + \delta = \mathrm{slack}(l, G_{\mathrm{b}+\delta})$. ■

Lemma 3.17 highlights a special aspect of the Timing-Driven Rectangle Packing Problem: The problem is invariant under changing b by constants. Though this problem was originally motivated as bounding lengths of paths, this property doesn't apply to path distances themselves. This is no contradiction but rather due to the fact that the type of relaxation used in Lemma 3.17 represents a uniform increase of rats.

Lemma 3.17 allows us to henceforth assume the following non-negative normalization of distance bounds:

Corollary 3.18. *Let* (R, G_b) *be an instance of* Timing-Driven Rectangle Packing. *There is an equivalent instance* $(R, G_{\mathrm{b}'})$ *for which* $\min \mathrm{b}' = 0$.

Proof. Apply Lemma 3.17 with $\delta := -\min\{\, \mathrm{b}(e) : e \in \mathrm{E}(G) \,\}$. ■

Since the distance bound $\mathrm{slack}(l, G_\mathrm{b})$ is defined on individual edges, it is monotonic w.r.t. subgraphs:

Lemma 3.19. *Consider an instance of* Rectangle Packing *with different distance bounds* (G, b, o) *and* (G', b, o) *where* $G' \subseteq G$ *is a subgraph. For any placement* l, $\mathrm{slack}(l, G_\mathrm{b}) \leq \mathrm{slack}(l, G'_\mathrm{b})$.

Proof. Let $H_e := ((\mathrm{V}(G), \{\, e \,\}), b, o)$ denote the restriction of G_b to a single edge $e \in \mathrm{E}(G)$. As $\mathrm{E}(G') \subseteq \mathrm{E}(G)$

$$\begin{aligned}\mathrm{slack}(l, G_\mathrm{b}) &= \min\{\, \mathrm{slack}(l, H_e) : e \in \mathrm{E}(G) \,\} \\ &\leq \min\left\{\, \mathrm{slack}(l, H_e) : e \in \mathrm{E}\left(G'\right) \right\} = \mathrm{slack}(l, G'_\mathrm{b}).\end{aligned}$$ ■

This implies the following corollary on the respective optimum solutions:

Corollary 3.20. *Consider an instance of* Rectangle Packing *with distance bounds* (G, b, o) *and* (G', b, o) *where* $G' \subseteq G$ *is a subgraph. For optimum solutions* l *for* G_b *and* l' *for* G'_b, $\mathrm{slack}(l, G_\mathrm{b}) \leq \mathrm{slack}(l', G'_\mathrm{b})$.

Proof. By Lemma 3.19 and optimality of l', $\mathrm{slack}(l, G_\mathrm{b}) \leq \mathrm{slack}(l, G'_\mathrm{b}) \leq \mathrm{slack}(l', G'_\mathrm{b})$. ■

We now want to analyze the structure of optimum solutions of elementary instances of Timing-Driven Rectangle Packing. Therefore we use the following geometric definitions:

Definition 3.21. Consider an instance of Timing-Driven Rectangle Packing with placement l. We define the **angle $\measuredangle_l(r_e)$** of $e = \{\, r, r' \,\} \in \mathrm{E}(G)$ w.r.t. l for $0 \neq (x, y) := l(r'_e) - l(r_e)$ as

$$\measuredangle_l(r_e) := \begin{cases} \frac{1}{2}\pi - \arctan\left(xy^{-1}\right) & \text{for } y > 0 \\ \frac{3}{2}\pi - \arctan\left(xy^{-1}\right) & \text{for } y < 0 \\ \frac{1}{2}\pi - \frac{1}{2}\pi \operatorname{sgn}(x) & \text{for } y = 0. \end{cases}$$

Informally the angle $\measuredangle_l(r_e)$ is the angle of a horizontal line through $l(r_e)$ and the line segment connecting endpoints of e placed according to l (cf. Figure 3.3). We can easily verify $\measuredangle_l(r_e) - \measuredangle_l(r'_e) = \pi \mod 2\pi$ by a case distinction whether y as in Definition 3.21 is non-zero.

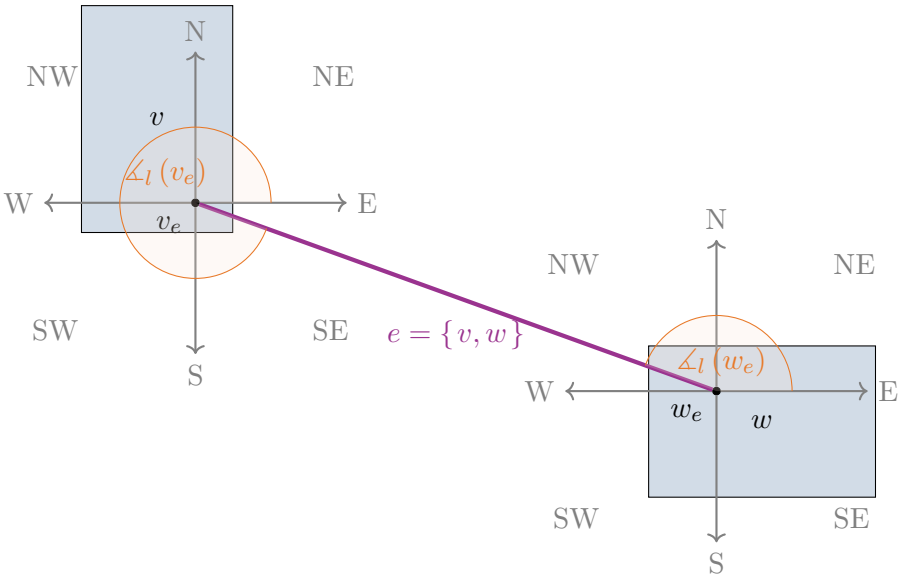


Figure 3.3: Edge $e = \{v, w\} \in \mathrm{E}(G)$ drawn in purple according to placement l. The corresponding angles at v_e and w_e are shown in orange, while respective directions are specified in gray.

Definition 3.22. Consider an instance of Timing-Driven Rectangle Packing with placement l. For $e = \{r, r'\} \in \mathrm{E}(G)$ with $l(e) > 0$, we call a $k \in [\![4]\!]$ for which $\measuredangle(r_e) \in [k\frac{\pi}{2}, (k+1)\frac{\pi}{2}] \mod 2\pi$ a **direction** of r_e. In case $l(e) = 0$, r_e is considered to have all four directions.

In analogy to the compass directions we will identify $k = 0$ with **NE**, 1 and **NW**, 2 and **SW** as well as 3 with **SE** (cf. Figure 3.3). Since Definition 3.22 considers closed intervals of angles, r_e can have up to two directions if $\|l(e)\|_1 > 0$. Edges e with $\|l(e)\|_1 = 0$ even have all four directions. We use the notion of directions in order to characterize certain paths:

Definition 3.23. Consider an instance of Timing-Driven Rectangle Packing with placement l. A path $(v^0, e^0, \ldots, e^{k-1}, v_k)$ in G_b is called ***l*-monotone** if all $e^i_{v^i}$ have a common direction $k \in [\![4]\!]$ w.r.t. l.

Monotone paths of critical edges will become crucial for finding optimum solutions. We start by characterizing whether such paths exist:

Lemma 3.24. *Consider an elementary instance (R, G_b) of* Timing-Driven Rectangle Packing. *Any optimum solution l for which the number of l-critical $e \in \mathrm{E}(G)$ is minimum, has the following property: For each l-critical $e \in \mathrm{E}(G)$ with $\square \in e$, $\|l(e)\|_1 > 0$ and direction $d \in [\![4]\!]$ there is a monotone cycle with direction d of critical edges w.r.t. l containing e.*

Proof. Since G contains no loops, let $\{u, \square\} = e \in \mathrm{E}(G)$ be critical with $u \in R$ and $\|l(e)\|_1 > 0$. Due to symmetry we may assume e has direction NE, possibly after mirroring the instance. Denote by $U \subseteq \mathrm{V}(G)$ the nodes of G that can be reached from u by a NE-monotone path of critical edges w.r.t. l. We will prove a contradiction if e can not be extended to the desired cycle i.e. if $\square \notin U$:

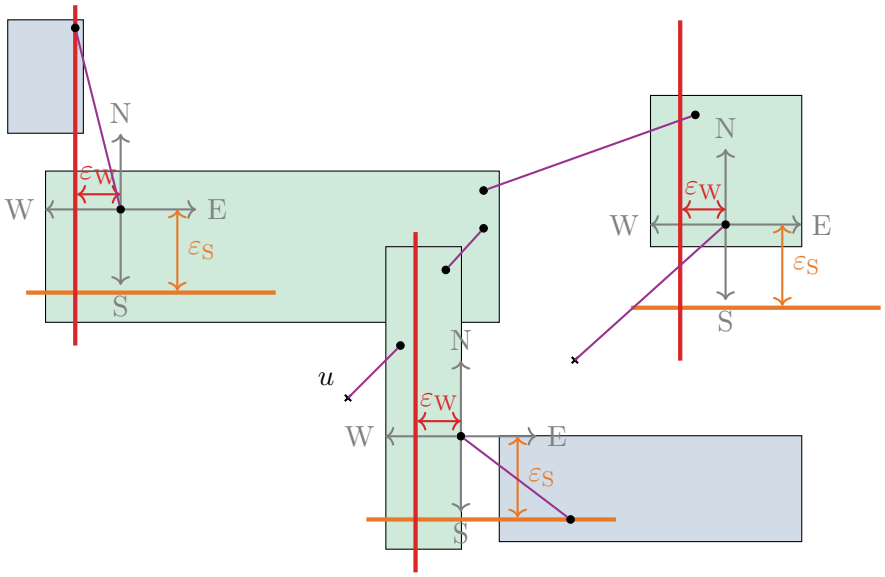


Figure 3.4: Construction for proving Lemma 3.24: Three green rectangles in U, two blue ones in $R \setminus U$. All six edges in purple are critical. The definitions of ε_S and ε_W based on F are highlighted in orange and red.

Let $\varepsilon_l := \min\{\operatorname{slack}(l,f) - \operatorname{slack}(l,G_b) : f \in E(G)\ l\text{-uncritical}\}$. Denote by F all l-critical edges $f \in \delta(U)$ and further

$$\varepsilon_W := \min\left\{\left|x\left(l\left(v_f\right) - l\left(w_f\right)\right)\right| : f=\{v,w\} \in F,\ v \in U,\ \measuredangle_l\left(v_f\right) \in \left(\frac{\pi}{2}, \frac{3\pi}{2}\right)\right\}.$$

In other words, ε_W denotes the margin w.r.t. x-coordinates of edges leaving U to W but not straight S or N. Analogously define ε_S w.r.t. y- instead of x-coordinates and $\measuredangle_l\left(u_f\right) \in (\pi, 2\pi)$, i.e. considering edges leaving U to S but not straight W or E. By choice $\varepsilon_l, \varepsilon_W, \varepsilon_S > 0$ and consequently $\varepsilon_U := \min\{\varepsilon_S, \varepsilon_W\} > 0$. The constellation is depicted in Figure 3.4.

Consider another solution l' obtained from l by translating $U \subseteq R$ by $-(\varepsilon,\varepsilon) \in \mathbb{R}^2$ for $0 < \varepsilon := \frac{1}{2}\min\{\varepsilon_l, \varepsilon_U\}$. Since l was optimum, $\operatorname{slack}(l,G_b) \geq \operatorname{slack}(l',G_b)$. By construction $\operatorname{slack}(l,f) = \operatorname{slack}(l',f)$ for $f \in E(G) \setminus \delta(U)$. Moreover $\operatorname{slack}(l',f) > \operatorname{slack}(l,G_b)$ for l-uncritical edges $f \in \delta(U)$ due to $\varepsilon < \varepsilon_l$ and thus f also l'-uncritical.

By choice of U no $f \in F$ leaves U to NE w.r.t. l. Due to $\varepsilon < \varepsilon_U$ no $f \in F$ can leave U to NE w.r.t. l' as well, particularly not straight to N or E. For that reason and since $\varepsilon < \varepsilon_U$, $f \in F$ leaves U to NW or SE w.r.t. l if and only if this is the case w.r.t. l'. Thus by construction of l', $\|l(f)\|_1 = \|l'(f)\|_1$ for $f \in F$ leaving U to NW or SE. Consequently we obtain $\operatorname{slack}(l,f) = \operatorname{slack}(l',f)$. Finally all $f \in F$ leaving U to SW excluding E and N are not l'-critical since $\|l(f)\|_1 > \|l'(f)\|_1$ by choice of ε_U. This contradicts the assumptions on l since this particularly applies to e. ■

Lemma 3.24 can be used for proving a central proposition of this thesis. This observation for cycles through $\square$ can be used in order to characterize monotone cycles in general:

Proposition 3.25. *Consider an elementary instance* (R, G_b) *of* Timing-Driven Rectangle Packing. *Any optimum solution* l *for which the number of* l*-critical* $e \in E(G)$ *is minimum, has the following property: For each* l*-critical* $e \in E(G)$ *and* $\|l(e)\|_1 > 0$*, there is a monotone cycle* C *of critical edges w.r.t.* l *in* G *containing* e.

Proof. Let $e \in E(G)$ be l-critical with $\|l(e)\|_1 > 0$. If $\square \in e$, the stated property is implied by Lemma 3.24. Thus we assume $\square \notin e = \{v, w\}$.

Let G'_b be the instance where v is considered to be artificially fixed at $l(v)$. In this instance v corresponds to $\square$ and e to $e' := \{\square, w\}$. Due to $e' \in E(G')$, $\text{slack}(l, G_b) = \text{slack}(l, G'_b)$. If l was suboptimal w.r.t. G'_b, we could extend $l(v)$ to an optimum solution of G_b with no further critical edges. But this contradicts the assumption on l since such an extension would avoid the l-critical edge e in particular. Thus l is an optimum solution of G'_b.

Moreover, since l has minimum number of critical edges for G_b, the same must hold for G'_b. Thus we can apply Lemma 3.24 for e' which is why there is a monotone cycle C' of critical edges w.r.t. l in G'_b starting with e'. Note that since v is fixed in G'_b, C' does not necessarily correspond to a cycle in G_b.

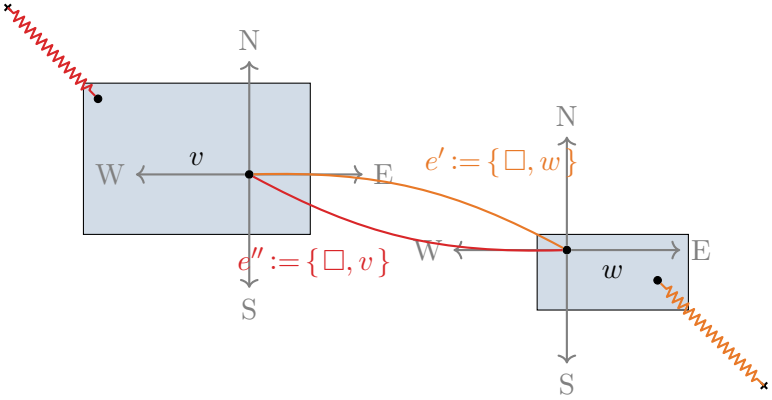


Figure 3.5: Construction of cycle C in the proof of Proposition 3.25. Two l-monotone cycles C' and C'' presented in orange and red both containing $\square$ with opposite directions.

If the unique edge in $\delta_{C'}(\square) \setminus \{e'\}$ represents an edge in $\delta_G(v)$, C' is also a cycle in G_b. Otherwise, we similarly consider the instance G''_b where w ($e = \{v, w\}$) is fixed at $l(w)$. By the same arguments, there is another l-monotone cycle of l-critical edges $C'' \subseteq G''$ containing $e'' := \{\square, v\}$ with direction opposite of C'. If C'' corresponds to a cycle in G, we are done. Otherwise C'/C'' represents a v/w-$\square$-path in G with opposite directions. This situation is visualized in Figure 3.5. Thus $C := C' \cup C'' \cup \{e\} \setminus \{e', e''\}$ is a cycle with the desired properties. ■

3.4 Strongly Polynomial Algorithm

Now we develop an algorithm for finding the optimum slack s^* for elementary instances of Timing-Driven Rectangle Packing. Due to Proposition 3.25, there are optimum solutions with critical edge of length 0 or monotone cycle of critical edges. The former case, s^* is easily identified.

In order to handle the latter case, we will compute s^* as minimum with the following guarantee: All monotone cycles have non-negative slack w.r.t. bounds relaxed by s^*. This

is equivalent by Lemma 3.17. We will explicitly exploit Proposition 3.25 by considering only monotone cycles when computing s^* with this approach.

Therefore we need to consider the following classical optimization problem:

Minimum Ratio Cycle Problem

Instance: A directed graph G with $w_i\colon \mathrm{E}(G) \to \mathbb{R}$ for $i \in [\![2]\!]$ where $w_1 \geq 0$ and w_0 is conservative on $(\mathrm{V}(G), \{\, e \in \mathrm{E}(G) : w_1(e) = 0 \,\})$.
Task: Find maximum $\lambda \in \mathbb{R} \cup \{\infty\}$ s.t. $f_\lambda := w_0 - \lambda w_1$ is conservative.

The problem is well-defined, i.e. $\lambda \in \mathbb{R}$ exists s.t. f_λ is conservative e.g.

$$\lambda < -\frac{\sum_{e \in \mathrm{E}(G)} |w_0(e)|}{\min\{\, w_1(e) : e \in \mathrm{E}(G),\, w_1(e) > 0 \,\}}$$

(for $\frac{\cdot}{\infty} := 0$): For any cycle $C \subseteq G$, either $f_\lambda(C) \geq 0$ by choice of λ if there is an $e \in \mathrm{E}(C)$ with $w_1(e) > 0$, or by assumption on w_0 otherwise.

The name of this problem stems from the fact that for instances with optimum $\lambda^* < \infty$ we have $\lambda^* = \min\{\, w_0(\mathrm{E}(C))/w_1(\mathrm{E}(C)) : C \subseteq G \text{ cycle} \,\}$ (when abbreviating $\frac{\cdot}{0} := \infty$). The Minimum Ratio Cycle Problem generalizes the Minimum Mean Cycle Problem for the special case of $w_1 \equiv 1$.

For an instance (G, f_λ) of Minimum Ratio Cycle we denote by $\mathbf{mr}\,(G, f_\lambda)$ the optimum solution λ of this instance.

Before we mention previously known positive results on the Minimum Ratio Cycle Problem, we will first deal with negative results. The following variants are NP-hard:

Lemma 3.26. *Consider the variation of* Minimum Ratio Cycle *where w_0 is arbitrary with the following task: determine the maximum λ s.t. $f_\lambda(\mathrm{E}(C)) \geq 0$ for all cycles $C \subseteq G$ with $w_1(\mathrm{E}(C)) \neq 0$. This problem is NP-hard.*

Proof. We reduce from Hamiltonian Path: Given a graph G with distinct $s, t \in \mathrm{V}(G)$, decide whether a Hamiltonian s-t-path exists in G.

For a given instance, we consider the graph $H := (\mathrm{V}(G), \mathrm{E}(G) \cup \{\, e^* \,\})$ with $e^* := (t, s)$. Furthermore we define $w_0 := \equiv -1$ and $w_1 := \chi_{\{\, e^* \,\}}$. Let λ^* be the optimum λ on this instance of the Minimum Ratio Cycle Problem variant. G has a Hamiltonian s-t-path if and only if $\lambda^* = -(n-1)$. ∎

Although the λ computed in the variant of Lemma 3.26 seems to disregard cycles $C \subset G$ with $w_0(\mathrm{E}(C)) < 0$ and $w_1(\mathrm{E}(C)) = 0$ completely, the sole presence of such cycles makes the problem difficult. The same arguments imply another negative result:

Lemma 3.27. *Consider the variation of* Minimum Ratio Cycle *where w_0 is arbitrary with the following task: determine for a fixed $e \in \mathrm{E}(G)$ with $w_1(e) \neq 0$ the maximum λ s.t. $f_\lambda(\mathrm{E}(C)) \geq 0$ for all cycles $C \subseteq G$ with $e \in \mathrm{E}(C)$. This problem is NP-hard.*

Proof. Implied by the same reduction as used for the proof of Lemma 3.26. ∎

Now we can turn our attention to the positive results: The Minimum Mean Cycle Problem can be solved in $\mathcal{O}(nm)$ by a dynamic programming algorithm due to Karp (1978).

The general Minimum Ratio Cycle Problem has been considered by Megiddo (1983) who provides two different algorithms with running time $\mathcal{O}(n^3 \log^2 n)$ and $\mathcal{O}(n^3 \log n + mn \log^2 n \log\log n)$ respectively.

For the special case where $w_1(e) \in \{0,1\}$ for all $e \in \mathrm{E}(G)$ faster algorithms are known: Karp and Orlin (1981) generalized the dynamic programming approach for Minimum Mean Cycle to an $\mathcal{O}\big(n^3\big)$ algorithm for Minimum Ratio Cycle and proposed a parametric shortest path algorithm with running time $\mathcal{O}(nm \log n)$. The running time of the latter approach was subsequently improved by Young, Tarjan and Orlin (1991) to $\mathcal{O}\big(mn + n^2 \log n\big)$ using Fibonacci-Heaps (Fredman and Tarjan 1987).

Now we work towards the announced algorithm for finding the optimum slack. This will be done by computing minimum ratio cycles in the following auxiliary graph:

Definition 3.28. Let (R, G_b) be an instance of Timing-Driven Rectangle Packing. The **distance graph $\mathrm{D}(G)$** is the bidirectionally oriented version of G, i.e. the directed graph

$$\mathrm{D}(G) := \Big(\mathrm{V}(G), \{(v,w) : \{v,w\} \in \mathrm{E}(G)\}\Big).$$

For given direction d we further define $f_\lambda^d := w_0^d - \lambda \colon \mathrm{E}(\mathrm{D}(G)) \to \mathbb{R}$ as

$$w_0^d((v,w)) := \mathrm{b}(e) + \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot z\big(\mathrm{o}_e(v) - \mathrm{o}_e(w)\big) \qquad e = \{v,w\} \in \mathrm{E}(G),$$

where $\sigma_z^d \in \{-1,1\}$ is the signum of z-coordinates in direction d for $z \in \{\mathrm{x},\mathrm{y}\}$ i.e.

$$\sigma_z^d := \begin{cases} +1 & \text{if } (z,d) \in \{(\mathrm{x},\mathrm{NE}),(\mathrm{x},\mathrm{SE}),(\mathrm{y},\mathrm{NE}),(\mathrm{y},\mathrm{NW})\}, \\ -1 & \text{else.} \end{cases}$$

Note that we usually omit the extra braces and just write $f_\lambda^d(v,w)$ for $f_\lambda^d((v,w))$. The distance graph is defined for general instances of Timing-Driven Rectangle Packing, but will only be used for elementary ones.

By Definition 3.28, f_λ^d depends on λ for every $e \in \mathrm{E}(\mathrm{D}(G))$. Thus $(\mathrm{D}(G), f_\lambda^d)$ is an instance of the Minimum Mean Cycle Problem.

Before we can investigate the relation of the distance graph $\mathrm{D}(G)$ to Timing-Driven Rectangle Packing, we analyze some basic properties: By definition of Timing-Driven Rectangle Packing, G contains no loops. This is why the same holds for the distance graph $\mathrm{D}(G)$.

Next we notice that despite $\mathrm{D}(G)$ being a directed graph, the certificates for edges in this graph are unordered $e \in \mathrm{E}(G)$. Hence the distance graph $\mathrm{D}(G)$ contains so called reverse edges, i.e. for $e = (v,w) \in \mathrm{E}(\mathrm{D}(G))$ the **reverse edge $\overline{e}$**$:=(w,v) \in \mathrm{E}(\mathrm{D}(G))$. Consequently every s-t-path $P \subseteq \mathrm{D}(G)$ corresponds to a t-s-path $\overline{P}$ in $\mathrm{D}(G)$ consisting of reverse edges of P.

Lemma 3.29. *Let (R, G_b) be an instance of* Timing-Driven Rectangle Packing *and c, d two opposite directions. The weighted graphs $(\mathrm{D}(G), f_\lambda^c)$ and $(\mathrm{D}(G), f_\lambda^d)$ are isomorphic, i.e. $\exists\varphi \colon \mathrm{E}(\mathrm{D}(G)) \to \mathrm{E}(\mathrm{D}(G))$ bijective s.t. for each $v \in \mathrm{V}(\mathrm{D}(G))$ there is $v' \in \mathrm{V}(\mathrm{D}(G))$ with $\varphi(\delta^p(v)) = \delta^q(v')$ for $\{p,q\} = \{+,-\}$ and for any $e \in \mathrm{E}(\mathrm{D}(G))$ we have $f_\lambda^c(e) = f_\lambda^d(\varphi(e))$.*

Proof. Mapping e w.r.t. f^c_λ to $\overline{e}$ according to f^d_λ defines such an isomorphism where we choose $v' := v$. ■

Due to Lemma 3.29, we will only need to consider two of the four versions of the distance graph $\mathrm{D}(G)$.

Next we explore the correspondence of cycles in $\mathrm{D}(G)$ and G_b. In order to analyze cycles $C \subseteq G$ and their corresponding counterparts in $\mathrm{D}(G)$, we introduce the following notation:

Definition 3.30. Let $C = (e_0, \dots, e_{k-1})$ be a cycle in G with $e_i := \{ v^i, v^{i+1} \} \in \mathrm{E}(G)$ for $i \in [\![k]\!]$ (where indices k and 0 are identified). We define the corresponding **distance graph cycle dgc(C)** as $\mathrm{dgc}(C) := ((v_i, v_{i+1}) : i \in [\![k]\!])$ in $\mathrm{D}(G)$.

We further define the cycle $\mathrm{dgc}(e) := ((v_e, w_e), (w_e, v_e))$ in $\mathrm{D}(G)$ for a single edge $e = \{ v, w \} \in \mathrm{E}(G)$.

Please note that Definition 3.30 of $\mathrm{dgc}(e_0, \dots, e_{k-1})$ for a cycle $C = (e_0, \dots, e_{k-1})$ is independent from the starting edge e_0 chosen, but rather assumes a fixed order of traversal for C. If this order is inverted,

$$\mathrm{dgc}(e_{k-1}, \dots, e_0) = \overline{\mathrm{dgc}(e_0, \dots, e_{k-1})}$$

consists of the reverse edges. In that sense, the definition of dgc depends on the chosen order of C. For a general cycle in G_b, we will either explicitly mention the order or write $\mathrm{dgc}(C)$ to emphasize the independence of any chosen order.

By construction of $\mathrm{D}(G)$ in Definition 3.28, any cycle in $\mathrm{D}(G)$ is the distance graph cycle $\mathrm{dgc}(C)$ where $C \subseteq G$ either is a single edge or a cycle. This applies to any minimum ratio cycle C' of $(\mathrm{D}(G), f^d_\lambda)$ in particular.

In the following we investigate how the optimum slack for G_b relates to the minimum ratio for any instance $(\mathrm{D}(G), f^d_\lambda)$ of the Minimum Ratio Cycle Problem.

Proposition 3.31. *For an elementary instance (R, G_b) of* Timing-Driven Rectangle Packing, *we have $s^* \geq \min_d \{ \mathrm{mr}(\mathrm{D}(G), f^d_\lambda) \}$ where s^* is the optimum slack achievable in (R, G_b).*

Proof. If $\mathrm{E}(G) = \varnothing$, there is nothing to show since $s^* = \infty$ by Definition 3.5. Otherwise, we consider an optimum solution l^* for G_b with minimum number of critical edges. By Proposition 3.25 there either is an l^*-critical $e \in \mathrm{E}(G)$ with $\|l^*(e)\|_1 = 0$ or a monotone cycle of critical edges w.r.t. l^*. We note $\|l^*(e)\|_1 = \mathrm{b}(e) - s^*$ for any critical edge e by Definition 3.5.

In the former case if $e = \{ v, w \} \in \mathrm{E}(G)$ critical with $\|l^*(e)\|_1 = 0$ exists, the cycle $C := \mathrm{dgc}(e) \subseteq \mathrm{D}(G)$ satisfies for any direction d

$$\begin{aligned} f^d_{s^*}(C) &= \sum_{(a,b) \in \{ (v,w),(w,v) \}} \left[\mathrm{b}(e) - s^* + \sum_{z \in \{ \mathrm{x}, \mathrm{y} \}} \sigma^d_z \cdot z(\mathrm{o}_e(a) - \mathrm{o}_e(b)) \right] \\ &= 2(\mathrm{b}(e) - s^*) = 2\|l^*(e)\|_1 = 0. \end{aligned}$$

For the other case let D be an l^*-monotone and l^*-critical cycle in G and denote its direction by d. We choose the order $D := (e_0, \dots e_{k-1})$ with $e_i := \{ v^i, v^{i+1} \}$ s.t. e_i has direction d at v_i. Consequently

$$\sum_{z \in \{ \mathrm{x}, \mathrm{y} \}} \sigma^d_z \cdot z\left(l^*\left(v^{i+1}_{e_i} \right) - l^*\left(v^i_{e_i} \right) \right) = \|l^*(e_i)\|_1$$

for all $i \in [\![k]\!]$. As all e_i are also critical, $\|l^*(e_i)\|_1 = \mathrm{b}(e_i) - s^*$. For the cycle $C := \mathrm{dgc}(e_0, \dots, e_{k-1}) \subseteq \mathrm{D}(G)$ we thus compute the telescoping sum

$$\begin{aligned}
0 &= \sum_{i \in [\![k]\!]} \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot z\Big(l^*\Big(v^{i+1}\Big) - l^*\Big(v^i\Big)\Big) \\
&= \sum_{i \in [\![k]\!]} \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot z\Bigg(\Big[l^*\Big(v_{e_i}^{i+1}\Big) - l^*\Big(v_{e_i}^i\Big)\Big] + \Big[\mathrm{o}_{e_i}\Big(v^i\Big) - \mathrm{o}_{e_i}\Big(v^{i+1}\Big)\Big]\Bigg) \\
&= \sum_{i \in [\![k]\!]} \Bigg[\mathrm{b}(e_i) - s^* + \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot z\Big(\mathrm{o}_{e_i}\Big(v^i\Big) - \mathrm{o}_{e_i}\Big(v^{i+1}\Big)\Big)\Bigg] \\
&= \sum_{i \in [\![k]\!]} f_{s^*}^d\Big(v^i, v^{i+1}\Big) = f_{s^*}^d(C).
\end{aligned}$$

In either case we constructed a corresponding cycle $C \subseteq \mathrm{D}(G)$ and direction d with $f_{s^*}^d(C) = 0$. Since $\mathrm{E}(C) \neq \varnothing$, $s^* \geq \min_d\{\,\mathrm{mr}\,(\mathrm{D}(G), f_\lambda^d)\,\}$. ■

In the following we work on the reverse direction i.e. upper bounds on the optimum slack given by minimum ratio cycles in $(\mathrm{D}(G), f_\lambda^d)$. We start by showing equality for certain subgraphs $H \subseteq \mathrm{D}(G)$ which will overall be sufficient to show the desired upper bound.

Lemma 3.32. *Let (R, G_b) be an elementary instance of* Timing-Driven Rectangle Packing *with $e \in \mathrm{E}(G)$. For the optimum slack s^* of $(\mathrm{V}(G), \{e\})_\mathrm{b}$ we have $s^* = \min_d\{\,\mathrm{mr}\,(\mathrm{dgc}(e), f_\lambda^d)\,\}$.*

Proof. Since G contains no loops, e has distinct endpoints. Thus there is a placement l for R where e has l-length 0 i.e. $\|l(e)\|_1 = 0 = \mathrm{b}(e) - s^*$. On the other hand, $f_\lambda^d(\mathrm{dgc}(e)) = 2\big(\mathrm{b}(e) - \lambda\big)$ for any direction d. Consequently $s^* = \lambda$. ■

The same holds for cycles but the proof requires more work.

Lemma 3.33. *Let (R, G_b) be an elementary instance of* Timing-Driven Rectangle Packing *with cycle $C \subseteq G$. For the optimum slack s^* of C_b we have $s^* = \min_d\{\,\mathrm{mr}\,(\mathrm{dgc}(C), f_\lambda^d)\,\}$.*

Before we prove this lemma, note that Lemma 3.33 is independent of the direction of C for $\mathrm{dgc}(C)$ since we consider all d and thus implicitly both choices for $\mathrm{dgc}(C)$.

Proof. We construct an optimum placement with the stated slack for $\mathrm{V}(C)$.

In order to do so, we normalize the instance C_b first. Let $C = (e^0, \dots, e^{k-1})$ for $e^i = \{\,v^i, v^{i+1}\,\}$ ($v^k = v^0$). We assume $v_0 = \square$. This is no loss of generality since in case $\square \notin \mathrm{V}(C)$, we have some degree of freedom and may fix the position of v^0 arbitrarily in advance.

We point out that the distance bound slack (Definition 3.5) of a single bound depends on the placement only via the distance of its endpoints. This distance remains constant when endpoints are translated equally. Thus for a single $e = \{\,v, w\,\} \in \mathrm{E}(G)$ and $\delta \in \mathbb{R}^2$ arbitrary, $\mathrm{o}'_e(x) := \mathrm{o}_e(x) + \delta$ for $x \in e$ defines an equivalent bound that in every solution

has the same slack as e. We consider o' obtained by translating offsets of all $e_i \in \mathrm{E}(C)$ with different $\delta_i \in \mathbb{R}^2$, namely

$$\delta_i := \sum_{j=1}^{i-1} \Big(\mathrm{o}\Big(v^j_{e^{j-1}}\Big) - \mathrm{o}\Big(v^j_{e^j}\Big) \Big) - \mathrm{o}(v^0_{e^0})$$

where we abbreviate $\mathrm{o}_e(v)$ by $\mathrm{o}(v_e)$. For the resulting o' we compute $\mathrm{o}'(v^0_{e^0}) = \mathrm{o}(v^0_{e^0}) + \delta_0 = \mathrm{o}(v^0_{e^0}) - \mathrm{o}(v^0_{e^0}) = 0$ as well as for $i > 0$ and $\mu_i := \mathrm{o}(v^i_{e^{i-1}}) - \mathrm{o}(v^i_{e^i})$

$$\begin{aligned} \mathrm{o}'\Big(v^i_{e^{i-1}}\Big) &= \mathrm{o}\Big(v^i_{e^{i-1}}\Big) + \delta_{i-1} = \Big(\mathrm{o}\Big(v^i_{e^{i-1}}\Big) - \mu_i\Big) + (\delta_{i-1} + \mu_i) \\ &= \mathrm{o}\Big(v^i_{e^i}\Big) + \delta_i = \mathrm{o}'\Big(v^i_{e^i}\Big). \end{aligned}$$

Let $\mu := \mathrm{o}'(v^k_{e^{k-1}}) = \sum_{j \in [\![k]\!]} \mu_j$

By construction the instances C_b and $(C, \mathrm{b}, \mathrm{o}')$ are equivalent and thus it suffices to prove the statement for the latter one. Since $\mathrm{o}'(v^i_{e^{i-1}}) = \mathrm{o}'(v^i_{e^i})$ and $v^0 = \square$, the instance $(C, \mathrm{b}, \mathrm{o}')$ asks for a placement of the $k-1$ intermediate points between $0 \in \mathbb{R}^2$ and μ. The maximum slack achievable thereby is

$$s^* = \frac{1}{k} \left(-\|\mu\|_1 + \sum_{i \,\in\, [\![k]\!]} \mathrm{b}\Big(e^i\Big) \right).$$

and this can be realized by placing the points along any l_1-shortest 0-μ-path in $\mathbb{R}^2$ every $(\mathrm{b}(e^i) - s^*)$ steps.

On the other hand, for any direction d

$$\begin{aligned} f^d_\lambda\Big(\mathrm{dgc}\Big(e^0, \ldots, e^{k-1}\Big)\Big) &= \sum_{i \,\in\, [\![k]\!]} \big[\mathrm{b}(e_i) - \lambda\big] + \sum_{z \,\in\, \{\mathrm{x},\mathrm{y}\}} \sigma^d_z \cdot z\Big(\mathrm{o}'\Big(v^i_{e_i}\Big) - \mathrm{o}'\Big(v^{i+1}_{e_i}\Big)\Big) \\ &= -\sum_{z \,\in\, \{\mathrm{x},\mathrm{y}\}} \sigma^d_z \cdot z(\mu) + \sum_{i \,\in\, [\![k]\!]} \big[\mathrm{b}(e_i) - \lambda\big]. \end{aligned}$$

Consequently $f^d_\lambda(\mathrm{dgc}(e^0, \ldots, e^{k-1})) \geq 0$ if and only if $\lambda \leq \lambda^d$ for

$$\lambda^d := \frac{1}{k} \left(-\sum_{z \,\in\, \{\mathrm{x},\mathrm{y}\}} \sigma^d_z \cdot z(\mu) + \sum_{i \,\in\, [\![k]\!]} \mathrm{b}(e_i) \right).$$

We note that $\min_d \lambda^d$ is attained for a direction d with $\sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma^d_z z(\mu) = \|\mu\|_1$. Thus $\lambda^d = s^*$ for this direction which concludes the proof. ■

We have considered all building blocks to show the following main theorem:

Theorem 3.34. *Let (R, G_b) be an elementary instance of* Timing-Driven Rectangle Packing. *For the optimum slack s^* of (R, G_b) we have*

$$s^* = \min_d \Big\{ \mathrm{mr}\left(\mathrm{D}(G), f^d_\lambda\right) \Big\}.$$

Proof. Denote the righthand side of the equation by λ^*. Proposition 3.31 implies $s^* \geq \lambda^*$.

If $\mathrm{E}(G) = \varnothing$, there is no cycle $C \subseteq \mathrm{D}(G)$ with $\mathrm{E}(C) \neq \varnothing$. Consequently $\operatorname{mr}(\mathrm{D}(G), f_\lambda^d) = \infty$ for any direction d and $s^* = \infty = \lambda^*$.

Otherwise $\lambda^* < \infty$ since for any $e \in \mathrm{E}(G)$ there is the cycle $\operatorname{dgc}(e) \subseteq \mathrm{D}(G)$ with $\operatorname{dgc}(e) \neq \varnothing$. Let d be a direction attaining λ^* and $C \subseteq \mathrm{D}(G)$ a cycle with $f_{\lambda^*}^d(C) = 0$. Denote by $C' \subseteq G$ the subgraph with edges corresponding to $\mathrm{E}(C)$. Since C is a cycle, C' is either also a cycle or a single edge. Let s' denote the optimum slack of the restricted instance (R, C'_b) and $\lambda' := \min_d\{\operatorname{mr}(\operatorname{dgc}(C'_\mathrm{b}), f_\lambda^d)\}$.

As $C'_\mathrm{b} \subseteq G_\mathrm{b}$ by Lemma 3.19 $s^* \leq s'$. If C' is a single edge $s' = \lambda'$ by Lemma 3.32, otherwise by Lemma 3.33. Since both $\operatorname{dgc}(C'_\mathrm{b})$ and $\mathrm{D}(G)$ contain the overall minimizer C, we have $\lambda^* = \lambda'$. Altogether $s^* \leq \lambda^*$. ∎

Consequently we can find the optimum slack for elementary instances of the Timing-Driven Rectangle Packing Problem in strongly polynomial time:

Corollary 3.35. *The optimum slack of an elementary instance* (R, G_b) *of* Timing-Driven Rectangle Packing *can be computed in* $\mathcal{O}(nm)$ *time where* $n := |\mathrm{V}(G)|$ *and* $m := |\mathrm{E}(G)|$ *respectively.*

Proof. By construction the distance graph $\mathrm{D}(G)$ has the same complexity as G both of nodes and of edges. Using Theorem 3.34 we thus can compute the optimum slack in $\mathcal{O}(nm)$ time using the Minimum Mean Cycle Algorithm of Karp (1978). ∎

For such elementary instances of Timing-Driven Rectangle Packing we not only can determine the optimum slack. By a geometric argument we are also able to derive solutions attaining a given slack:

Proposition 3.36. *Let* (R, G_b) *be an elementary instance of* Timing-Driven Rectangle Packing *and* $s \in \mathbb{R}$. *We can either find a placement* l *with* $\operatorname{slack}(l, G_\mathrm{b}) = s$ *or decide that none exists at all in* $\mathcal{O}(nm)$ *time.*

Proof. We assume G is connected as we otherwise use the algorithm below for individual connected components, which results in the stated running time overall. Moreover we assume $\square \in \mathrm{V}(G)$ as we can otherwise fix an arbitrary $r \in R$ at $(0,0) \in \mathbb{R}^2$ and identify it with $\square := r$.

Let $\operatorname{dist}_d(v, w)$ denote the length of a shortest v-w-path in $(\mathrm{D}(G), f_s^d)$ for some direction d. We run the algorithm of Bellman (1958), Ford (1956) and Moore (1959) on $(\mathrm{D}(G), f_s^d)$ for $d \in \{\mathrm{NE}, \mathrm{SE}\}$ with source $\square$. This takes $\mathcal{O}(nm)$ time. We thereby either find a negative f_s^d-cycle in $\mathrm{D}(G)$ or compute $\operatorname{dist}_{\mathrm{SE}}(\square, \cdot)$ as well as $\operatorname{dist}_{\mathrm{NE}}(\square, \cdot)$.

If $\mathrm{D}(G)$ contains a negative f_s^d-cycle, there is no placement for G_b with slack s by Theorem 3.34.

Otherwise $\operatorname{dist}_{\mathrm{SE}}(\square, r)$ and $\operatorname{dist}_{\mathrm{NE}}(\square, r)$ are finite since $\square \in \mathrm{V}(G)$ and $\mathrm{D}(G)$ is strongly connected as G is assumed to be connected. We consider the unique rightmost point $l(r) \in \mathbb{R}^2$ with the given distances from $\square$ in SE and NE. The situation is visualized in Figure 3.6. Thus we define for any $r \in R \cup \{\square\}$

$$l(r) := \left(\frac{\operatorname{dist}_{\mathrm{NE}}(\square, r) + \operatorname{dist}_{\mathrm{SE}}(\square, r)}{2}, \frac{\operatorname{dist}_{\mathrm{NE}}(\square, r) - \operatorname{dist}_{\mathrm{SE}}(\square, r)}{2} \right) \in \mathbb{R}^2.$$

Automatically $l(\square) = 0 \in \mathbb{R}^2$.

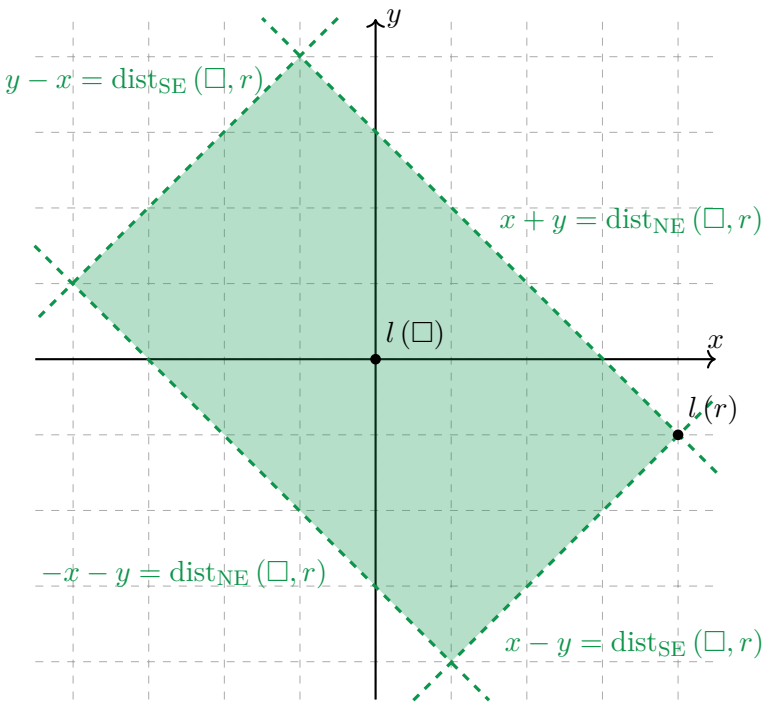


Figure 3.6: Visualization of the choice of $l(r)$ for the proof of Proposition 3.36.

We claim l is the desired placement with slack s: Let $e = \{v, w\} \in \mathrm{E}(G)$ where we select v s.t. e leaves v in direction d which is NE or SE, i.e. $\mathrm{x}\big(l(v_e)\big) \leq \mathrm{x}\big(l(w_e)\big)$. Finally

$$\begin{aligned}
\big\|l(v_e) - l(w_e)\big\|_1 &= \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot z\Big(\big[l(w) + \mathrm{o}_e(w)\big] - \big[l(v) + \mathrm{o}_e(v)\big]\Big) \\
&= \mathrm{dist}_d(\square, w) - \mathrm{dist}_d(\square, v) + \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot z\big(\mathrm{o}_e(w) - \mathrm{o}_e(v)\big) \\
&= \Big(\mathrm{dist}_d(\square, w) - \mathrm{dist}_d(\square, v) - f_s^d(v, w)\Big) + \mathrm{b}(e) - s \\
&\leq \mathrm{b}(e) - s,
\end{aligned}$$

since $\mathrm{dist}_d(\square, w) \leq \mathrm{dist}_d(\square, v) + f_s^d(v, w)$ for $(v, w) \in \mathrm{E}(\mathrm{D}(G))$ by Bellman's principle of optimality and conservativity of f_s^d. Consequently we conclude $\mathrm{slack}(l, G_{\mathrm{b}}) \geq s$. ■

Before we discuss consequences of Proposition 3.36, we point out that the proposed algorithm does not necessarily find a solution with minimum number of critical edges. An example where this is not the case, even when applied with the optimum slack s^* of the instance, is shown in Figure 3.7.

In this example the number of critical bounds could be reduced. This can be achieved by moving r_1 to the left whereby the worst slack attained at r_0 remains constant.

By Corollary 3.35 we can solve the Timing-Driven Rectangle Packing Problem on elementary instances in strongly polynomial time:

Corollary 3.37. *Let (R, G_{b}) be an elementary instance of* Timing-Driven Rectangle Packing. *We can compute an optimum solution in $\mathcal{O}(nm)$ time.*

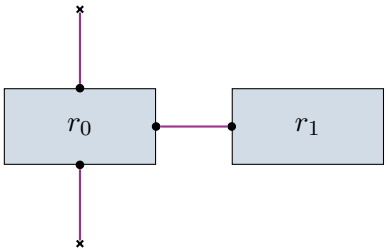


Figure 3.7: Example instance with uniform bounds where the algorithm of Proposition 3.36 does not find a solution with minimum number of critical edges.

Proof. By Corollary 3.35 we can find the optimum slack $s^* \in \mathbb{R}$ in $\mathcal{O}(nm)$ time. If $s^* = \infty$, we have $\mathrm{E}(G) = \varnothing$ and thus any placement is optimum. Otherwise we apply Proposition 3.36 for s^* in the same running time to obtain a solution l with $\mathrm{slack}(l, G_\mathrm{b}) = s^*$. ■

3.5 Distance Bound Pruning

In this section we work on efficient representations of Timing-Driven Rectangle Packing instances. Crucially we are not only interested in preserving the worst slack value but also any optimum solution. Therefore we mainly focus on pruning bounds that are in some way induced by other bounds.

First we focus on parallel bounds. We will see that for any pair $(r, r') \in R \times (R \cup \{\square\})$ a constant number of edges connecting r and r' are sufficient. The reason for this are geometric arguments known due to Chao et al. (1992) and Li et al. (2010). In the following we present a different, more thorough proof for a slightly stronger statement.

We start by formalizing the central geometric problem:

Definition 3.38. Let $Q := \{ (q_i, s_i) : i \in [\![n]\!] \}$ for $0 < n \in \mathbb{N}$, points $q_i \in \mathbb{R}^2$ and shifts $s_i \in \mathbb{R}$. The **maximum shifted point distance msd_Q** is the function $\mathrm{msd}_Q \colon \mathbb{R}^2 \to \mathbb{R}$ defined as

$$\mathrm{msd}_Q(p) := \max \Big\{ s + \|p, q\|_1 : (q, s) \in Q \Big\}.$$

By this definition msd_Q can clearly be evaluated in $\mathcal{O}(n)$ time. But this naïve approach can be improved: We develop a representation of msd_Q in $\mathcal{O}(n)$ pre-processing time that can be evaluated in $\mathcal{O}(1)$ time. In order to find such a representation we need the following building block:

Definition 3.39. A **Manhattan arc** is a set of convex combinations

$$A = \{\lambda a_0 + (1 - \lambda) a_1 : \lambda \in [0, 1]\} \subseteq \mathbb{R}^2$$

of $a_i \in \mathbb{R}^2$ with $|\mathrm{y}(a_0) - \mathrm{y}(a_1)| = |\mathrm{x}(a_0) - \mathrm{x}(a_1)|$, i.e. A is a closed line segment with angle $\frac{1}{4}\pi$ or $\frac{3}{4}\pi$ and endpoints a_i for $i \in [\![2]\!]$.

As we will see, Manhattan arcs are important since any function msd_Q can be expressed by the distance to such an arc (up to a constant). We further abbreviate $\|p, X\|_1 := \inf \Big\{ \|p, x\|_1 : x \in X \Big\}$ for any $X \subseteq \mathbb{R}^2$, in particular for Manhattan arcs.

We start by analyzing certain functions closely related to msd_Q. One instance of such a function is visualized in Figure 3.8. With this image in mind it is not surprising that such functions can be expressed as distance to a Manhattan arc.

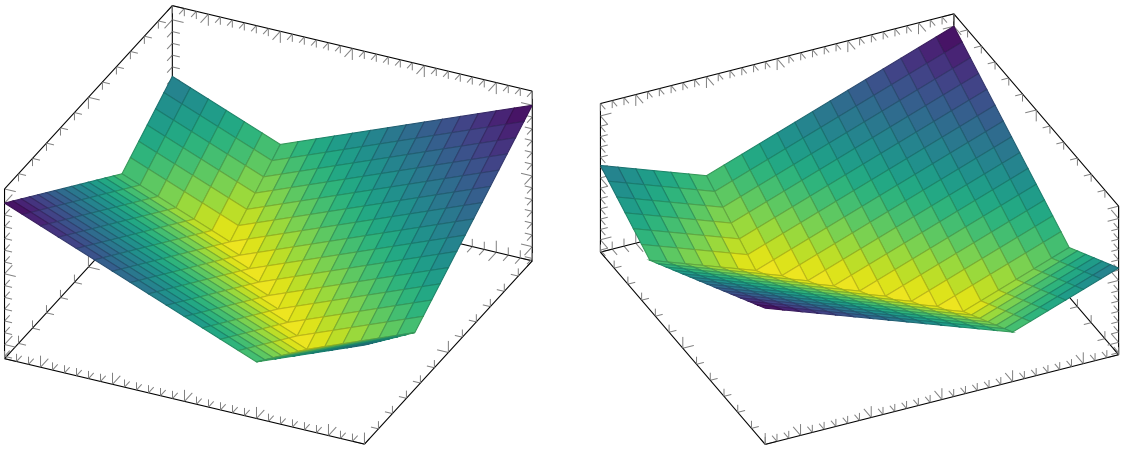


Figure 3.8: Visualization of the distance to a Manhattan arc as seen from two different angles.

Lemma 3.40. *Consider* $g_d\colon \mathbb{R}^2 \to \mathbb{R}$ *defined as* $g_d(p) := c_d + \sum_{z\in\{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot z(p)$ *for* $c_d \in \mathbb{R}$ *and* $d \in [\![4]\!]$. *Then there is Manhattan arc* A *and* $c \in \mathbb{R}$ *s.t.* $g(p) := \max\{\, g_d(p) : d \in [\![4]\!] \,\} = c + \|p, A\|_1$ *for all* $p \in \mathbb{R}$.

Proof. We first consider $h_i\colon \mathbb{R}^2 \to \mathbb{R}$ defined as $h_i(p) := \max\{\, g_i(p), g_{i+2}(p) \,\}$ for $i \in [\![2]\!]$. The function h_0 (for NE and SW) attains its minimum $m_0 := \frac{1}{2}(c_0 + c_2)$ along the straight line L_0 of points (x, y) with $x+y = \frac{1}{2}(c_2 - c_0)$. Moreover above (below) L_0 the function h_0 is defined by g_0 (g_2) and thus $h_0(p) = m_0 + \|p, L_0\|_1$. By symmetry $h_1(p) = m_1 + \|p, L_0\|_1$ where $m_1 := \frac{1}{2}(c_1 + c_3)$ and L_1 is the straight line of points (x, y) with $x - y = \frac{1}{2}(c_1 - c_3)$.

Possibly after rotating by $\frac{1}{2}\pi$ we may assume w.l.o.g. $m_1 \leq m_0$. As $g(p) = \max\{\, h_i(p) : i \in [\![2]\!] \,\}$, g attains its minimum $c := m_0$ on $A := L_0 \cap \mathrm{B}_\delta(L_1) \subseteq \mathbb{R}^2$ for $\delta := m_0 - m_1$. Since L_0 and L_1 are orthogonal, A is bounded. Moreover as $A \subseteq L_0$, A is a Manhattan arc. This construction is depicted in Figure 3.9.

Let $Q_i \subseteq \mathbb{R}^2$ for $i \in [\![4]\!]$ denote the points where g is defined by g_i. The geometry of Q_i can be observed in Figure 3.9. For $p \in Q_i$ we thus have $g(p) = g_i(p) = h_j(p) = m_j + \left\|p, L_j\right\|_1$ for $j \in [\![2]\!]$ with $i \equiv j \pmod 2$. By construction $\|p, L_0\|_1 = \|p, A\|_1$ for $p \in Q_0 \cup Q_2$. Denote by $q_i \in \mathbb{R}^2$ the unique point in $Q_i \cap A$ for $i \in \{1, 3\}$. Thus for any $p \in Q_i$ where $i \in \{1, 3\}$, we have $\|p, L_1\|_1 = \|p, q_i\|_1 + \delta = \|p, A\|_1 + \delta$. Clearly $\bigcup_{i\in[\![4]\!]} Q_i = \mathbb{R}^2$ and thus in any case $g(p) = c + \|p, A\|_1$ as claimed. ∎

Note that the proof of Lemma 3.40 is constructive and explicitly states how to compute Manhattan arc A and shift c. Moreover all steps involved can be performed in constant time altogether.

We can use this lemma in order to find an efficient representation of maximum shifted point distance functions:

Proposition 3.41. *Consider any maximum shifted point distance* msd_Q. *We can compute* $s \in \mathbb{R}$ *and a Manhattan arc with endpoints* $a_i \in \mathbb{R}^2$ *for* $i \in [\![2]\!]$ *with* $\mathrm{msd}_Q \equiv \mathrm{msd}_{Q^*}$ *where* $Q^* := \{\, (a_i, s) : i \in [\![2]\!] \,\}$ *in* $\mathcal{O}(|Q|)$ *time.*

Proof. Let $Q = \{\, (q_i, s_i) : i \in [\![n]\!] \,\}$. We define $\varphi_d\colon \mathbb{R}^2 \to \mathbb{R}$ as

$$\varphi_d(p) := \sum_{z\in\{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot z(p)$$

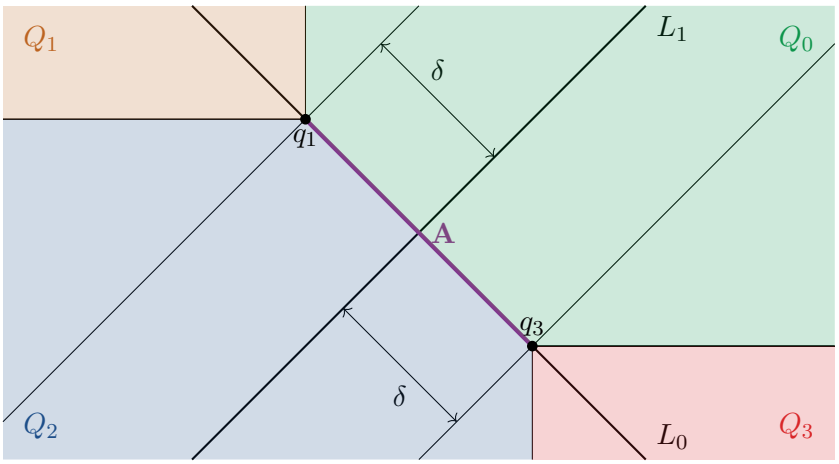


Figure 3.9: Construction of the Manhattan arc A in the proof of Lemma 3.40.

for each direction $d \in [\![4]\!]$. Further let $c_d^i := s_i - \varphi_d(q_i)$ for $i \in [\![n]\!]$ and $d \in [\![4]\!]$.

As $\|p,q\|_1 = \max\{\varphi_d(p) - \varphi_d(q) : d \in [\![4]\!]\}$, $s_i + \|p,q_i\|_1 = \max\{c_d^i + \varphi_d(p) : d \in [\![4]\!]\}$ for all $p \in \mathbb{R}^2$. By Definition 3.38 of msd_Q, we deduce

$$\mathrm{msd}_Q(p) = \max\Big\{\max\Big\{c_d^i + \varphi_d(p) : i \in [\![n]\!]\Big\} : d \in [\![4]\!]\Big\}.$$

For any direction d, all $c_d^i + \varphi_d$ are parallel affine linear functions. Thus $\mathrm{msd}_Q(p) = \max\{c_d + \varphi_d(p) : d \in [\![4]\!]\}$ for $c_d := \max\{c_d^i : i \in [\![n]\!]\}$. Consequently we can apply Lemma 3.40 and obtain a Manhattan arc $A \subseteq \mathbb{R}^2$ and $s' \in \mathbb{R}$ s.t. $\mathrm{msd}_Q(p) = s' + \|p,A\|_1$ for all $p \in \mathbb{R}^2$.

Let a_i' for $i \in [\![2]\!]$ be the endpoints of A. By Definition 3.39 of Manhattan arcs, a_i' are opposite corners of a square $S \subseteq \mathbb{R}^2$. Let $\delta \in \mathbb{R}$ denote the edge length of S. We finally define the desired $s := s' - \delta$ and a_i for $i \in [\![2]\!]$ to be the corners of S other than a_i'. As a_i' are the endpoints of a Manhattan arc, so are a_i. Consequently for any $p \in \mathbb{R}^2$

$$\mathrm{msd}_Q(p) = s' + \|p,A\|_1 = \Big(s' - \delta\Big) + \max\Big\{\|p,a_i\|_1 : i \in [\![2]\!]\Big\} = \mathrm{msd}_{Q^*}(p).$$

It takes $\mathcal{O}(|Q|)$ time to determine all c_d, everything else can be computed in constant time. ■

Equipped with Proposition 3.41 we return to distance bounds. It allows the following pruning of parallel distance bound edges:

Corollary 3.42. *Consider an instance (R, G_b) of* Timing-Driven Rectangle Packing *with $r \in R$ and $r' \in R \cup \{\square\}$. Denote all bounds for r and r' by $P := \mathrm{E}(G[\{r,r'\}])$. We can compute an extension of* b *and* o *to $e_0 := e_1 := \{r,r'\}$ in $\mathcal{O}(|P|)$ time s.t. $\mathrm{slack}(l, G_\mathrm{b}) = \mathrm{slack}(l, H_b)$ for any placement l where $H := G - P + \{e_0, e_1\}$.*

Proof. To simplify notation, we first translate $\mathrm{o}_p(\,\cdot\,)$ by $-\mathrm{o}_p(r)$ for all $p \in P$ (as has already been done before e.g. in the proof of Lemma 3.33). Thus we may assume $\mathrm{o}_p(r) = 0$ for all $p \in P$.

Let $G' := G[\{r, r'\}]$. For any placement l, $\operatorname{slack}\left(l, G'_{\mathrm{b}}\right)$ depends only on the distance of $l(r)$ and $l(r')$. Thus we also assume fixed $l(r') = 0 \in \mathbb{R}^2$ without loss of generality for the following arguments.

By Definition 3.5 of slack, this implies

$$\operatorname{slack}\left(l, G'_{\mathrm{b}}\right) = -\max\left\{ \left\| l(r) - \mathrm{o}_p\left(r'\right) \right\|_1 - \mathrm{b}(p) : p \in P \right\} = -\operatorname{msd}_Q(l(r))$$

for $Q := \left\{ (\mathrm{o}_p(r'), \mathrm{b}(p)) : p \in P \right\}$. Thus by Proposition 3.41 there are $a_i \in \mathbb{R}^2$ for $i \in [\![2]\!]$ and $s \in \mathbb{R}$ s.t. $\operatorname{msd}_Q \equiv \operatorname{msd}_{Q'}$ for $Q' := \{ (a_i, s) : i \in [\![2]\!] \}$. Consequently $\mathrm{o}_{e_i}(r') := a_i$ and $\mathrm{b}(e_i) := s$ for $i \in [\![2]\!]$ defines the two bounds that are equivalent to P.

The running time is dominated by applying Proposition 3.41 and thus $\mathcal{O}(|P|)$. ■

Note that the argument of translating distance bound offsets used for Corollary 3.42 requires fixed orientations for all rectangles. If we would consider variable orientations as part of the Timing-Driven Rectangle Packing Problem, this statement no longer holds.

Using this corollary for parallel edges in G_{b} we obtain the following algorithm:

Proposition 3.43. *For an elementary instance of* Timing-Driven Rectangle Packing*, we can compute an optimum solution in $\mathcal{O}(n^3 + m)$ time.*

Proof. Let (R, G_{b}) denote the elementary instance. First we prune parallel edges in G using Corollary 3.42. This takes $\mathcal{O}(m)$ time overall and leaves at most n^2 edges. On the remaining instance the algorithm of Corollary 3.37 needs $\mathcal{O}(n^3)$ time. ■

We emphasize that in practice there are many parallel edges in G_{b}. Therefore Proposition 3.43 states an important improvement for those cases.

Now we turn our attention to more complex configurations involving paths. Note that due to Lemma 3.17, a solution l with maximum $\operatorname{slack}(l, G_{\mathrm{b}})$ corresponds to the maximum $s \in \mathbb{R}$ for which $\operatorname{slack}(l, G_{\mathrm{b}-s}) \geq 0$. For paths however, this relaxation is added to each edge. Consequently for large enough s a single edge $e \in \mathrm{E}(G)$ can never be dominated by a path P with $\mathrm{E}(P) \geq 2$.

An example for this is presented in Figure 3.10: Either $P_0 := (e_0, e_1)$ or $P_1 := (e_0, f_0, f_1)$ can define the optimum position for r: For fixed $\mathrm{b}(e_i)$, the optimum slack $s = \frac{1}{2}(\|p, q\|_1 - \mathrm{b}(e_0) - \mathrm{b}(e_1)) \in \mathbb{R}$ can be arbitrary depending on $\|p, q\|_1$. In this case P_1 defines the most restrictive constraints for r if and only if $s \geq \mathrm{b}(f_0) + \mathrm{b}(f_1) - \mathrm{b}(e_1)$.

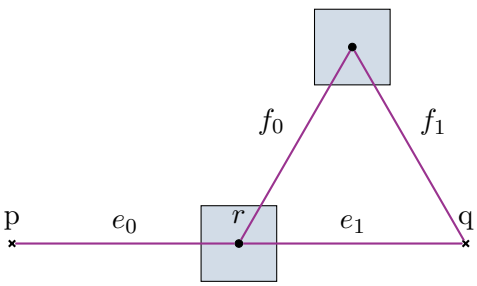


Figure 3.10: Example where either path (e_0, e_1) or (e_0, f_0, f_1) can induce the tightest constraints for r.

In practice however, we can use a priori bounds on the slack (e.g. lower bounds by legal placements or upper bounds by Corollary 3.37). Consequently we may restrict the considered slack to some interval $S \subseteq \mathbb{R}$. The length of this range S will affect the effectiveness of the following pruning strategies.

Lemma 3.44. *Consider an instance of* Timing-Driven Rectangle Packing *with* $S \subseteq \mathbb{R}$. *Let* $e^* = \{u, w\} \in \mathrm{E}(G)$ *and* $P = (e^0, \dots, e^{k-1})$ *for* $e^i = \{v^i, v^{i+1}\} \in \mathrm{E}(G)$, $0 < k \in \mathbb{N}$ *a* u-w*-path in* G. *Further assume* $e^* \neq e^i$ *for* $i \in [\![k]\!]$, *as well as for all* $s \in S$

$$\sum_{i \in [\![k]\!]} \Big(\mathrm{b}(e^i) - s \Big) + \sum_{i \in [\![k+1]\!]} \left\| \delta^i \right\|_1 + \left\| \delta^* \right\|_1 \leq \mathrm{b}(e^*) - s,$$

where $\delta^i \in \mathbb{R}^2$ *are defined as* $\delta^i := \mathrm{o}(v^{i+1}_{e^{i+1}}) - \mathrm{o}(v^{i+1}_{e_i})$ *for* $i \in [\![k-1]\!]$ *as well as* $\delta^* := \mathrm{o}(v^0_{e^0}) - \mathrm{o}(v^0_{e^*}) + \mathrm{o}(v^k_{e^*}) - \mathrm{o}(v^k_{e^{k-1}})$.

In that case every solution l *with* $\mathrm{slack}(l, H_\mathrm{b}) \in S$ *satisfies* $\mathrm{slack}(l, H_\mathrm{b}) = \mathrm{slack}(l, G_\mathrm{b}) \in S$ *where* $H := G - e^*$.

Proof. To simplify notation we also use the shorthand $\mathrm{o}(v_e)$ for $\mathrm{o}_e(v)$. We can translate $\mathrm{o}_{e^*}(\,\cdot\,)$ by $\mathrm{o}(v^0_{e^0}) - \mathrm{o}(v^0_{e^*})$. Thus we may assume $\mathrm{o}(v^0_{e^*}) = \mathrm{o}(v^0_{e^0})$ and $\mathrm{o}(v^k_{e^*}) - \mathrm{o}(v^k_{e^{k-1}}) = \delta^*$.

As $\mathrm{slack}(l, H_\mathrm{b}) \in S$ and $e^i \neq e^*$ we have $\|l(e^i)\|_1 \leq \mathrm{b}(e^i) - \mathrm{slack}(l, H_\mathrm{b})$. Consequently by the triangle inequality along P

$$\begin{aligned}
\left\| l(e^*) \right\|_1 &\leq \sum_{i \in [\![k]\!]} \left\| l(e^i) \right\|_1 + \sum_{i \in [\![k-1]\!]} \left\| \mathrm{o}(v^{i+1}_{e^{i+1}}) - \mathrm{o}(v^{i+1}_{e^i}) \right\|_1 + \left\| \mathrm{o}(v^k_{e^*}) - \mathrm{o}(v^k_{e^{k-1}}) \right\|_1 \\
&\leq \sum_{i \in [\![k]\!]} \Big[\mathrm{b}(e^i) - \mathrm{slack}(l, H_\mathrm{b}) \Big] + \sum_{i \in [\![k-1]\!]} \left\| \delta^i \right\|_1 + \left\| \delta^* \right\|_1 \\
&\leq \mathrm{b}(e^*) - \mathrm{slack}(l, H_\mathrm{b})
\end{aligned}$$

Thus $\mathrm{slack}(l, H_\mathrm{b}) \leq \mathrm{slack}(l, \{e^\star\}_\mathrm{b})$ by which $\mathrm{slack}(l, H_\mathrm{b}) \leq \mathrm{slack}(l, G_\mathrm{b})$. According to Lemma 3.19 $\mathrm{slack}(l, G_\mathrm{b}) \leq \mathrm{slack}(l, H_\mathrm{b})$. Consequently $\mathrm{slack}(l, G_\mathrm{b}) = \mathrm{slack}(l, H_\mathrm{b}) \in S$. ■

This lemma has an immediate implication on the set of optimum solutions preserved:

Corollary 3.45. *Consider the setting of Lemma 3.44. If* S *contains the optimum slack, we have* $\mathrm{opt}(G_\mathrm{b}) = \mathrm{opt}(H_\mathrm{b})$ *for the set of optimum solutions* $\mathrm{opt}(X)$ *of* (R, X).

Proof. Any solution l with $\mathrm{slack}(l, H_\mathrm{b}) \in S$ satisfies $\mathrm{slack}(l, H_\mathrm{b}) = \mathrm{slack}(l, G_\mathrm{b})$ by Lemma 3.44. This particularly holds for optimum l for either H_b or G_b. ■

3.6 LP-Formulation for Fixed Relations

Practical algorithms for the Rectangle Packing Problem often use a branch-and-bound approach on the relations of the rectangles. This is why the special case of solving a Rectangle Packing Problem with fixed relations is interesting.

Primal Timing-Driven Rectangle Packing LP

For the timing-driven version of this problem, we can also use an LP formulation. Therefore we denote by $z_{\min}(r), z_{\max}(r)$ the boundaries of the feasible area $A(r)$ of $r \in R$ in dimension $z \in \{\mathrm{x}, \mathrm{y}\}$. Moreover we use the shorthand $r \lhd_z r'$ for $r \lhd r'$ and $r \rhd r'$ depending on whether $z = \mathrm{x}$ or $z = \mathrm{y}$.

With this notation we formulate the following primal linear program:

$$
\begin{aligned}
&\max && s && && \\
&\text{s.t.} && -z_r \le -z_{\min}(r) && (r \in R;\, z \in \{\mathrm{x},\mathrm{y}\}) && (3.4a)\\
& && z_r \le z_{\max}(r) && && \\
& && z_r - z_{r'} \le -\operatorname{len}(r,z) && (r, r' \in R;\, r \lhd_z r';\, z \in \{\mathrm{x},\mathrm{y}\}) && (3.4b)\\
& && z_e^- - z_r \le z\big(\mathrm{o}_e(r)\big) && && \\
& && z_r - z_e^+ \le -z\big(\mathrm{o}_e(r)\big) && (r \in R;\, e \in \delta(r);\, z \in \{\mathrm{x},\mathrm{y}\}) && (3.4c)\\
& s + \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \Big(z_e^+ - z_e^-\Big) \le \mathrm{b}(e) && (e \in \mathrm{E}(G)) && (3.4d)
\end{aligned}
$$

Constraints 3.4a ensure coordinates z_r are within the feasible area of rectangle r. Disjointness in dimension z according to the fixed relations is guaranteed by constraints 3.4b. Constraints 3.4c describe the defining properties of the maximum/minimum z-coordinate z_e^+/z_e^- of bound $e \in \mathrm{E}(G)$. Using these extrema, constraints 3.4d model the defining property of the worst slack s.

The presented linear program 3.4 is general enough to include rectangles and pins with fixed locations: Fixed rectangles can be modeled by appropriate feasible areas for constraints 3.4a. We can further include the chip area $\square$ in R as artificial rectangle without any disjointness constraints 3.4b. This rectangle $\square$ can be fixed at at $0 \in \mathbb{R}^2$ as mentioned previously.

Handling both those cases implicitly allows an easier notation for the primal program 3.4. Note that all constraints of this LP are also normalized in order to easily look at the corresponding dual LP.

Recall that for timing unaware Rectangle Packing with fixed relations and e.g. netlength objective we can formulate a very similar LP. For this purpose, we would consider a different graph G (namely the netlist hypergraph), simply eliminate constraints 3.4d and optimize $\sum_{e \in \mathrm{E}(G)} \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \Big(z_e^+ - z_e^-\Big)$ as new objective.

Altogether constraints 3.4d are the essential difference which have the following implications on the dual LP.

Dual Timing-Driven Rectangle Packing LP

We name dual variables f_t^i corresponding to the inequality t in the ith group of the primal linear program 3.4, i.e. for example $f_{r,z}^0$ and $f_{r,z}^1$ for constraints 3.4a.

In this notation we denote the objective function of the dual LP by

$$
\begin{aligned}
\nu(f) := &\sum_{z;\, r} \Big(z_{\max}(r) \cdot f_{r,z}^1 - z_{\min}(r) \cdot f_{r,z}^0\Big)\\
&+ \sum_{z;\, r \lhd_z r'} -\operatorname{len}(r,z) \cdot f_{r,r',z}^2\\
&+ \sum_{z;\, r;\, e \in \delta(r)} z\big(\mathrm{o}_e(r)\big)\Big(f_{r,e,z}^3 - f_{r,e,z}^4\Big)\\
&+ \sum_{e \in \mathrm{E}(G)} \mathrm{b}(e) \cdot f_e^5.
\end{aligned}
$$

Using this notation,we have the following LP dual to linear program 3.4:

$$\begin{aligned}
&\max && \nu(f) && && \\
&\text{s.t.} && f_t^i \geq 0 && && (3.5a)\\
& && \sum_{e\in \mathrm{E}(G)} f_e^5 = 1 && && (3.5b)\\
& && \sum_{r\in e} f_{r,e,z}^3 - f_e^5 = 0 && && \\
& && f_e^5 - \sum_{r\in e} f_{r,e,z}^4 = 0 && (e\in \mathrm{E}(G);\ z\in\{x,y\}) && (3.5c)\\
& \left(f_{r,z}^1 + \sum_{r':r\lhd_z r'} f_{r,r',z}^2 + \sum_{e\in\delta(r)} f_{r,e,z}^4\right) && && \\
& -\left(f_{r,z}^0 + \sum_{r':r'\lhd_z r} f_{r,r',z}^2 + \sum_{e\in\delta(r)} f_{r,e,z}^3\right) = 0 && (r\in R;\ z\in\{x,y\}) && (3.5d)
\end{aligned}$$

Note that constraint 3.5b corresponds to variable s in the primal LP, constraints 3.5c are caused by z_b^- and z_b^+ while constraints 3.5d correspond to primal coordinate variables z_r. Non-negativity constraints 3.5a arise naturally from duality.

The dual linear program 3.5 is similar to the dual LP for the regular Rectangle Packing Problem. The latter is equivalent to a minimum-cost flow problem. But this is no longer the case for program 3.5:

The timing-driven dual LP contains f_e^5 variables corresponding to constraints 3.4d of the primal LP. The constraints on those variables are no ordinary flow constraints simply because f_e^5 appears in 5 constraints (and ordinary flow variables appear at most twice). For timing-unaware dual LP formulation, there would be no problematic variables f_e^5 which is why this dual is a minimum-cost flow problem.

We could split the dimension independent f_e^5 into $f_{e,\mathrm{x}}^5$ and $f_{e,\mathrm{y}}^5$. If we disregard constraint 3.5b for a moment, the resulting problem is a flow problem with additional equality-constraints $f_{e,\mathrm{x}}^5 = f_{e,\mathrm{y}}^5$.

Flow-problems with additional equality-constraints have been studied a little before. They arise naturally from computing generalized matchings. Such *Balanced Network Flows* problems have been studied by Kocay and Stone (1993, 1995) and Goldberg and Karzanov (2004) in skew-symmetric, bipartite graphs. With this combinatorial structure it is in fact easy to find a fractional flow satisfying the equality-constraints. Integral max-flows can be computed by augmenting along pairs of paths. For balanced flows in arbitrary graphs or with more general equality-constraints as arising in linear program 3.5, we are not aware of any results – even in the unweighted case.

Chapter 4

Timing-Driven Macro Placement

This chapter is devoted to BonnMacro, the framework for Macro Placement developed at the Research Institute for Discrete Mathematics of the University of Bonn. Overviews of earlier versions of BonnMacro have been presented by Brenner (2007), Brenner, Struzyna and Vygen (2008), Funke, Hougardy and Schneider (2016) and Schneider (2009).

BonnMacro is applicable to the Macro Placement Problem introduced in Section 2.2.3 (page 10). It specifically targets cells whose placement highly impacts the overall solution i.e. large cells. We call these cells **macros**, although they are technically not required to be large and it may even be reasonable to add certain standard cells to BonnMacro instances. Recall that by Definition 2.7 all cells and thus particularly macros have rectangles as cell shape.

Similar to other tools presented in Section 2.3 (page 11), BonnMacro initially determines possibly illegal positions for all cells, i.e. particularly macros. During this stage we minimize (weighted) bounding box netlength subject to density constraints. This is done using the global placement algorithm BonnPlace on an artificial netlist. In Section 4.1 we elaborate details of this approach.

Afterwards inter-macro overlaps need to be resolved. This stage is referred to as **macro legalization**. We model macro legalization as Rectangle Packing Problem subject to minimizing movement from the input locations. As Rectangle Packing is hard (Theorem 3.16) the resulting instances with up to hundreds of rectangles can not be solved optimally in practice.

In order to overcome this complexity, we successively legalize each macro optimally in a smaller, local neighborhood to optimality. Thereby we make sure that no overlaps to previously legalized macros can be introduced. When this can be completed for all macros successfully, we end up with a feasible macro placement.

Details of the hinted workflow are elaborated in Section 4.2, particularly how local instances are chosen and additional constraints are modeled in the Rectangle Packing Problem. Afterwards in Section 4.3 we extend this framework to minimize violations of timing constraints.

From this point onward, BonnMacro always preserves legality in the macro placement. In additional post-optimization steps further objectives can be considered. We are able to optimize for sufficient whitespace (e.g. for subsequent buffer insertion), macro alignment (for improved routability along straight channels) or even netlength of inter-macro connections. These types of macro post-optimization will be considered in Section 4.4.

Neither the overall BonnMacro approach nor single steps (e.g. macro legalization) have any theoretical guarantees and even may fail on instances that are solvable feasibly. Despite that, BonnMacro works very well in practice and finds competitive solutions on a large variety of real-world instances. It is used regularly at IBM for interactive workflows (e.g. manual illegal changes combined with BonnMacro legalization) in order to gain insights in early design phases.

4.1 Shredded Placement

In this section we explain how the initial placement for BonnMacro is computed. These initial locations are not required to obey all legality constraints e.g. may contain overlaps and can ignore grid constraints entirely. On the other hand, this first solution is the starting point for the subsequent macro legalization (cf. Section 4.2). Thus it should in particular have reasonable netlength and be well spread i.e. roughly respect density constraints.

In theory, this looks like a problem similar to Global Placement of standard cells. As has already been elaborated in Section 2.3 (page 11), directly incorporating macro-handling into global placement paradigms is complicated. Both predominantly used placement approaches, force-directed and partitioning-based, have intrinsic drawbacks when applied to mixed-size instances.

A common idea to overcome any issues with large cells is **macro shredding**. This approach has been described first by Adya and Markov (2005) and Doll, Johannes and Antreich (1994). The basic idea is to replace large macros by multiple artificial small cells which we call **macro fragments** (cf. Figure 4.1). Special nets connecting these cells ensure that all fragments of a macro are placed close to each other. Most importantly, instances resulting from this type of macro shredding can directly be processed by any existing global placement engine. A location for original macros can afterwards be inferred from the respective fragments' positions.

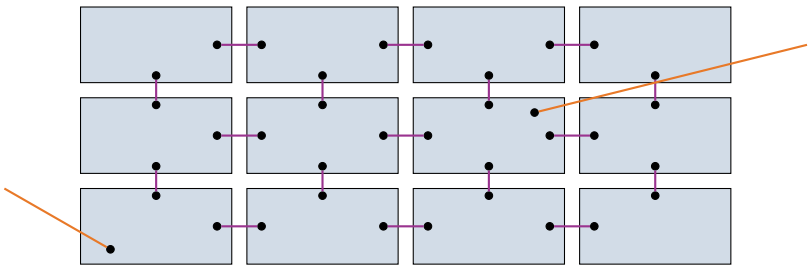


Figure 4.1: Macro representation by fragments. Artificial fragment interconnections are indicated in violet, original nets in orange. Pin positions in original nets match offsets of the original (unshredded) macro.

Although this is a very simple idea it is very effective in practice and still in use today. Clearly shredding a single macro into multiple fragments increases the netlist complexity. Thus for solving real world instances with shredded macros, it is particularly important to use a very fast and reliable global placer.

For this reason BonnMacro makes use of BonnPlaceGlobal for solving the Global Placement Problem, which is another component of BonnPlace also developed at Research Institute for Discrete Mathematics of the University of Bonn. Further details concerning this partitioning-based placer will be presented in Chapter 5.

Figure 4.2: The placement with shredded macro fragments on Rosemarie. Fragments colored according to the original macros.

4.1.1 Placement with Shredded Macros

We now elaborate considerations for macro shredding that we employ in order to infer reliable location details using BonnPlace.

As a first adjustment we will neither execute the last levels of global placement nor detailed placement. During a recursive partitioning scheme like BonnPlace, the assignment of fragments becomes more or less fixed in very fine grids. This is due to the fact that separating tightly connected macro fragments always has a negative effect on netlength. On the other hand, all macro fragments together will not be part of a common partitioning instance in a sufficiently fine grid. Since only locations of macro fragments matter anyway, this has a minor effect on the output and saves unnecessary running time.

The size of the fragments is chosen significantly smaller than the finest grid window of the last level performed. Thus we ensure no macro fragments ever require to be fixed in place during any BonnPlace step.

Additionally macros should be placed more densely than standard cells – at least as long as we are able to resolve all overlaps subsequently. Thus we scale fragments uniformly such that placing them with (standard cell) target density (usually 75 %) results in a user specified macro density (usually above 80 %).

All macro fragments are connected in a regular grid of highly weighted two-terminal nets (cf. Figure 4.1). The pins of these nets are not directly attached to the fragments' centers but rather shifted towards the outlines. This helps finding distinct locations for fragments during analytical placement of BonnPlace, which in turn makes movement minimization during partitioning more effective.

In order to infer reasonable positions for original macros, fragments need to be placed closely together. To enforce this, we use two different techniques: First we choose the net weights for fragment interconnections significantly above the standard net weight. Our choice of weights thereby also takes macro connectivity into account which results in larger weights for macros with many pins.

On the other hand, we also cluster macro fragments. Clustering is a common technique for global placement algorithms by which groups of multiple original cells are considered as single artificial cell. This simplifies Global Placement which is why algorithms run faster and often can compute better results. BonnPlace also supports clustering. In each BonnPlace level, we adaptively dissolve clusters which became too large for the respective partitioning grid. In contrast to global placement on regular netlists, we can further delay such unclustering on netlists with clustered shredded macros.

We assign all macro fragments to the movebounds of their respective original macro. This is why the shredded placement always already considers movebounds directly.

Finally we also use the self-stabilizing BonnPlace extension which we elaborate in Section 5.2. In this force-directed framework we essentially perform multiple iterations of complete global placement. For netlists with shredded macros we only run few (usually 3) iterations but use a larger force weight increase (usually 0.1).

A placement for shredded macros on Rosemarie computed by this approach is presented in Figure 4.2. It can be observed that fragments are placed closely together and mostly arrange in a shape which roughly is a rectangle. But these shapes are far from perfect and often not entirely disjoint. Note that seemingly the respective fragments are smaller than the original macros as visualized in Figure 4.3. But actually this is not the case: due to clustering and reduced number of placement levels, fragments frequently overlap in the finest grid considered. This explains the visual impression.

4.1.2 Macro Reassembly

Based on positions for shredded macro fragments we infer macro locations. An important part of this **macro reassembly** also is the choice of an appropriate orientation for the macros.

We proceed similar to Adya and Markov (2005) and use the center of gravity of the fragments as position (for the center of gravity) of the original macro. The macro flip-code is selected as the predominant orientation among the respective fragments.

There have been experiments to emphasize actual pin positions and relative positions of corner macro fragments (Engels 2013). Such considerations usually result in macro placements with less netlength but increase overlaps. This is why this approach is only used for special cases, particularly not for experiments presented in this thesis.

The macro placement resulting from our reassembly based on the fragments' locations of Figure 4.2 is depicted in Figure 4.3. Note that macros are colored equally as their respective fragments.

4.2 Macro Legalization

For a given, possibly illegal input placement, BonnMacro resolves all macro overlaps as a next step. The main idea is to iteratively legalize macros one after another with minimum movement. An outline of the overall algorithm is given in Algorithm 4.1.

MacroLegalization maintains a set $L \subseteq M$ of **legalized** macros that is initially empty (Algorithm 4.1, line 1). We always preserve the invariant that l is a legal solution for the Macro Placement Problem restricted to L. In particular, all macros in L have

Figure 4.3: The reassembled macro placement based on the fragments' positions of Figure 4.2 on Rosemarie.

to be placed on-grid and in their movebound. Moreover macros in L are disjoint from other legalized macros and blockages.

In each iteration of the main loop (Algorithm 4.1, lines 2 to 10), we try to change the placement l such that above invariant is satisfied for $L \cup \{m\}$. For this purpose we try to squeeze m between neighboring blockages and legalized macros in L.

As more and more macros are already legalized, it becomes harder to find a legal position for the next macro m. This is why the order matters in which macros are legalized. In our approach we impose an a priori ordering by **priority** of the macros (Algorithm 4.1, line 2). We assign macros high priority whose positioning greatly influences the space of remaining feasible solutions. Therefore we lexicographically prioritize macros with non-trivial movebounds, with larger size and smaller distance to the chip boundary (in this order).

Packing a given macro m is achieved by PackMacro (Algorithm 4.1, line 5). Here the problem of legalizing m is modeled as geometric Rectangle Packing Problem. Since this problem is NP-hard by Theorem 3.16, we only can handle small packing instances with few rectangles. In this model we thus carefully ensure disjointness to macros in L that can not be considered explicitly. We defer all details concerning this model to Section 4.2.1. An example instance for PackMacro is visualized in Figure 4.4. Thereby each such Rectangle Packing Problem instance either can not be solved at all or implies an extension of l enhancing MacroLegalization invariants to $L \cup \{m\}$.

As we will see, selecting local neighborhoods for m depends of various parameters that effectively trade running time for overall solution quality. We call a reasonable combination

Algorithm 4.1: MacroLegalization

Input : Instance of the Macro Placement Problem, possibly illegal target locations t.
Output: Legal locations for all macros or failure.

```
1  L := ∅, l := t
2  foreach macro m in descending order of priority do
3      S := ∅
4      foreach packing mode pm do
5          s := PackMacro(m, L, pm, l, t)
6          if s is not failure then
7              S := S ∪ {s}
8      if S = ∅ then
9          return failure
10     Extend l by BestSolution(S, t), L := L ∪ {m}
11 return l
```

of these parameters **packing mode**. For better quality, we try to pack m with various packing modes (Algorithm 4.1, lines 4 to 7). For each mode pm for which this is possible, we save the obtained solution in S (lines 6 to 7).

MacroLegalization minimizes total area-weighted movement of l with respect to target locations t. How PackMacro optimizes this objective precisely will be covered in Section 4.2.1. Overall BestSolution greedily chooses a best solution amongst S with regard to the same objective (Algorithm 4.1, line 10).

Any solution in S extends MacroLegalization invariants to $L \cup \{m\}$. Consequently if MacroLegalization never returns from line 9, $L = M$ and thus l is a feasible solution overall. We summarize the preceding elaboration in the following proposition:

Proposition 4.1. MacroLegalization *(Algorithm 4.1) either fails or finds a feasible solution to the* Macro Placement Problem. ■

Due to the restriction to consider only local packing instances, MacroLegalization can not guarantee to find a feasible solution even on instances for which one exists.

4.2.1 Macro Packing

The main building block of MacroLegalization is PackMacro. An overall description of the latter algorithm is presented in Algorithm 4.2.

PackMacro consists of two phases: Selecting appropriate details from the Macro Placement instance for modelling as Rectangle Packing Problem (Algorithm 4.2, lines 1 to 5) and solving the afore-built instance (lines 6 to 10). Recall that any subcomponent can fail, which in turn causes PackMacro to fail as well (lines 8 to 9).

We presented multiple strategies for solving the Rectangle Packing Problem by enumeration in Section 2.3. Although theoretically not the fastest approach, we rely on the Spark algorithm (Funke 2011; Funke, Hougardy and Schneider 2016). It is a branch-and-bound approach enumerating all allowed spatial relations for any pair of rectangles. Thus the worst case running time is exponential in the number of relations that have to be determined. Despite that, Spark incorporates various bounding strategies making this

Algorithm 4.2: PackMacro

Input : Instance of the Macro Placement Problem, Macro m, already legalized macros L, packing mode pm, current positions l, target locations t.

Output: Legal locations for $L \cup \{m\}$ or failure.

```
// Model as rectangle packing instance
U := SelectUnconstrained(m, L, pm, l)
w := SelectWindow(U, L, pm, l)
Let C be the shapes of b ∈ B and m' ∈ L \ U intersecting w acc. to l
F := SelectFeasibleArea(U, C, l, w)
S := SelectRelations(U, C, pm, l)
// Solve rectangle packing instance
Find rectangle packing p for (U ∪ C, F, S, t)
Round p on-grid towards lower left
if any of the above steps failed then
    return failure
return Project(U, C, p)
```

algorithm efficient in practice. Moreover it intrinsically provides the flexibility to consider restricted spatial relation options for pairs of rectangle. This is an advantage we make use of in the following.

In order to keep the running time of PackMacro under control, we must not include all macros into the Rectangle Packing Problem instance and thus restrict the instance complexity. This can be done by restricting the overall number of rectangles included in the instance. Former versions of BonnMacro pursued this approach only (Brenner 2007). To achieve a reasonable running time, Rectangle Packing Problem instances were limited to very few movable rectangles and blockages. Due to this limitation this approach has two major disadvantages:

1. The Rectangle Packing Problem instances have to be chosen so small that larger groups of macros can never be moved at all. Consequently overall solutions frequently induce unnecessary movement although local instances are solved to optimality.
2. As more and more macros are already legalized, it becomes difficult to select a local instance subject to the restrictive instance size constraints. If no such instance can be found at all, BonnMacro had to fall back to moving m to the closest legal position. While this fallback often helps finding any completely legal placement, it often induces movement that could have easily been avoided.

To circumvent both drawbacks, BonnMacro additionally controls the Rectangle Packing Problem instance complexity directly. We allow much more rectangles but explicitly restrict the allowed relations of most of the rectangle pairs (Michaelis 2015).

More precisely, PackMacro maintains two different sets of rectangles: **unconstrained rectangles U** and **constrained rectangles C**. We impose no restrictions on the allowed relations for any pair of rectangles with at least one element of U. For a

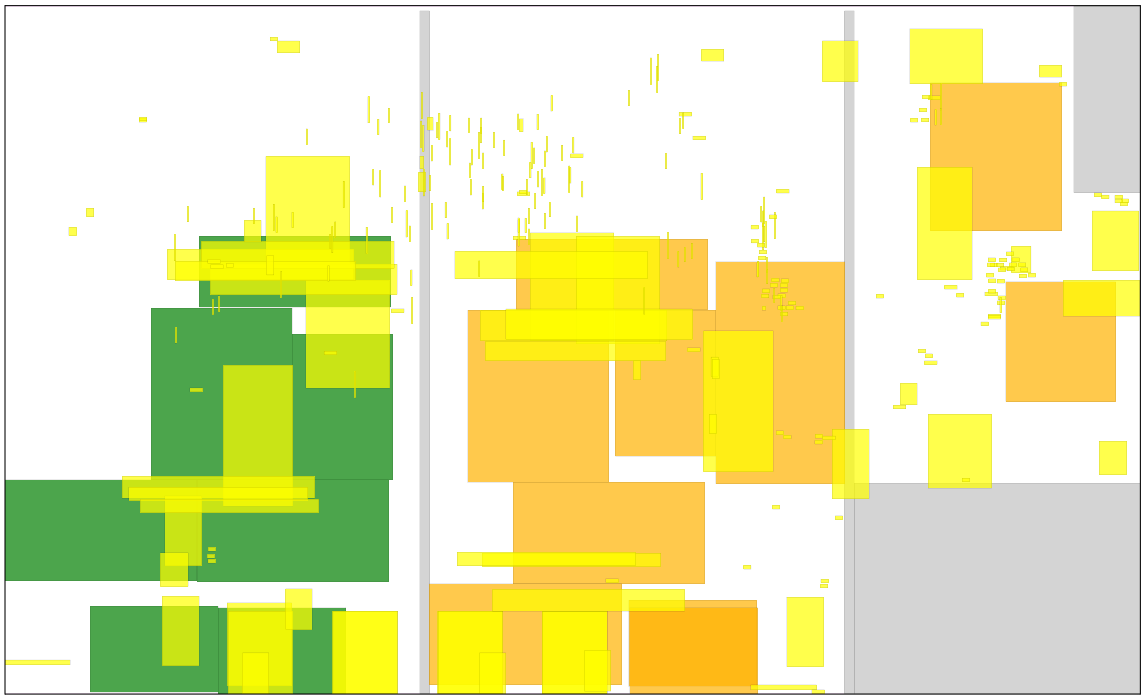


Figure 4.4: Instance of PackMacro during legalization on Lisa: $U \cup C$ in orange, $L \setminus (U \cup C)$ in green, unlegalized macros in yellow and blockages in gray.

pair of constrained rectangles though, only a single relation will be allowed. Consequently Spark can avoid any branching on such pairs whatsoever.

In the example instance presented in Figure 4.4, two orange shapes overlap at the bottom. One of them is m which is about to be legalized. In this example, the four orange rectangles closest to the lower chip boundary are U, i.e. unconstrained. Other orange rectangles with increasing distance to m are constrained.

Thereby we can consider instances with much more rectangles but a similar number of (pairwise) spatial relations subject to optimization. This strategy avoids both previously mentioned drawbacks of the former BonnMacro approach: It provides more flexibility for selecting a local Rectangle Packing Problem instance which therefore succeeds more often. Moreover constrained macros are not fixed in place but rather only limited in the ability to be moved. Thus we often can reduce the overall movement by moving constrained macros slightly.

Now that we have outlined the overall idea of PackMacro, we go through all individual steps and elaborate the details.

4.2.2 Choosing Unconstrained Rectangles

The initial step of PackMacro is the selection of unconstrained macros U by SelectUnconstrained (Algorithm 4.2, line 1). SelectUnconstrained always selects m, i.e. the macro m that is subject to legalization.

Subsequently we determine the number of additional unconstrained rectangles u_{pm} that should be considered. This number $u_{pm} \in \mathbb{N}$ is a constant depending on the current packing mode pm. In practice $u_{pm} = 2$ for almost all packing modes, but there are rare exceptions.

Further unconstrained macros are then selected from L based on proximity to m with

respect to current locations l. More precisely we choose $U' \subseteq L$ as the u_{pm} macros with smallest $c(m') \in \mathbb{R}$ where

$$c(m') := \left\| l(m), l\left(m'\right) \right\|_1 \cdot \sqrt[4]{\frac{\max\{w, w'\}}{\min\{w, w'\}} + \frac{\max\{h, h'\}}{\min\{h, h'\}}}$$

and (w, h) as well as (w', h') denote widths and heights of m and m'. By this choice we primarily select macros based on the l_1-distance to m with respect to l. In addition we favor macros m' with shape similar to m. The precise formula was determined by Schneider (2009) based on experimental results.

Finally SelectUnconstrained returns the $u_{pm} + 1$ unconstrained macros selected as $U := \{m\} \cup U'$.

4.2.3 Selecting the Window

Based on already chosen unconstrained macros U, PackMacro selects a window w using SelectWindow (Algorithm 4.2, line 2).

This window w serves as boundary for the Rectangle Packing Problem instance. In particular, it predetermines the subsequent selection of constrained rectangles C (Algorithm 4.2, line 3). Thus selecting an appropriate window w is the most important step in the creation of the Rectangle Packing Problem instance.

Note that blockages fit very well in our framework of constrained rectangles with limited movability. The additional constraint that blockages must not be moved at all can be modeled by feasible areas which are precisely the shape of the blockage.

In order to control the overall complexity of the created Rectangle Packing Problem instance, we must choose the window w in such a way that C has small cardinality. Therefore we impose the upper bound $c_{pm} \in \mathbb{N}$ on this cardinality. The upper bound c_{pm} depends on the packing mode pm and is chosen as $c_{pm} \in [10, 18]$ in practice.

Selecting a needlessly small windows on the other hand leaves little or even no room at all for packing U. Therefore we naturally are especially interested in windows w containing much unused whitespace. This leads to the following purely geometric optimization problem:

Bounded Window Selection Problem

Instance: Two rectangles $q \subseteq Q \subseteq \mathbb{R}^2$, disjoint rectangles R with $r \subseteq Q \setminus q$ for all $r \in R$, a bound $k \in \mathbb{N}$ and a non-negative density function $s \colon Q \to \mathbb{R}_{\geq 0}$.

Task: Determine a window w maximizing

$$\int_w s(p) \, \mathrm{d}p$$

subject to $q \subseteq w \subseteq Q$ and $\left|\{r \in R : r \cap w \neq \varnothing\}\right| \leq k$.

Note that the Bounded Window Selection Problem is well defined: On the one hand, q always is a feasible solution. On the other hand, by additivity of the integral and non-negativity of s there is an optimum w determined by borders of $r \in R \cup \{Q\}$ in all four cardinal directions. Since there are only finitely many of such rectangles, an optimum window w maximizing $\int_w s(p) \, \mathrm{d}p$ exists. Recall that our notion of intersection disregards one-dimensional areas according to Definition 2.4.

We can specify coordinates of potential optima even more precisely. The left boundary of w for example may be selected from

$$\{b : r = [a, b] \times [c, d] \in R, r \lhd q\} \cup \{a : Q = [a, b] \times [c, d]\}.$$

Extending this definition to all cardinal directions leads to a restricted Hanan-Grid G.

A reasonable approach for solving the Bounded Window Selection Problem without further assumptions on the density function s is enumeration. For this purpose we consider all $\mathcal{O}(|R|^2)$ potential combinations of left and right borders of w in G. For such a fixed x-interval $I \subseteq \mathbb{R}$, it is sufficient to consider all (inclusion-wise) maximal windows i.e. at most one window for each potential upper y-coordinate of w.

All these windows can be enumerated in $\mathcal{O}(|R|\varphi)$ by a double-sweep-line approach using separate sweep-lines for the upper and lower border of w. Hereby φ denotes the running time for deciding whether a candidate window is feasible after having moved either of the sweep-lines.

Wochnik (2017) showed $\varphi \in \mathcal{O}(1)$: This can be achieved by precomputing for each tile $t \in G$ how many $r \in R$ intersect t in the interior, the boundaries and the corners. We can realize this pre-processing in $\mathcal{O}(|R|^3)$ running time. Subsequently we can determine feasibility after moving one of the sweep-lines in $\varphi \in \mathcal{O}(1)$ time. This is possible by the inclusion-exclusion principle based on the precomputed tile information.

So far, we did not consider the objective at all. But it can easily be incorporated in the enumeration scheme: We precompute $\int_t s(p)\,\mathrm{d}p$ for any $t \in G$. When either of the sweep-lines moves, we update the current objective based on precomputed tile information by additivity of the integral in $\mathcal{O}(1)$ running time.

Overall this results in the following complexity of the Bounded Window Selection Problem:

Theorem 4.2 (Wochnik 2017)**.** *The* Bounded Window Selection Problem *can be solved in* $\mathcal{O}\left(\rho^3 + \rho^2\sigma\right)$ *running time where* $\rho := |R|$ *and* σ *denotes the maximum running time for evaluating* $\int_t s(p)\,\mathrm{d}p$ *on any tile* $t \in G$*.*

In the following we explain how we apply Theorem 4.2 for SelectWindow in practice. The choice of $k := c_{pm}$ has already been discussed before. We further choose $q \subseteq \mathbb{R}^2$ as the bounding box of U according the the current locations l.

Further we do not directly apply Theorem 4.2 to all shapes of $L \cup B$. This is done in order to avoid the cubic running time on instances with thousands of macros and blockages. Instead we first extend q in all cardinal directions separately until more than k shapes are intersected. This defines an a priori boundary Q for any optimum window. Then we only need to consider rectangles R corresponding to $L \cup B$ with shapes contained in Q.

Our density function s is zero on blockages. For points p not contained in a blockage, we use

$$s(p) := \frac{1}{\frac{\|p,q\|_1}{C} + 1} = \frac{C}{\|p, q\|_1 + C},$$

where $C := \max\{1, \mathrm{BB}(q)\}$. Thereby the density in q is 1 and converges to 0 with increasing distance from q. We thereby reflect the intuition that free space in close proximity to q is more valuable than in further distance. This is reasonable as we need additional free space for m – preferably close to q in order to minimize movement.

This type of density function s can be integrated in $\mathcal{O}(1)$ on each tile $t \in G$. Thus we can find an optimum solution to the Bounded Window Selection Problem in $\mathcal{O}\left(|R|^3\right)$ by Theorem 4.2 as $\sigma \in \mathcal{O}(1)$.

4.2.4 Computing Feasible Areas

For determined constrained and unconstrained rectangles C and U the next step of PackMacro is the selection of feasible areas F using SelectFeasibleArea (Algorithm 4.2, line 4).

The easiest case are blockages which all have predetermined fixed positions. As already outlined in Section 4.2.3, we model fixed positions by feasible areas that can only be satisfied by a unique position. More precisely we use the rectangle $b \subseteq \mathbb{R}^2$ itself as feasible area for $b \in B$.

Next we consider rectangles $r \in C \uplus U$ corresponding to (movable) macros that are contained in the window, i.e. $r \subseteq w$. Recall that this not necessarily includes all rectangles but particularly applies to $r \in U$ (cf. Algorithm 4.2, line 3). In this case we use the feasible area $m_r \cap w$ for r where m_r denotes the movebound of the macro corresponding to r.

Otherwise $r \in C$ corresponds to a (movable) macro but $r \not\subseteq w$. Here we proceed slightly differently. In particular we need to ensure that r is placed legally to already legalized shapes outside of w. Recall that these will not be modeled in our Rectangle Packing Problem instance explicitly (Algorithm 4.2, line 3).

Let $\overline{B}$ be this set of legalized shapes outside of w. More precisely $\overline{B}$ contains all shapes of already legalized macros in $L \setminus C$ as well as blockages $\{ b \in B : b \cap w = \varnothing \}$. We extend each border of r which intersects w until either the corresponding border of w or some $b \in \overline{B}$ is reached. Denote the resulting rectangle by $f \subseteq \mathbb{R}^2$. We use $f \cap m_r$ as feasible area for r, where m_r again denotes the movebound of the macro corresponding to r.

As r corresponds to a legally placed macro, r in particular is disjoint from $\overline{B}$. By construction the same holds for f and thus any placement of r in f is automatically disjoint from $\overline{B}$. Since $r \subseteq f$, the chosen feasible area for r is satisfiable.

4.2.5 Choosing Spatial Relations

As next step PackMacro determines restricted spatial relations and required minimum spacings using SelectRelations (Algorithm 4.2, line 5).

Considering minimum spacing requirements on rectangles was introduced for modeling grid constraints on feasible macro locations. Respecting grid constraints directly as part of the Rectangle Packing Problem is notoriously difficult:

Theorem 4.3 (Schneider 2009, Theorem 8)**.** *The* Rectangle Packing Problem *with additional grid constraints is NP-hard, even if an optimum solution to the corresponding problem without grids is known.*

Due to Theorem 4.3, we pursue a different approach. We impose certain minimum distance requirements and solve the Rectangle Packing Problem subject to these additional constraints. Thereby our choice of minimum spacings ensures that we can round any feasible solution to the respective macro grids without introducing overlaps.

The central idea is the following: Consider two rectangles r_i for $i \in [\![2]\!]$ and suppose $r_0 \sim r_1$ for $\sim \in \{ \lhd, \rhd, \triangledown, \vartriangle \}$ in any feasible solution. Without loss of generality $\sim \in \{ \lhd, \vartriangle \}$ as we can swap r_i otherwise. We select a minimum distance requirement $\psi(r_0 \sim r_1) \in \mathbb{R}$. Our choice thereby ensures that we can round r_1 on-grid towards r_0 whenever r_0 and r_1 are at least $\psi(r_0 \sim r_1)$ apart in the respective dimension corresponding to $\sim$. We can compute $\psi(r_0 \sim r_1)$ in $\mathcal{O}(1)$ by calculations presented by Engels (2013).

This can also be viewed differently: Any rounding onto a grid will be done towards the lower left (cf. Algorithm 4.2, line 7). The spacing ψ serves as artificial halo towards the

upper right of r_0 that is guaranteed to contain an on-grid position for the lower left corner of r_1. But in contrast to the version presented by Engels (2013), we avoid introducing this halo explicitly. In particular, this allows different halos for r_0 in e.g. (r_0, r_1) and (r_0, r_2) and thus is less restrictive.

To complete this modelling of grid constraints, we need to ensure that this rounding never will move a rectangle outside of the respective feasible area. We do so by rounding any feasible area as defined in Section 4.2.4 on-grid towards the upper right in advance.

Note that the overall approach to model grid constraints clearly still is pessimistic. There even are instances that have no solution obeying the minimum spacing constraints while feasible on-grid solutions exist. On the other hand, minimum spacing requirements are often exact as large groups of macros have a shared grid in practice.

Precomputing all minimum spacing requirements can be done in $\mathcal{O}(\rho^2)$ where $\rho := |U \dot\cup C|$. Recall that selecting U and C takes $\mathcal{O}\left(\rho^3\right)$ running time by Theorem 4.2 (cf. Section 4.2.3). Consequently the running time for minimum spacing computations is acceptable although we will now restrict the allowed relations and thus not explicitly use all of these values.

In addition to minimum spacings SelectRelations also chooses allowed relations: For a pair $(r_0, r_1) \in U \times (U \dot\cup C)$ with an unconstrained rectangle r_0, SelectRelations allows all relations. Otherwise $(r_0, r_1) \in C^2$ for which only a single relation will be allowed. This relation $\sim \in \{\lhd, \rhd, \triangledown, \triangle\}$ is determined as a spatial relation with maximum gap beyond the required spacing $\psi(r_0 \sim r_1)$ according to the current positions l.

Note that although l is legal on C, a relation with non-negative gap not necessarily exists as minimum spacings are pessimistic. We already mentioned that this occurs rarely in practice. This effect is further reduced by the post-processing Project elaborated in Section 4.2.7.

This concludes the description of all constraints in our legalization model as Rectangle Packing instance (Algorithm 4.2, lines 1 to 5).

4.2.6 Solving the Rectangle Packing Problem

As next step PackMacro optimizes over the afore-built Rectangle Packing instance $(U \dot\cup C, F, S)$ (Algorithm 4.2, line 6).

This is done using the Spark algorithm (Funke 2011; Funke, Hougardy and Schneider 2016). More precisely we use use an extension of Spark that is capable of obeying our restricted relations (Michaelis 2015).

Spark only considers linear netlength as objective, but we prefer minimizing area-weighted quadratic movement with respect to target locations t. Thus we approximate this quadratic function by a sum of piecewise linear, convex, even functions. We do this by few nets (around 10 in practice) with artificial pins in quadratically increasing distance to t.

Spark is a branch-and-bound algorithm that explores all combinations of allowed spatial relations for $C \dot\cup U$. Although these are exponentially many, Spark usually performs well in practice by using effective bounding strategies. Nonetheless at times certain instances can consume an undesired amount of running time. Therefore we impose a limit on the maximum number of expanded branch-and-bound nodes as deterministic limit for the running time. This limit is decreasing for macros with lower priority (cf. Algorithm 4.1, line 2).

In case our Rectangle Packing model has no solution, we are unable to legalize m (Algorithm 4.2, line 9). Otherwise we can round the obtained positions p on-grid towards

the lower left without introducing overlaps (line 7). Recall that this is due to our choice of minimum spacings and spatial relations which has been discussed in Section 4.2.5.

At this point p is a satisfactory solution, i.e. it extends l to be a legal solution of $L \cup \{ m \}$. PackMacro preserves this property for p, but finally applies a post-optimization which we discuss next.

4.2.7 Local Post-Optimization

The final step of PackMacro is the post-optimization Project (Algorithm 4.2, line 10). Denote the macros corresponding to $U \cup C$ by $M' \subseteq L \cup \{ m \}$. Project greedily optimizes movement for each macro $m' \in M'$ while keeping positions of all other macros $L \cup \{ m \} \setminus \{ m' \}$ fixed.

More precisely, we treat all macros $L \cup \{ m \} \setminus \{ m' \}$ as additional blockages. Under these constraints we compute an on-grid position for m' with minimum movement. Thereby we again (cf. Section 4.2.6) approximate the area-weighted quadratic movement objective by the same piecewise-*linear*, convex function.

Recall that our model of grid constraints by means of minimum spacings contains intrinsic pessimism (cf. Section 4.2.5). Therefore even though Project optimizes the same objective, it oftentimes can reduce movement as grid constraints are modeled exactly here.

Moreover we additionally allow flipping m' under a heuristically chosen objective penalty. BonnPlace supports orientation-specific grid constraints due to which some instances allow particularly small movement when m' is flipped into an appropriate flip-code. Incorporating this directly allows reducing overall movement even more effectively.

Project is a purely geometric algorithm running in two stages. Denote the fixed shapes corresponding to $L \cup \{ m \} \setminus \{ m' \}$ and B by B'. We first compute a set A of rectangles covering precisely all positions in which m' can be placed disjointly from B'. In the second stage we project the movement target $t(m')$ into each candidate rectangle $a \in A$ and round this off-grid projection in all directions onto the grid of m'. As we consider movement as piecewise-linear, convex function, the optimum on-grid position for m' thereby is considered as one candidate. Note that this optimization for a given rectangle $a \in A$ gave rise to the name Project.

The crucial part of Project is the first stage. Feasible locations for m' are the complement of appropriately extended blockages B', which can be computed by sweep-line algorithms. The intricate part is covering this rectilinear polygon with few rectangles A such that the subsequent stage finishes quickly. Note that finding a minimum cardinality cover for simple polygons is NP-hard (Culberson and Reckhow 1988) and MaxSNP-hard for general rectilinear polygons (Berman and DasGupta 1992). We therefore apply the algorithm of Franzblau (1989) finding in $\mathcal{O}(|B| \log |B|)$ running time a $\mathcal{O}(\log \alpha)$-approximation of the minimum cover with cardinality α of the general rectilinear polygon A.

In practice we avoid considering all B' to make Project even faster (Wochnik 2017): We first search with an exponentially expanding window w' around $t(m')$ for any feasible position p for m' that is contained in w' and disjoint from B'. This p is not necessarily an optimum solution and w' might still be very large. But it allows using Project on the restriction $w \cap B'$ where w is the rectangle

$$w := \left\{ x \in \mathbb{R}^2 : \left\| x - t\left(m'\right) \right\|_\infty \leq \left\| t\left(m'\right) - p \right\|_1 \right\}.$$

Note that using the l_1-ball instead of w would be geometrically smaller, but computationally more difficult to handle. Using this pre-processing makes PROJECT very efficient in practice. This is why we can utilize PROJECT frequently as part of any call to PACKMACRO.

4.3 Timing-Driven Macro Legalization

This section is devoted to extending BONNMACRO legalization to become timing aware. Note that even for fixed placement of macros, finding a placement with good timing properties for all other cells is a complicated and challenging problem in itself (cf. Section 5.4). Therefore it is not practical to evaluate timing exactly on any (partial) macro placement as part of the BONNMACRO legalization.

Instead we use the model of distance bounds presented in Section 3.1. We will optimize for a maximum distance bound slack in this model.

More precisely we start with canonical distance bounds G_b of Definition 3.7. Recall that in addition to macros, further shapes of cells are considered in $\mathrm{V}(G_\mathrm{b})$. The slack with respect to G_b is an upper bound for the actual slack (Proposition 3.8). Furthermore under practical assumptions (Assumption 3.6) the model is chosen to be particularly accurate for overall critical paths.

Since the TIMING-DRIVEN RECTANGLE PACKING PROBLEM is strongly NP-hard (Theorem 3.16), we can not hope for solving this problem globally optimally. Instead we proceed similarly as for the (regular, timing unaware) legalization of macros presented in Section 4.2.

We legalize one macro after another. Thereby we optimize for a maximum distance bound slack while considering neighborhoods of the macros both geometrically on the chip area and in G_b. The resulting instances of the TIMING-DRIVEN RECTANGLE PACKING PROBLEM can be solved optimally for sufficiently restricted neighborhoods using a mixed-integer programming formulation.

As before, we try various choices of sub-instances with different packing modes in MACROLEGALIZATION. Among all solutions extending legality to the next macro, we adapt the choice of BESTSOLUTION to consider distance bounds: We select the solution with maximum slack with respect to G_b and only consider the former objective movement as tie-breaker.

Now we consider all new steps that have been newly introduced in this approach of timing-driven macro legalization. This includes construction of distance bounds G_b in Section 4.3.1, modelling local instances of TIMING-DRIVEN RECTANGLE PACKING PROBLEM in Section 4.3.2 as well as details how to solve this model presented in Section 4.3.3.

4.3.1 Distance Bound Construction

In an initial pre-processing to the timing-driven MACROLEGALIZATION, we once construct distance bounds G_b. These bounds G_b remain constant for the remainder of the algorithm. All further optimizations of the placement target a best possible slack with respect to these bounds.

Consider a fixed instance of the MACRO PLACEMENT PROBLEM on macros M. The construction of said distance bounds starts with computing all circuit delay estimates $\mathrm{d_c}(s,t)$ for any timing startpoint s and endpoint t using Proposition 3.15. Based on $\mathrm{d_c}$, we can construct the canonical distance bounds G_b according to Definition 3.7.

By Assumption 3.6, G_b encodes timing information for any path in the timing graph that starts or ends in a macro. In particular, without loss of generality $\mathrm{V}(G_\mathrm{b})$ contains

rectangles for all macros $m \in M$. But in practice $\mathrm{V}(G_\mathrm{b})$ consists of many other rectangles. Most prominently among these extra objects are latches, which all intrinsically are timing start- and endpoints as elaborated in Section 2.2.2. But depending on the timing graph there can be arbitrary additional other rectangles. Especially due to latches, G_b has thousands of vertices in practice.

In addition to that, canonical distance bounds represent any timing assertions of the timing graph. This representation can not be handled directly due to the mere quantity of this information. Moreover assertions are generated based on assumptions with varying degree of certainty and additionally can be overridden by designers. Consequently such bounds range from being very loose and irrelevant to being overly ambitious and unsatisfiable.

This is why it is important to infer a more efficient representation of G_b for our purposes. We thereby ensure to capture the worst distance bound slack of any (yet to be determined) macro placement. Note that other timing metrics (e.g. FOM) are not necessarily preserved.

The first of these steps is a pruning of irrelevant distance bounds. We start with an upper bound $u \in \mathbb{R}$ on the best possible slack. It is derived from individual bounds assuming its endpoints are placed together as close as possible. Then we prune all bounds $e \in \mathrm{E}(G_\mathrm{b})$ that will have slack better than u for any placement whatsoever. Thereby we consider placing endpoints of e in the chip area as far apart as possible. If e has better slack than u in this worst case placement, we clearly need not consider e for worst slack optimization any longer.

In order to derive a more efficient representation of G_b, we eliminate induced parallel edges of G_b. This pruning removes dominated distance bounds based on the triangle inequality irrespective of the orientation of the endpoints of the bound. Despite lacking theoretical guarantee, this pruning strategy leaves at most 2 bounds for the majority of node pairs in practice.

Still for thousands of nodes in G on real-world instances, this is a huge graph. But by this pruning strategy we usually reduce the number of edges to a few percent of the canonical input which we can manage.

Note that we can not apply Corollary 3.42 at this point. Mentioned algorithm requires fixed macro orientations for the used argument of translating distance bounds. We optimize flip-codes not in Timing-Driven Rectangle Packing instances, but rather as post-optimization during PackMacro. In order to do so, we thus can not prune bounds here with the approach of Corollary 3.42.

4.3.2 Timing-Driven Macro Packing

In this section we elaborate the extensions necessary to optimize distance bound slack using PackMacro (cf. Algorithm 4.2, page 53). We will optimize the distance bound slack while legalizing m in a small neighborhood.

For this purpose we add a new step to the modelling stage of PackMacro (Algorithm 4.2, lines 1 to 5). Once U, w, C, F and S all have already been determined, we finally select further rectangles $N \subseteq \mathrm{V}(G)$.

We include these rectangles N as addition to $U \cup C$ into the Timing-Driven Rectangle Packing Problem instance. All other rectangles in $\mathrm{V}(G) \setminus N$ are fixed in their former positions. But in contrast to $U \cup C$, there will be no spatial relations imposed on any rectangle in N.

This is motivated as follows: We intentionally allow overlaps with $r \in N$ corresponding to a non-macro cell. For $r \in N$ corresponding to a macro, disjointness to $U \cup C$ is already implied by our choice of feasible areas (cf. Section 4.2.4, page 57). Thus we only fix the position of such rectangles and can avoid introducing any spatial relation choices at all.

Recall that the task of the MacroLegalization is finding legal positions for macros M. Determining legal positions for other non-macro cells, in particular those corresponding to N, is subject to later steps of BonnPlace (cf. Chapter 5). Thus it is appropriate to defer legalizing non-macros to these steps.

We include N in our Timing-Driven Rectangle Packing Problem model nevertheless in order to find better solutions in practice. If cells important for the overall placement (i.e. macros) are moved, we need to consider the flexibility of moving other incident cells. This allows in particular to react to new macro positions in order to maximize the worst slack overall. We emphasize that moving non-macros is essential for our objective. Otherwise the model of distance bounds only is a very inaccurate estimate for timing properties of legal placements of the complete netlist. Thus we allow moving $C \cup U \cup N$.

Thereby N is chosen to be close to $U \cup C$ in the distance bound graph G. We stress that this is not influenced by the current positions l, but purely depends on distance bounds G_b. The selection of N is done by a BFS starting from $U \cup C$. We restrict the selection based on the number of induced bounds, the cardinality of N as well as the maximum distance from $U \cup C$ in G. The actual numbers are empirically chosen around 5000 depending on the used packing mode *pm*.

We then optimize worst slack of distance bounds incident to $U \cup C \cup N$. For this purpose we assume all other rectangles to be fixed at their current positions, i.e. we formally consider the distance bound graph $H := G/\overline{R}$ resulting from contracting $\overline{R}$ in G where $R := (U \cup C \cup N)$ and $\overline{X} := \mathrm{V}(G) \setminus X$. Note that particularly $\square \in \overline{R}$. By H_b we denote the distance bound graph for which offsets of contracted endpoints are adapted based on their currently (fixed) positions.

Compared to our previous model as Rectangle Packing Problem, optimizing worst slack over H_b is more complicated. In order to compensate for this, we usually reduce the number of constrained rectangles for the timing-driven PackMacro in practice.

4.3.3 Solving the Timing-Driven Rectangle Packing Problem

We now explain our model of the Timing-Driven Rectangle Packing Problem as mixed-integer program (MIP). We use the linear program 3.4 as a starting point and incorporate decision variables for spatial relations. Such a formulation is very obvious and has e.g. already been used by Sutanthavibul, Shragowitz and Rosen (1990) on very small instances and without timing constraints.

Recall that an instance $(R, \varphi, \psi, A, G_\mathrm{b})$ of the Timing-Driven Rectangle Packing specifies allowed relations $\varphi\colon R \times R \to \mathcal{P}(\{\lhd, \rhd, \triangledown, \vartriangle\})$. Note that φ is defined to be anti-symmetric. In order to simplify notation it thus is sufficient to consider relations $\lhd_z \in \{\lhd, \triangledown\}$ where $z \in \{\mathrm{x}, \mathrm{y}\}$ refers to the affected dimension.

The required spacing for $r \sim r'$ is denoted by $\psi(r \sim r') \in \mathbb{R}$. Based on this spacing we introduce the shorthand $\mathrm{len}(r \lhd_z r') := \mathrm{len}(r, z) + \psi(r \lhd_z r')$, where $\mathrm{len}(r, z)$ denotes the edge length of r in dimension $z \in \{\mathrm{x}, \mathrm{y}\}$. Moreover we abbreviate the minimum and maximum z coordinates of feasible areas $A(r)$ for $r \in R$ by $z_{\min}(r)$ and $z_{\max}(r)$ where $z \in \{\mathrm{x}, \mathrm{y}\}$.

As in program 3.4, we consider fractional variables z_r for rectangles positions ($r \in R$, $z \in \{\mathrm{x}, \mathrm{y}\}$) as well as z_e^+, z_e^- denoting the borders of distance bound e in dimension z. In addition to that we introduce binary variables $D\big(r \lhd_z r'\big) \in \{0,1\}$ for any allowed relation $\lhd_z \in \varphi(r,r')$. By $\overline{D}$ we abbreviate the negation of these binary variables, i.e. $\overline{D}(r \sim r') := 1 - D(r \sim r') \in \{0,1\}$.

In order to enable or disable certain inequalities based on the decision variables D we use the method of big M. For this purpose we consider a sufficiently large $M \in \mathbb{R}$, e.g.

$$M := -\max\Big\{z_{\min}(r) - z_{\max}(r) : r, r' \in R, z \in \{\mathrm{x}, \mathrm{y}\}\Big\}.$$

Equipped with this notation we can formulate the following MIP:

Timing-Driven Rectangle Packing Mixed-Integer Program

$$\begin{aligned}
\max \quad & s \\
\text{s.t.} \quad & z_r \geq z_{\min}(r) && (r \in R;\ z \in \{\mathrm{x}, \mathrm{y}\}) && \text{(4.1a)}\\
& z_r \leq z_{\max}(r) \\
& D\Big(r \sim r'\Big) \in \{0,1\} && (\sim \in \varphi(r,r')) && \text{(4.1b)}\\
& z_r - z_{r'} \leq M \cdot \overline{D}\Big(r \lhd_z r'\Big) - \mathrm{len}\Big(r \lhd_z r'\Big) && (\lhd_z \in \varphi(r,r')) && \text{(4.1c)}\\
& z_r - z_{r'} \geq M \cdot D\Big(r \lhd_z r'\Big) - \mathrm{len}\Big(r \lhd_z r'\Big) \\
& 1 \leq \sum_{\sim \,\in\, \varphi(r,\,r')} D\Big(r \sim r'\Big) && (r, r' \in R : \varphi(r,r') \neq \varnothing) && \text{(4.1d)}\\
& z_e^- \leq z_r + z\big(\mathrm{o}(r,e)\big) && (e \in \delta(r);\ z \in \{\mathrm{x}, \mathrm{y}\}) && \text{(4.1e)}\\
& z_e^+ \geq z_r + z\big(\mathrm{o}(r,e)\big) \\
& \mathrm{b}(e) \geq s + \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \Big(z_e^+ - z_e^-\Big) && (e \in \mathrm{E}(G)) && \text{(4.1f)}
\end{aligned}$$

Constraints 4.1a, 4.1e and 4.1f are carried over from the linear program 3.4. By constraints 4.1b we enforce binary decision variables one of which is required to be 1 for any pair of rectangles requiring disjointness by constraints 4.1d.

For enforcing spatial relations in constraints 4.1c, we can use semantic branching. This technique was first introduced by Armando, Castellini and Giunchiglia (1999–1999) and first used in the area of rectangle packing by Korf, Moffitt and Pollack (2010). It avoids redundant assignments to rectangle relations, e.g. for placing r right and above of r'.

For branch-and-bound algorithms semantic branching is a strict improvement. In a MIP formulation this is not a priori clear: On the one hand, it strengthens the LP relaxation, especially if decision variables already have been partially fixed. On the other hand, the negated constraints add extra complexity due to which solving the MIP can become harder and slower. Based on practical experiments we decided to use semantic branching: This provides a clear benefit for instances with many rectangles requiring disjointness and the slowdown on smaller instances is negligible.

Note that due to distance bounds b, that are not necessarily integer, we no longer can strengthen the inequality for unselected relations. This has been possible e.g. for rectangle packing under wirelength minimization (cf. Funke, Hougardy and Schneider 2016, Theorem 7).

Practical MIP solvers need to use floating point numbers. In order to make computations numerically more stable, we generally need to ensure input numbers to be in a reasonable interval. For this purpose our implementation uses differing values of big M for each pair of rectangles $\{r, r'\}$ for which $\varphi(r, r') \neq \varnothing$.

Note that we actually can avoid binary variables for pairs of rectangles $\{r, r'\}$ with $\varphi(r, r') = \{\sim\}$. For such pairs, there is only one relation allowed and thus no choice at all. This can simply be modeled by one constraint similar to 4.1c for $D(r \sim r') = 1$. Recall that this applies to any pair of constrained rectangles in our model. Our MIP solver is capable of deducing this optimization in a pre-processing step, which is why we for reasons of simplicity use program 4.1 in both theory and practice.

We will now work towards improving our MIP formulation. In order to compare different variants, we explicitly state the baseline implied by mixed-integer program 4.1:

Proposition 4.4. *The* Timing-Driven Rectangle Packing Problem *can be formulated as mixed-integer program with* $2n + 2m + 1$ *fractional variables,* s *binary variables and* $4n + 2s + j + 5m$ *constraints where* $n := |R|$, $m := |\mathrm{E}(G)|$, $s := \frac{1}{2}\sum_{r,r' \in R} |\varphi(r, r')|$ *and* $j := |\{\{r, r'\} : r, r' \in R, \varphi(r, r') \neq \varnothing\}|$.

Proof. Certified by the mixed-integer program 4.1. ■

As a first improvement, we can avoid the variables z_e^+ and z_e^- for computing the length of bounds $e \in \mathrm{E}(G)$ entirely. This can be achieved by computing the length of e by enumerating all four directions $d \in [\![4]\!]$, which has implicitly already been done in Definition 3.28 and Proposition 3.31.

Proposition 4.5. *The* Timing-Driven Rectangle Packing Problem *can be formulated as mixed-integer program with* $2n + 1$ *fractional variables,* s *binary variables and* $4n + 2s + j + 4m$ *constraints where* n, m, s *and* j *are defined as in Proposition 4.4.*

Proof. To shorten notation we use the abbreviation $z(r, e) := z_r + z\big(\mathrm{o}_e(r)\big)$. We consider the following constraints as replacement for constraints 4.1e and 4.1f:

$$\mathrm{b}(e) - s \geq \sum_{z \in \{\mathrm{x}, \mathrm{y}\}} \sigma_z^d \Big[z(r, e) - z(r', e) \Big] \qquad \left(e = \left\{ r, r' \right\} \in \mathrm{E}(G), d \in [\![4]\!]\right) \tag{4.2e}$$

Note that constraints 4.2e do not depend on the ordering of e as we consider all directions and thus all combinations of signs anyway. We further reference the mixed-integer program 4.1 with replacement constraints 4.2e as program 4.2.

The righthand side of constraints 4.2e computes the length of e in direction d. By considering all directions $d \in [\![4]\!]$, we thereby can correctly limit the worst slack variable s.

Replacing constraints 4.1e and 4.1f by constraints 4.2e in mixed-integer program 4.2 makes all variables z_e^+ and z_e^- for $e \in \mathrm{E}(G)$ obsolete. Thereby we reduce the number of fractional variables by $2m$ and the number of constraints by m. ■

But we can even do better than Proposition 4.5. Recall that we only consider local sub-instances of the Macro Placement Problem as Timing-Driven Rectangle Packing instances, which has been described in Section 4.3.1.

The overall distance bound graph on all rectangles contains parallel edges. For our local distance bound graph G in our Timing-Driven Rectangle Packing Problem

instance, even more parallel edges exist. These edges result from contracting $\square$ with rectangles not considered in these instances.

We define $m' \in \mathbb{N}$ as the number of edges in $\mathrm{E}(G)$ with distinct endpoints, i.e. formally

$$m' := \left| \left\{ \left\{ r, r' \right\} \subseteq R : \exists e \in \mathrm{E}(G) \text{ with } \Psi(e) = \left\{ r, r' \right\} \right\} \right|.$$

Note that in contrast to the ordinary number of edges m of G, m' accounts for parallel edges only once.

Remember that the Timing-Driven Rectangle Packing Problem considers fixed orientations of all rectangles. Thus we can use Corollary 3.42 and exploit the geometry of Manhattan arcs in order to handle parallel distance bounds more efficiently:

Proposition 4.6. *The* Timing-Driven Rectangle Packing Problem *can be formulated as mixed-integer program with* $2n + 1$ *fractional variables,* s *binary variables and* $4n + 2s + j + 4m'$ *constraints where* m' *denotes the number of non-parallel edges of* G *and* n, s *and* j *are defined as in Propositions 4.4 and 4.5.*

Proof. Consider the mixed-integer program 4.2 described in Proposition 4.5 as starting point. Let $\{ r_0, r_1 \} \in \mathrm{E}(G)$ and $E := \mathrm{E}(G[\{ r_0, r_1 \}])$ the set of bounds between r_0 and r_1. We will construct replacements for all constraints 4.2e for $e \in E$.

Using Corollary 3.42 we can equivalently represent all bounds E by two new bounds e_0, e_1. Moreover by the constructive proofs of Proposition 3.41 and Corollary 3.42, e_0 and e_1 are not arbitrary bounds. In fact we rather know for any placement l of r_i

$$\mathrm{slack}(l, E) = \mathrm{slack}\Big(l, \{ e_i : i \in [\![2]\!] \}\Big) = c - \Big\| \big[l(r_0) - l(r_1) \big], A \Big\|_1$$

where A is a Manhattan arc and $c \in \mathbb{R}$ is some constant. Moreover A contains $0 \in \mathbb{R}^2$ and is rotationally symmetric around 0.

We can compute $\| \cdot, A \|_1$ as maximum distance to points in specific directions. Denote the endpoints of A by $a'_i \in \mathbb{R}^2$ for $i \in [\![2]\!]$ with $\mathrm{x}(a'_0) \leq \mathrm{x}(a'_1)$. We further define $a_d \in \mathbb{R}^2$ for each direction $d \in [\![4]\!]$ as $a_d := a_0$ for $d \in \{ \mathrm{NW}, \mathrm{SW} \}$ and $a_d := a_1$ for the other directions. Consequently for any point $p \in \mathbb{R}^2$

$$\| p, A \|_1 = \max \left\{ \sum_{z \,\in\, \{ \mathrm{x}, \mathrm{y} \}} \sigma_z^d \cdot z(p - a_d) : d \in [\![4]\!] \right\}.$$

This can easily be verified by a case distinction whether $p \in Q_d$ for $d \in [\![4]\!]$ and $Q_d \subseteq \mathbb{R}^2$ as in the proof of Lemma 3.40 (page 40, visualized in Figures 3.8 and 3.9).

Using this representation of $\| \cdot, A \|_1$ we can can bound slack s for all bounds $\mathrm{E}(G)$ in our mixed-integer program by the following constraints

$$c - s \geq \sum_{z \,\in\, \{ \mathrm{x}, \mathrm{y} \}} \sigma_z^d \cdot \Big[(z_{r_0} - z_{r_1}) - z(a_d) \Big] \qquad (d \in [\![4]\!]). \tag{4.3e}$$

Note that since A is rotationally symmetric around 0, constraints 4.3e do not depend on the order of r_i. We refer to mixed-integer program 4.3 as the the mixed-integer program 4.2 in which constraints 4.2e have been replaced by constraints 4.3e for all sets of parallel bounds.

Replacing constraints 4.2e by constraints 4.3e introduces no extra variables and substitutes $4m$ constraints by $4m'$ constraints overall. Consequently mixed-integer program 4.3 is a formulation with the claimed complexity. ■

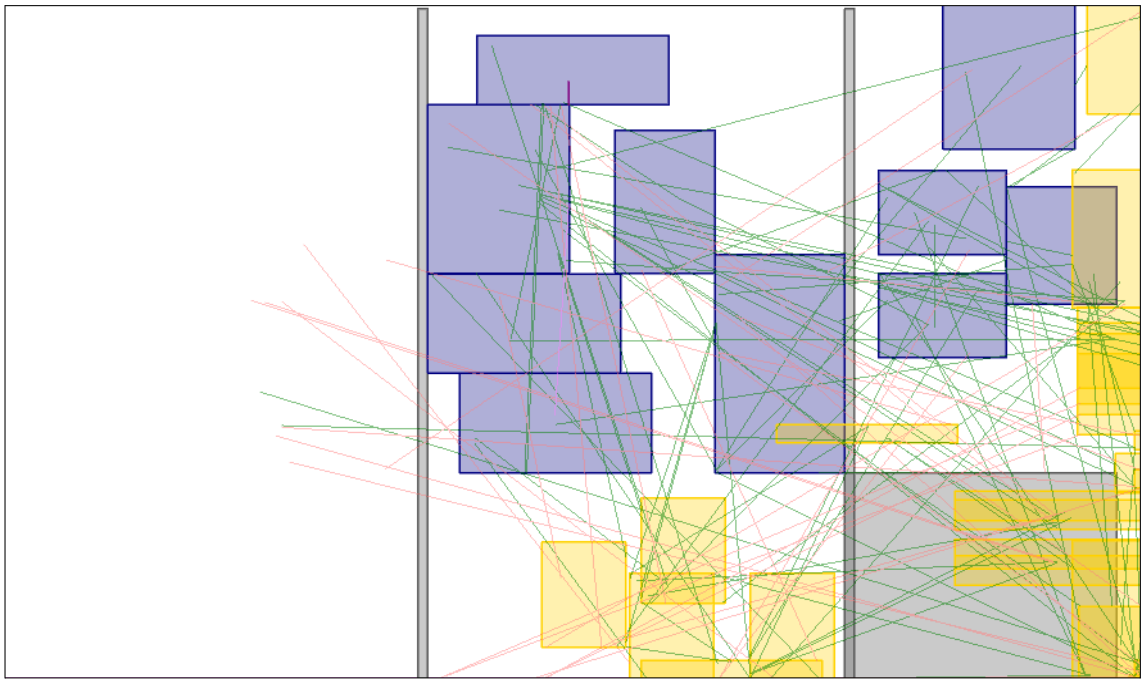


Figure 4.5: Worst slack optimum solution to the instance presented in Figure 4.4 on Lisa: Rectangles $U \dot{\cup} C$ are drawn in blue, $r \in N$ movable in orange and fixed in gray. Distance bounds are drawn as lines connecting respective endpoints colored by slack. The critical bound in purple is highlighted in the center. Less critical bounds are drawn in pink and green.

Note that $m' \leq \min\left\{ n^2, m \right\}$. Consequently Proposition 4.6 implies the first bound on the number of constraints which is independent from the number of (parallel) bounds.

Recall that the Manhattan arcs for Proposition 4.6 can be computed in linear time by Proposition 3.41. Consequently formulating program 4.3 requires $\mathcal{O}(m)$ pre-processing time only.

Mixed-Integer Programs Solved in Practice

Next we will elaborate how we make use of mixed-integer program 4.3 in our practical application. For this purpose consider Figure 4.5 for an illustration of an optimum solution to this program. It is of special interest that the solution apparently does not minimize the length of all bounds.

This is due to the fact our solver usually determines a basic solution to the MIP. In such a solution many constraints of the type of constraints 4.3e are tight. Consequently all corresponding bounds attain exactly the overall worst slack despite being clearly avoidable. But for the overall worst slack objective of program 4.3, this is irrelevant. Thus such solutions are very much expected.

But for our application in practice such solutions are undesirable and complicated to handle for two main reasons:

- Given such solutions, we are unable to tell which of the bounds actually cause the overall worst slack. Thus we have less understanding of the underlying timing model of distance bounds. Finally it makes development of this approach much harder since we can not validate the worst slack.

- As explained in Section 4.3.2, the BonnMacro legalization considers local packing instances only. Thus we depend on reliable positions for all rectangles, particularly those excluded from the local instance. With that regard worst slack optimum solutions are unsatisfactory.

We now explain how to handle both of these issues. For each we will change the MIP model and especially the objective in order to meet our practical requirements.

First we tackle identifying distance bounds that actually determine the overall worst slack. Let $r_i(j)$, $a_d(j)$ and c_j for $j \in [\![m']\!]$, $i \in [\![2]\!]$ and $d \in [\![4]\!]$ be the constants for which Proposition 4.6 introduced constraints 4.3e into mixed-integer program 4.3. We now introduce new fractional variables $0 \leq \iota_j \leq \varepsilon_I$ for $j \in [\![m']\!]$ and a small constant $\varepsilon_I \in \mathbb{R}$.

The idea is to use ι_j as artificial term to tighten constraints 4.3e. Advertedly tightening theses constraints is only possible for uncritical bounds without decreasing the overall worst slack s. On the other hand, we encourage MIP solutions to maximize ι_j by incorporating it into the objective. Thus we are able to identify uncritical bounds based on the value of ι_j.

Formally this results in the following mixed-integer program 4.4:

$$\begin{aligned}
\max \quad & \Big(1 + m'\Big) \cdot s + \sum_{j \in [\![m']\!]} \iota_j \\
\text{s.t.} \quad & \text{Constraints 4.1a to 4.1d (page 63)} \\
& 0 \leq \iota_j \leq \varepsilon_I && (j \in [\![m']\!]) && \text{(4.4e)} \\
& c_j - s - \iota_j \geq \sum_{z \,\in\, \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \Big[\Big(z_{r_0(j)} - z_{r_1(j)} \Big) - z\big(a_d(j)\big) \Big] \\
& && (d \in [\![4]\!], j \in [\![m']\!]) && \text{(4.4f)}
\end{aligned}$$

A solution of mixed-integer program 4.4 is visualized in Figure 4.5. Here the ι_j variables have been utilized to identify the overall critical distance bound in purple.

Note that program 4.4 somewhat lexicographically maximizes worst slack s and subsequently $\sum_{j \in [\![m']\!]} \iota_j$. Out choice of objective can not guarantee to actually find a solution precisely attaining the best possible slack s^* (as e.g. program 4.3 does). But still, any optimum solution to program 4.4 can only be worse than s^* by at most ε_I:

Lemma 4.7. *Let $(R, \varphi, \psi, A, G_{\mathrm{b}})$ be an instance of* Timing-Driven Rectangle Packing *with optimum slack s^*. Any solution $\Big(s, \iota_j\Big)$ to the corresponding mixed-integer program 4.4 satisfies $s \geq s^* - \varepsilon_I$.*

Proof. Assume $s < s^* - \varepsilon_I$. This implies

$$\Big(1 + m'\Big) \cdot s + \sum_{j \in [\![m']\!]} \iota_j < \Big(1 + m'\Big) \cdot (s^* - \varepsilon_I) + m' \cdot \varepsilon_I < m' \cdot s^*.$$

Note that any optimum solution to the Timing-Driven Rectangle Packing instance implies a solution of mixed-integer program 4.4 with $\iota'_j :\equiv 0$. Such a solution has objective value $m' \cdot s^*$. Thus $\Big(s, \iota_j\Big)$ is suboptimal which is a contradiction. ∎

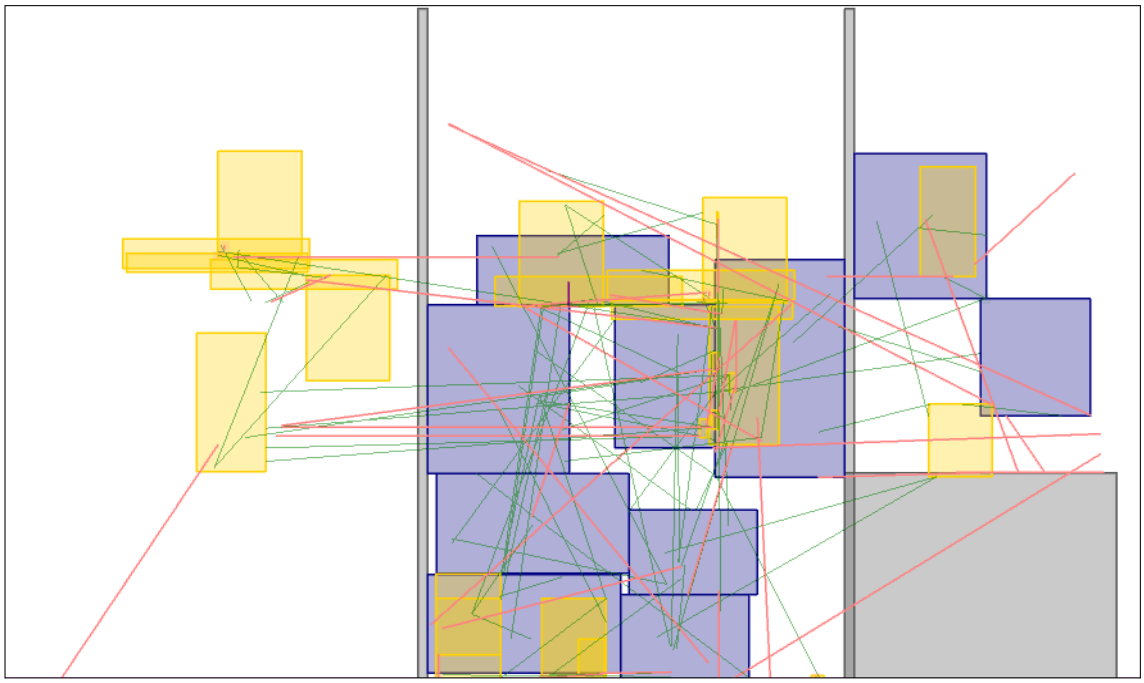


Figure 4.6: FOM optimum solution to the instance presented in Figure 4.5 on Lisa: The critical bound in purple is preserved. FOM defining bounds are highlighted in pink. Most of these have positive slack and thus are not shortest possible.

In practice ε_I can be chosen as a few nm. An alternative to our objective in program 4.4 would be solving two separate MIPs: One for finding the optimum worst slack s^* and one to determine ι_j subject to attaining worst slack s^* overall. But remember that we can only solve mixed-integer programs numerically and thus can only find an optimum solution up to a certain error anyway. Since ε_I can be chosen so small compared to s^*, such an approach hardly ever is more accurate.

Recall that we tried hard to keep the number of variables and constraints as small as possible (cf. Proposition 4.6). With that regard introducing variables ι_j is undesirable.

On the other hand, these variables have very limited effect on the overall structure of the MIP, in particular since ε_I is small. Practical experiments have demonstrated that solving mixed-integer program 4.4 is not harder than solving program 4.3 while at the same time solutions provide much more information. This is why we solve program 4.4 in practice.

Next we want to tackle the second problem illustrated by Figure 4.5. Therefore we need to determine reliable positions for all rectangles, particularly those irrelevant from a perspective of worst slack. We do this by another MIP subject to the constraint of preserving the overall optimum worst slack s^* determined by solving mixed-integer program 4.4.

For this purpose we optimize the Figure of Merit (FOM), i.e. the sum of negative slacks for all rectangles. All rectangle positions contribute to this objective. Most importantly this applies particularly to those rectangles which have been irrelevant for the worst slack s^*. This is why FOM is a reasonable secondary objective for our purpose.

To this end we introduce new variables f_r for $r \in R$ that will denote the worst slack of any bound incident to r. The FOM can be expressed as sum of these new variables f_r.

This results in the following mixed-integer program 4.5 :

$$\begin{aligned}
\max \quad & (1+m') \cdot \sum_{r \in R} f_r + \sum_{j \in [\![m']\!]} \iota_j \\
\text{s.t.} \quad & \text{Constraints 4.1a to 4.1d (page 63)} \\
& s^* \leq f_r \leq 0 && (r \in R) && \text{(4.5e)} \\
& c_j - f_r - \iota_j \geq \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \Big[\Big(z_{r_0(j)} - z_{r_1(j)} \Big) - z(a_d(j)) \Big] \\
& && (d \in [\![4]\!],\, j \in [\![m']\!],\, r \in \{ r_i(j) : i \in [\![2]\!] \}) && \text{(4.5f)}
\end{aligned}$$

By constraints 4.5e we ensure that no rectangle has slack worse than s^*, i.e. any solution is required to have optimum worst slack. Using constraints 4.5f we model the defining property of f_r, i.e. denoting the worst slack of any bound incident to r. Similarly to program 4.4 we honor ι_j as additional objective in program 4.5 for identify FOM-defining connections.

Note that from a timing point of view, all solutions with positive slack are somewhat identical. There is nothing more to achieve than making all signals arrive in time. Thus constraints 4.5e bounds f_r by 0 from above according to the definition of FOM (cf. Section 2.2.2, page 8). We also assume $s^* \leq 0$ or equivalently consider $\min\{ s^*, 0 \}$ as optimum worst slack.

Mixed-integer program 4.5 lexicographically maximizes overall worst slack, FOM and sum of ι_j. In analogy to Lemma 4.7, we thereby identify the optimum FOM F up to ε_I additively. By the same arguments as presented in the context of program 4.4, this error is negligible in practice.

Finally it is worth mentioning that constraints 4.5e frequently make mixed-integer program 4.5 infeasible in practice. Solving a MIP is a numerical procedure executed with errors caused by floating point arithmetic.

Consequently we can not expect to find the same optimum slack s^* when this value is expressed differently. To cope with this behavior, we marginally relax s^* for the purpose of modelling program 4.5.

A solution to constraints 4.5e is presented in Figure 4.6. In this example most distance bounds have positive slack, which is why the slack defining pink bounds are not necessarily shortest.

Optimizing FOM as tiebreaker helps finding reasonable positions for all rectangles. This applies particularly to rectangles that end up with at least one bound of negative slack. But there are also cases of totally timing uncritical rectangles for which this is not the case. Placing these according to a basic solution of program 4.5 can result in surprising placements that can not be justified by any objective. In addition we prefer these rectangles to be positioned closed to their input locations in order to minimize the perturbation of the input.

For this reason we introduce a third program 4.6 as additional tiebreaker. We thereby minimize the area-weighted (linear) movement to the target locations $t(r) \in \mathbb{R}^2$ for $r \in R$. Therefore we introduce new variables m_r for $r \in R$ describing the movement of r from $t(r)$

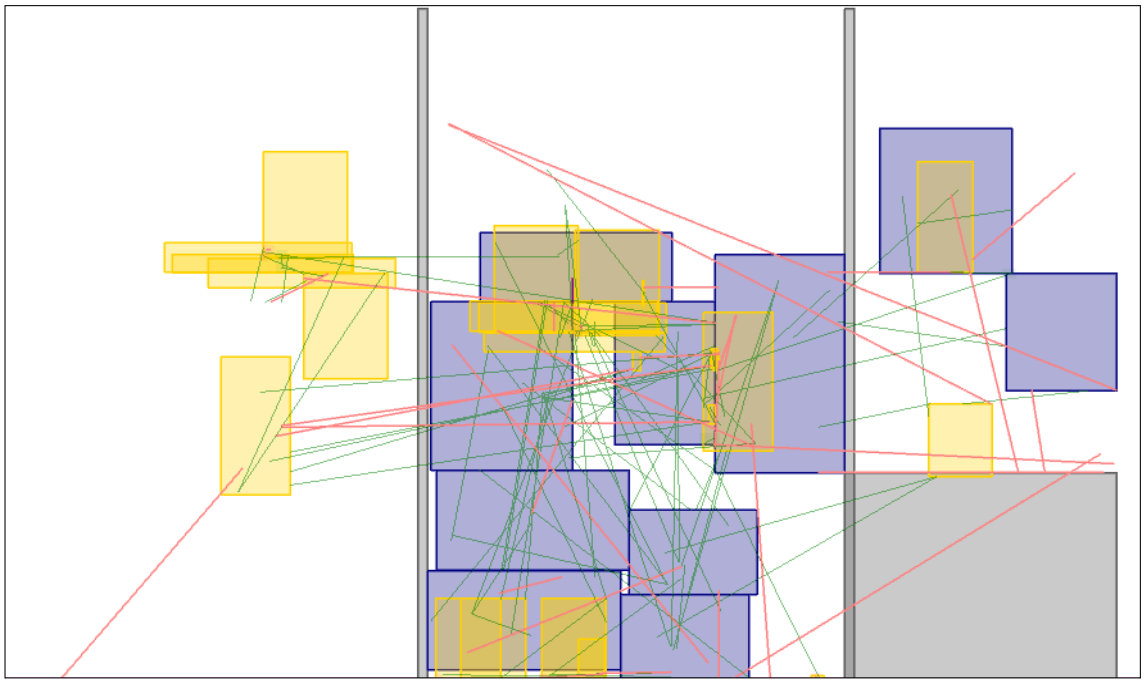


Figure 4.7: Movement optimum solution to the instance presented in Figure 4.6 on Lisa: The critical bound in purple is preserved. Most cells can be moved towards their input locations while preserving FOM.

to (x_r, y_r). By means of additional constraints, we ensure that the optimum worst slack s^* as well as the optimum FOM F are preserved. This leads to the following program 4.6:

$$\begin{aligned}
\min \quad & \sum_{r \in R} m_r \cdot w(r) \\
\text{s.t.} \quad & \text{Constraints 4.1a to 4.1d (page 63), 4.5e and 4.5f} \\
& F \leq \sum_{r \in R} f_r && \text{(4.6g)} \\
& m_r \geq \sum_{z \in \{\mathrm{x},\mathrm{y}\}} \sigma_z^d \cdot \Big[z_r - z\big(t(r)\big) \Big] && (d \in [\![4]\!],\ r \in R) \quad \text{(4.6h)}
\end{aligned}$$

Effectively program 4.6 is an extension of program 4.5 with a different objective. Instead of FOM we now minimize the movement weighted by the area $w(r)$ of the rectangles. By constraint 4.6g we ensure that the optimum FOM F is preserved. Constraints 4.6h measures the movement m_r of $r \in R$ from $t(r)$ to the current location of r. Thereby we use the same expression as has already been utilized for Proposition 4.5. This is why we need to introduce $|R|$ new variables only.

As already explained, we solely solve program 4.6 as tiebreaker. Thus we do so with fixed relations only. For this purpose we fix $D(r \sim r')$ to 1 if this has been the case in the solution of program 4.5. Recall that for each $r, r' \in R$ with disjointness requirements, there is such a relation $\sim$ by constraints 4.1d. Moreover we remove all other, non-true binary variables $D(r \sim r')$. Thus program 4.6 in fact becomes a linear program and can be solved more efficiently.

Similar as for solving programs 4.3 and 4.5, we carefully have to ensure numerical stability under floating point arithmetic. Therefore we again need to relax F slightly to ensure linear program 4.6 is feasible. Additionally we are not allowed to use $w(r) := \mathrm{len}(r, \mathrm{x}) \cdot \mathrm{len}(r, \mathrm{y})$ directly. We have to scale this weight into a small numerical range that can be handled reliably by our solver.

An illustration of positions determined by linear program 4.6 is given in Figure 4.7. We see the same timing-driven rectangle packing instance on Lisa that has already been considered in Figures 4.5 and 4.6. In this solution many cells can be moved towards their input locations without decreasing either worst slack or FOM.

Rounding MIP Solutions

BonnMacro operates on integers: All coordinates, widths and heights of any considered rectilinear shapes are managed as integers. Since all mixed-integer programs presented in Section 4.3.3 consider fractional variables z_r for rectangles $r \in R$ and are solved numerically, we need to round MIP solutions.

Recall that in the case of (non timing-driven) Rectangle Packing optimum integer solutions exist: If all rectangles, feasible areas and pins have integer coordinates, for fixed relations the corresponding dual min-cost flow problem has integral input and thus an integral optimum. Since this applies to spatial relations of any optimum solution in particular, it implies integral optimum solutions of Rectangle Packing in context of BonnMacro (Funke, Hougardy and Schneider 2016, Lemma 12).

But integral optima not necessarily exist for Timing-Driven Rectangle Packing – even with integral input: Consider a path of an even number of distance bounds with $\mathrm{b} :\equiv 0$ connecting fixed points with odd distance. On such an instance all optima are non-integral. Consequently finding an integral optimum is more difficult.

But on the bright side, fractional solutions can be rounded with marginal slack decrease:

Lemma 4.8. *Consider an instance $(R, \varphi, \psi, A, G_{\mathrm{b}})$ of* Timing-Driven Rectangle Packing *with integral coordinates. Let z_r for $r \in R$ and $z \in \{\mathrm{x}, \mathrm{y}\}$ be an optimum solution with slack s^*. Then for any $p \in [0, 1)$ the solution $z'_r := \lfloor z_r + p \rfloor$ is feasible and has slack at least $(s^* - 2)$.*

Proof. Consider the linear program Proposition 4.5 with fixed relations for which z_r and s^* are optimum. Since $\lfloor \cdot + p \rfloor \colon \mathbb{R} \to \mathbb{Z}$ is monotone and the identity on $\mathbb{N}$, z'_r satisfies all feasible area and disjointness constraints. Moreover $(z_r - z_{r'}) \leq \left(z'_r - z'_{r'}\right) + 1$ for any $z \in \{\mathrm{x}, \mathrm{y}\}$ and $r, r' \in R$. Consequently z' has slack at least $(s^* - 2)$. ∎

In practice we have to choose $p \in [0, 1)$ carefully in order to apply Lemma 4.8. Note that an actual MIP solution is determined numerically and thus only feasible up to a certain precision $\varepsilon > 0$. Thus for example $z_r := 0 < z_{r'} < z_r + \varepsilon$ satisfy the constraint $z_{r'} \leq z_r$ in the numerical sense. But rounding with $p := (1 - \delta)$ for $\delta \in (z_r, z_{r'})$ results in coordinates z' with $z'_{r'} > z'_r$ violating this constraint.

Nevertheless this issue can be avoided: Consider all n decimal places $d_i \in [0, 1)$ of coordinates z_r for $n := 2|R|$. Since $\varepsilon < \frac{1}{2n}$ in practice, we can choose $\delta \in [0, 1)$ s.t. for all $i \in [\![n]\!]$ either $d_i < \delta - \varepsilon$ or $d_i > \delta + \varepsilon$. Rounding with respect to $p := (1 - \delta)$ consequently yields an integral solution that also preserves feasibility numerically.

Finally we remark that the downside of losing 2 in terms of slack by Lemma 4.8 is minor in practice. This distance corresponds to less than 0.001 % cycle time on all our instances.

4.3.4 Timing-Driven Local Post-Optimization

Recall that finally once legal positions have been computed PackMacro applies the local post-optimization Project. Since the latter algorithm explicitly targets movement minimization, we skip this step for timing-aware packing modes pm.

Nevertheless it is worth mentioning that the following problem can be solved optimally with respect to distance bound slack in polynomial time: Find a legal, on-grid position for a single rectangle $r \in R$ while considering all other rectangles as fixed blockages. Thus all distance bounds can be considered as parallel to $\{r, \square\}$, which is why by Corollary 3.42 the set of optimum off-grid positions is a Manhattan arc A. By the same geometric decomposition used for Project (cf. Section 4.2.7) if suffices to consider the sub-problem of placing r into some rectangle $X \subseteq \mathbb{R}^2$ without blockages. An optimum on-grid position in X for r has to be closest to A and thereby can be determined by an integer program. Note that for this purpose, exactly two integral variables are required in order to model grid constraints for r. An integer program with exactly two variables can be solvable in polynomial time. This was first discovered by Scarf (1981a,b) and subsequently generalized to a constant number of integer variables by Lenstra (1983).

But this problem is relatively insignificant in practice and would mostly be useful as fallback if the instance creation fails for all other timing-driven packing modes. This is why it has not been implemented.

Nevertheless changing the orientation of a macro can often improve the overall worst slack. Although flip-code optimization is explicitly not considered as part of the Timing-Driven Rectangle Packing Problem, it could theoretically be incorporated in formulations as MIP. Such a MIP would require additional binary variables for all rectangles R in order to model flip-code choices. In addition the improvements of Proposition 4.6 no longer apply, since the necessary Corollary 3.42 requires fixed macro orientations. Consequently also many additional constraints are needed in order to model the overall worst slack s. This is why we expect such an extension by flip-code variables to be impractical and thus did not implement it.

Despite that, macro orientations are relevant still. Instead of optimizing locations and orientations together, we consider the easier problem of flip-code optimization for fixed locations. This we solve greedily for one macro at a time in order to maximize slack locally. Such an approach is very efficient and thus can be used frequently as post-optimization of any MIP solution.

4.4 Post-Optimization

In this section we briefly summarize post-optimizations of BonnMacro. Recall that such procedures are applied to a completely legal macro placement, which in particular is disjoint and obeys grid constraints. Any post-optimization preserves this property.

The first such step targets unused space for standard circuits. We are able to optimize free space next to macros by trading additional movement for more **whitespace**. This is useful in order to easily facilitate placement of standard cells right next to macros (e.g. during buffering).

Optimization for whitespace is approached similarly as presented in MacroLegalization: We consider local subinstances and re-optimize spatial relations while allowing

limited additional movement. The objective is extra free space around macros which receives higher reward in close proximity of the macros.

This can be modeled in a similar MIP to the one presented in Section 4.2.6. We just need to consider extra variables for the amount of whitespace around each rectangle. These variables artificially enlarge the size of the rectangles. Details of this approach have been presented by Wochnik (2017).

Mixed-integer programs targeting whitespace are considerably smaller and thus easier to solve than those considered for maximizing distance bound slack. Thus we can apply this type of post-optimization for our experiments discussed in Section 6.6. The whitespace made available will ensure that standard logic can be placed between macros, which is assumed by our distance bound model. In order to preserve the optimized worst slack overall, we only honor few circuit rows of whitespace and tightly restrict additional movement.

In addition, we are also able to improve macro **alignment**. Aligned macro borders result in straight routing channels. Such structures considerably simplify routing and thus usually reduce routing congestion.

BonnMacro supports a greedy post-optimization for alignment with bounded additional movement. We thereby align groups of macros to enforce a shared x- or y-coordinate. This is achieved by iteratively uniting the closest groups of macros starting with each macro as singleton group. Details of this approach are presented in Engels (2013).

Since routability is not the primary focus of this thesis, we did not apply alignment post-optimization here.

Chapter 5

Global Placement

In this chapter we discuss an enhanced version of BonnPlaceGlobal, the global placement component of BonnPlace. BonnPlace is the placement framework of the BonnTools (Korte, Rautenbach and Vygen 2007), a suite of physical design optimization tools developed at the Research Institute for Discrete Mathematics of the University of Bonn in an industrial cooperation with IBM.

An efficient and reliable global placer is essential: It is required for assessing the quality of a macro placement. For this purpose we fix the positions of all macros and consider the remaining placement instance as global placement problem (cf. Section 2.2.3). But also shredded placement discussed in Section 4.1 already relied on BonnPlaceGlobal.

As first ingredient we describe an effective speedup technique in Section 5.1. It serves as important improvement for instances with large chip images and complicated movebound structures which particularly arise in early chip design phases.

Afterwards we present our extension of BonnPlace: Self-Stabilizing BonnPlace (Brenner et al. 2015). It incorporates a partitioning-based legalization into a force-directed loop by iteratively pulling circuits towards their positions in a legalized placement. This self-stabilizing algorithm combines the accuracy of partitioning-based methods with the stability of force-directed placement strategies. It is capable of optimizing netlength as well as more involved objective functions like routability and timing behavior. We present an overview of this approach in Section 5.2, before we discuss congestion- and timing-driven extension in Section 5.3 and Section 5.4.

5.1 Fast Partitioning

BonnPlace is a partitioning-based global placement framework (Brenner, Struzyna and Vygen 2008; Struzyna 2011, 2013). We also denote this algorithm by **partitioning-based BonnPlace** or BonnPlaceGlobal to distinguish it from the version presented in Section 5.2.

It consists of two major components: An analytical placer that finds locations with small netlength but significant overlaps and a partitioning algorithm that assigns circuits to windows of a grid. Both steps are repeated recursively until the grid windows are so small that remaining overlaps can be resolved locally. The phase of partitioning-based BonnPlace during which we work on the same grid is called a **level**.

As **analytical placement** component, we optimize super-linear approximations of netlength. We are able to minimize quadratic netlength by solving a linear equation system with the conjugate gradient method (Hestenes and Stiefel 1952) as described by

Brenner, Struzyna and Vygen (2008). Moreover due to Struzyna (2013), we can also use this approach to approximate netlength with the "bound-to-bound" (**B2B**) netmodel (Spindler, Schlichtmann and Johannes 2008).

The subsequent **partitioning** assigns all cells to windows of a grid. This is done under the constraint that individual windows are not filled with too many cells. As objective we minimize total weighted cell movement from the analytical placement locations.

As this problem contains Partition, it is NP-complete to even decide whether a feasible solution exists. Thus we naturally consider the fractional version of partitioning which is a Hitchcock Problem, i.e. a min-cost b-flow problem on a graph $G \subseteq (A \cup B, A \times B)$ for sets A and B. Due to Shmoys and Tardos (1993) and Vygen (2005), fractional solutions can be rounded with minor violation of window capacities.

In the case of partitioning, $A := C$ and $B := W$ for the set of grid windows W. By virtue of recursive partitioning we have $\mathcal{O}(4^d)$ grid windows in placement level d. Thus on practical instances with millions of cells, the partitioning problem can not be solved globally.

Instead we make use of the **Flow-Based Partitioning** approach (Struzyna 2011): We first compute globally how cells are moved out of overfull windows. This is done on a graph on nodes W with constantly many edges per window $w \in W$. Since the underlying graph does not contain individual cells, the actual movement cost can only be approximated. But on the upside, such a flow provides the global information that can subsequently be realized locally. Such local subinstances are solved as Hitchcock Problems on individual cells and only few neighboring windows.

The actual implementation of BonnPlace used in practice contains multiple further heuristics, most importantly cell clustering and repartitioning. Since not particularly relevant here, we only refer to Brenner et al. (2015) and Struzyna (2011) for the details.

5.1.1 Movebound Constraints in Partitioning

From above introduction it is not apparent how to deal with movebound constraints (cf. Definition 2.12). We now elaborate how those are incorporated into partitioning.

To do this we use the concept of regions: For any window $w \in W$, we consider a partition of the window w into rectilinear shapes denoted by **regions $R(w)$**. For the purpose of partitioning, a region is semantically equivalent to a window. They both provide an area and a capacity for cells (respecting the target density). Thus we can equivalently assign cells to regions in preference of windows.

Respecting regions in partitioning has advantages and disadvantages: Clearly regions provide a more detailed view. In particular movement costs can be modeled more accurately by using multiple regions for a window. Additionally they provide the necessary flexibility to incorporate movebound constraints as we will discuss shortly. On the other hand, fine-grained partitions into regions increase the resulting size of min-cost flow instances. Consequently they need be used cautiously in order to control running time.

Next we describe how to respect movebounds by means of regions.

Definition 5.1. Let M be a set of movebounds and R a rectilinear shape. R is **M-homogeneous** if $R \cap m \in \{R, \varnothing\}$ for any $m \in M$. If $R \subseteq m$ for an M-homogeneous R and movebound $m \in M$, m is **involved** in R.

Remember that we use the geometry explained in Section 2.1 (page 3) here, particularly Definitions 2.1 to 2.4.

If we consider regions with areas that are M-homogeneous, we can utilize M-homogeneity in partitioning. More precisely, in this case it is possible to restrict the global flow-based partitioning graph as well as the local realization instances to assignments of cells to windows that are feasible w.r.t. movebounds. This can be done by omitting edges in the resulting min-cost flow instances. The details are described in Struzyna (2010, 2011).

But how can we find regions with these properties? This can be done using the following important geometric construction:

Definition 5.2. Consider a rectilinear shape R and a set of movebounds M. The ***M*-homogeneous partition** of R is a partition of R into M-homogeneous, rectilinear shapes $R_i \subseteq R$ for $i \in [\![k]\!]$ with unique sets of involved movebounds, i.e. $R = \bigcup_{i \in [\![k]\!]} R_i$.

Note that by Definition 5.2, distinct rectilinear shapes R_i and R_j in an M-homogeneous partition have different sets of involved movebounds. Since both are M-homogeneous $R_i \cap R_j = \varnothing$. This justifies the term *partition* and demonstrates that the M-homogeneous partition in fact is unique.

Struzyna (2011, page 43) computes M-homogeneous partitions by explicitly enumerating sets of involved movebounds. The actual implementation prunes subsets of movebounds with empty intersections. Nevertheless this approach has running time $\Omega(2^k)$ where k denotes the maximum number of movebounds with non-empty intersection. Consequently this algorithm has never been applicable to complicated, overlapping movebounds.

At the same time, M-homogeneous partitions can be computed efficiently:

Proposition 5.3. *Consider a rectilinear shape R and a set of movebounds M. We can compute the M-homogeneous partition of R in*

$$\mathcal{O}(\rho \log \rho + |M| \omega (\log \rho + \log \omega))$$

running time where ρ denotes the size of representations for R and all movebounds $m \in M$ and ω is the size of the output.

Proof. We first compute homogeneous rectangles by a standard sweep-line algorithm and afterwards build a homogeneous partition based on these.

The first part uses a sweep-line on left and right borders of rectangles in representations of R and $m \in M$ sorted by their respective x-coordinates. Upon iterating all events, we maintain the sweep-line as balanced binary search-tree of y-coordinate intervals that currently are covered M-homogeneously. Examples of such trees are AVL-trees due to Adelson-Velsky and Landis (1962) or Red-Black-trees due to Bayer (1972) and Guibas and Sedgewick (1978). In addition to the y-coordinate intervals, we also track how many rectangles of each movebound $m \in M$ are involved in each such interval (e.g. by an array of length $|M|$).

Our sweep-line procedure identifies ω M-homogeneous rectangles including involved movebounds as bitset of length $|M|$ for each of these rectangles. For all events together, we can update the sweep-line, mentioned counters and the result in $\mathcal{O}(|M|\omega)$. Sorting all events initially can be done in $\mathcal{O}(\rho \log \rho)$ and processing all events thus takes $\mathcal{O}(|M|\omega \log \rho)$ running time.

As a second step, we transform this representation into rectilinear shapes with unique movebound bitsets. This can be done by sorting. Since each comparison of bitsets requires

$\mathcal{O}(|M|)$ running time, sorting takes $\mathcal{O}(|M|\omega \log \omega)$ overall. This implies the stated running time. ■

We would like to point out that the M-homogeneous partition is part of the Hanan-Grid (Hanan 1966) of the corners of representations of R and m for $m \in M$. Thus $\omega \in \mathcal{O}\left(\rho^2\right)$ and Proposition 5.3 implies a polynomial time algorithm.

Although initially computed with exponential running time, Struzyna (2011) makes use of M-homogeneous partitions of windows in order to determine decompositions into regions. The approach relies on quad-trees due to Finkel and Bentley (1974).

This data structure is very efficient in practice for querying all rectangles intersecting a given query rectangle. In practice such queries can be answered in $\sim\mathcal{O}(d+\omega)$ where d is the depth of the quad-tree and ω is the size of the result. Nonetheless this can not be stated more precisely as there are examples known for queries in quad-trees of depth d on n rectangles requiring $\mathcal{O}(2^d + n)$ running time. To emphasize this, we use the $\sim\mathcal{O}(\cdot)$-notation for statements on running time involving quad-trees.

In order to compare different approaches for computing regions, we state the status quo of Struzyna (2011) in the following lemma:

Lemma 5.4. *Consider a set W of windows of a grid, M of movebounds and a rectilinear shape B of blockages. We can partition all windows $w \in W$ into M-homogeneous regions $R(w)$ in*

$$\sim\mathcal{O}\left(\sum_{\gamma\in\{\beta,\mu\}}[\gamma\log\gamma] + \sum_{w\in W}\left[(\log\mu + \mu_w\log\mu_w) + \sum_{r\in R(w)}[\log\beta+\beta_r]\right]\right)$$

time, where we denote the size of representations of all movebounds M by μ, those movebounds $m \in M$ with $w \cap m \neq \varnothing$ by μ_w for $w \in W$, all blockages B by β and finally those $b \in B$ for which $r \cap b \neq \varnothing$ by β_r.

Proof. We store movebounds and blockages into separate quad-trees. The running time for this initialization can be bounded by the first summand.

For any window $w \in W$, we thereby can determine the movebounds M_w intersecting w in $\sim\mathcal{O}(\log\mu + \mu_w)$ running time. Afterwards we can partition w M_w-homogeneously into rectilinear shapes w_i in $\mathcal{O}(\mu_w \log\mu_w)$ running time due to Proposition 5.3. Subsequently we determine $R(w)$ by introducing a region for each w_i.

Analogously we also can determine the regions' capacities: For any region r we query the blockages within r in $\sim\mathcal{O}(\log\beta + \beta_r)$ running time by a single query for the bounding box of r. For all those rectangles we subtract the area of the intersection with r from the partitioning capacity of r.

This implies the stated running time overall. ■

Struzyna (2011) even chose to introduce separate regions for each rectangle in the representation of w_i. Consequently all regions are defined by single rectangles only. This was done in order to model movement-costs more accurately when computing the global cell movement in the first stage of flow-based partitioning.

To get an impression what this can result into, we consider the instance Mia presented in Figure 5.1. What makes this particular instance challenging in practice is not the pure number of 85 movebounds. As Figure 5.2 illustrates, the movebounds particularly in the

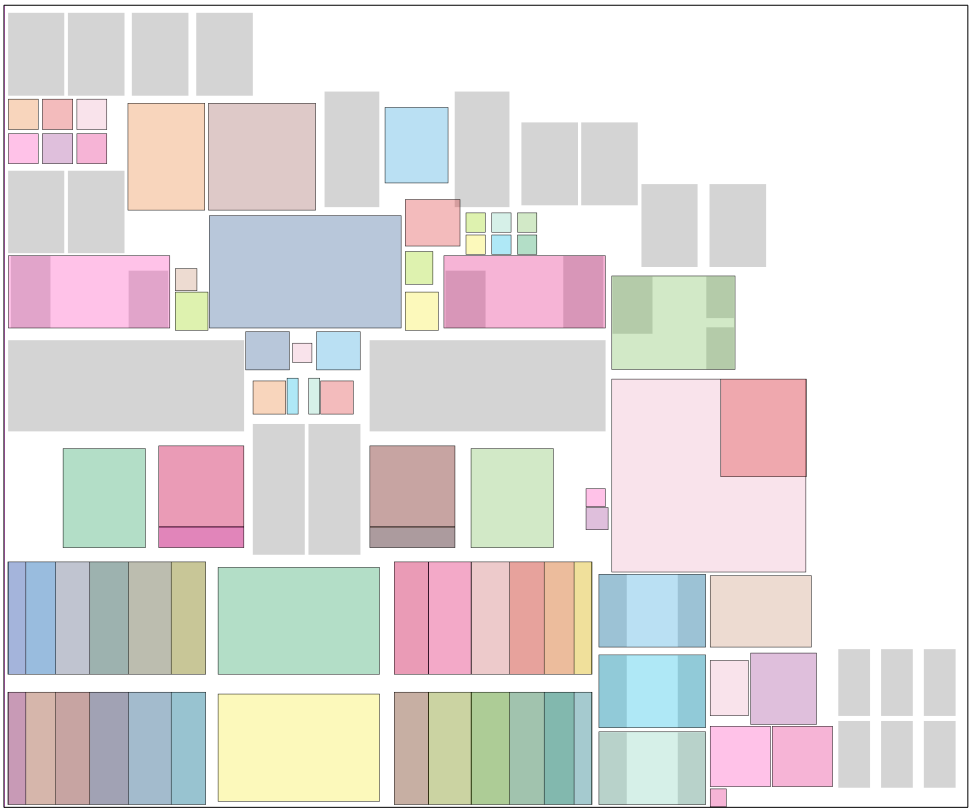


Figure 5.1: Overview of Mia with 85 movebounds. Blockages are drawn in gray; colored shapes represent non-default movebounds.

lower left area of the chip overlap frequently. Thus an M-homogeneous partition of the chip area contains many elements for all these distinct, tiny intersections.

These in turn need to be modeled by different regions for partitioning. With the original approach by Struzyna (2011) this leads to 317 regions in the first level of partitioning. These regions are visualized in Figure 5.3. In this figure regions are colored purely with the purpose to differentiate neighboring regions visually. The grid in this level 1 partitions the entire chip image into 4 windows (that is clearly visible in Figure 5.4).

The downside of this emphasis on precision is apparent: It significantly increases the complexity of the min-cost flow instances both for the global flow and more importantly for the local realizations. In the latter case in particular, this is crucial since local realizations require one edge in the min-cost flow instance for each possible assignment of circuit to region. Consequently the running time spent on local realization depends super-linearly on the number of regions.

In practice this effect can especially be observed during the first levels of global placement. Here windows are fairly large and often intersect a considerable fraction of all movebounds. This often results in intricate homogeneous movebound partitions and thus similar regions. Moreover in the first levels, there are only few windows in the grid, which is why local realizations can barely be computed in parallel. An example where this can be seen prominently is given in Figure 5.3.

On the other hand, this running time complexity can easily be avoided by restricting created regions. As indicated in the proof of Lemma 5.4 it is applicable to create a single

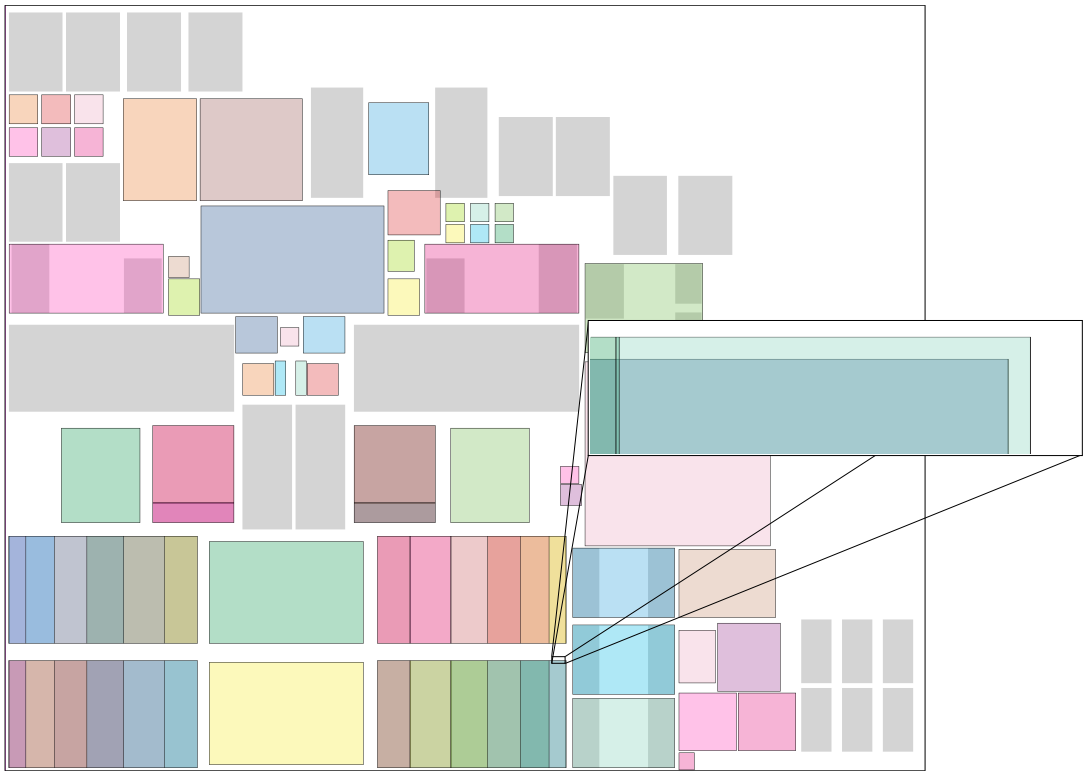


Figure 5.2: Closeup of overlapping movebounds on Mia.

region for each element w_i of the M-homogeneous partition of w.

The effect of this change is presented on the same instance Mia in Figure 5.4. The resulting reduction of 317 regions (cf. Figure 5.3) down to 166 is apparent. As a side effect Figure 5.4 also clearly visualizes the original grid windows.

5.1.2 Weak Movebound Homogeneity

But this still leaves room for improving the number of regions: Suppose there are two movebounds $m_0, m_1 \in M$ with $m_0 \neq m_1$ and $m_i \cap m' = \varnothing$ for any $m' \in M \setminus \{m_0, m_1\}$, $i \in [\![2]\!]$. Further assume $m_i \subseteq w$ for some window $w \in W$ and both $i \in [\![2]\!]$. Note that m_0 and m_1 are allowed to overlap. In this case, the approach of Lemma 5.4 results in (at least two) distinct regions. Figure 5.4 shows multiple movebounds with this properties e.g. within the upper left window.

But assigning to either of the regions where m_i are involved is actually equivalent for the purpose of partitioning in this level: As $m_i \subseteq w$, all cells of both m_i have to be moved into w anyway. Due to $m_i \cap m' = \varnothing$ for $m' \in M \setminus \{m_0, m_1\}$, no m'-cells require capacity in any m_i. Consequently it is not required to distinguish between both m_i. Thus we actually only need a single region for assignments to either of those movebounds inside w.

Note that by unifying all regions where m_i are involved, the movement of assigning m_i-cells is modeled less accurately. But by the previous arguments this can never change the optimum partition into the underlying windows W.

We emphasize that this can only be done due to $m_i \subseteq w$ for both $i \in [\![2]\!]$. Suppose $m_0 \subseteq w$, $m_0 \cap m_1 \neq \varnothing$ but $m_1 \not\subseteq w$. Here the utilization of $m_0 \cap m_1$ as well as $m_1 \cap w$ depends on the actual movement costs and is unknown in advance. In order to preserve

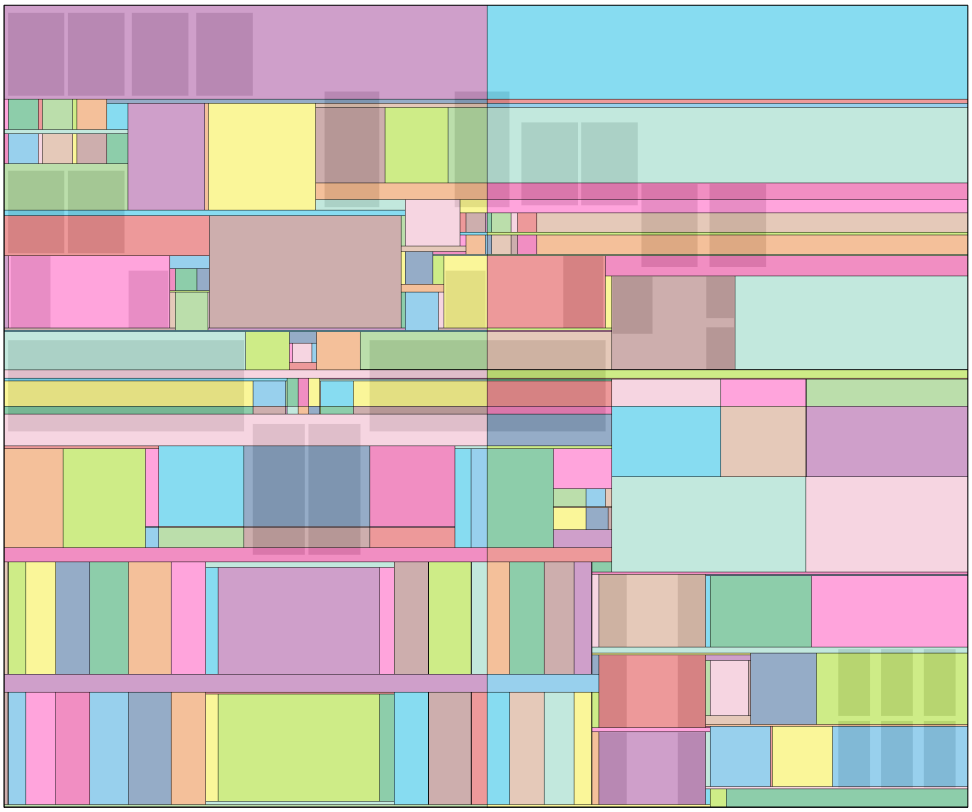


Figure 5.3: Overview of 317 regions in partitioning level 1 on Mia generated by the Struzyna (2011) approach. Regions are colored distinctly in order to improve visual differentiation.

the optimum partition of cells into windows, we thus have to model above parts of w by distinct regions.

We will now generalize this idea to larger groups of movebounds and elaborate how to exploit it efficiently. We start by generalizing the notion of movebound homogeneity:

Definition 5.5. Consider a set M of movebounds and a rectilinear shape $R \subseteq \bigcup_{m \in M} m$. A **weakly M-homogeneous partition of R** is a partition of R into rectilinear shapes $R_i \subseteq R$ for $i \in [\![k]\!]$ where $R_i \cap m \in \{ R_i, m, \varnothing \}$ for any $m \in M$ and the movebounds $\{ m \in M : R_i \cap m \neq \varnothing \}$ involved in R_i are unique to R_i for any $i \in [\![k]\!]$.

The partition $\{ R_i : i \in [\![k]\!] \}$ is said to have **cardinality** k.

Weakly M-homogeneous partitions can be utilized for the purpose of global partitioning during placement. We partition every window $w \in W$ of the placement grid weakly M-homogeneously. As indicated before, it is sufficient to introduce one region for any element R_i of such a partition. Consequently minimizing the cardinality of these partitions is an important objective.

(Strongly) M-homogeneous movebound partitions are also weakly M-homogeneous, so we know such partitions exist. But in contrast to their strong counterparts, a weakly M-homogeneous partition is neither unique nor has unique cardinality. Finding a weakly M-homogeneous partition with minimum cardinality is an optimization problem. Next we discuss how to solve this problem optimally.

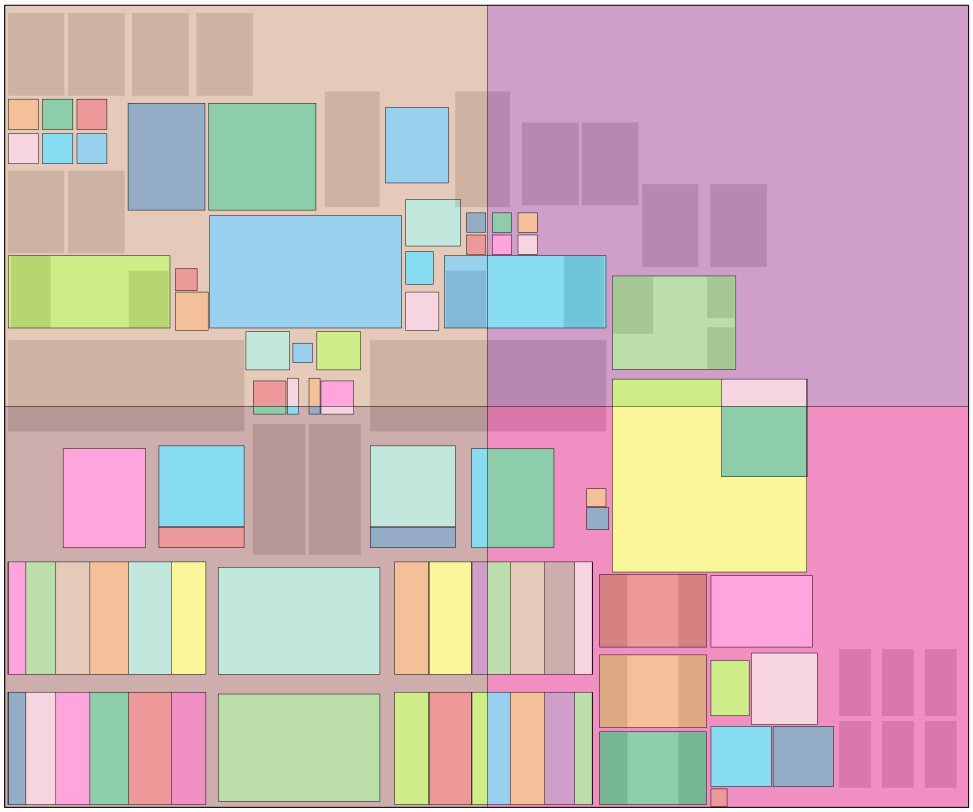


Figure 5.4: Overview of 166 regions in partitioning level 1 on Mia generated based on M-homogeneous partitions of all windows using Lemma 5.4.

Definition 5.6. For a set M of movebounds the **movebound graph G_M** is the directed graph $G_M := (M, E_M)$ where

$$E_M := \{ (m_0, m_1) : m_i \in M,\ m_0 \cap m_1 \neq \varnothing,\ m_1 \not\subseteq m_0 \}.$$

Definition 5.6 is motivated as follows: Consider a weakly M-homogeneous partition R_i for $i \in [\![n]\!]$ of rectilinear shape R. Suppose $m \in M$ satisfies $m \not\subseteq R_i$ for any $i \in [\![n]\!]$ but $m \cap R \neq \varnothing$. If $(m, m') \in E_M$, $m \cap m'$ and $m' \setminus m \neq \varnothing$ must be covered by a different R_i by weak movebound homogeneity. Consequently also m' satisfies $m' \not\subseteq R_i$ for any $i \in [\![n]\!]$.

We now elaborate formally, how this reduces the identification of certain important movebounds to a reachability problem in G_M. As a first step, we examine a particular set of movebounds $M' \subseteq M$ which are sufficient for finding weakly M-homogeneous partitions:

Proposition 5.7. *Consider a set of movebounds M and a rectilinear shape $R \subseteq \bigcup_{m \in M} m$. Let $M' \subseteq M$ be the movebounds reachable from*

$$S := \{ m \in M : m \cap R \neq \varnothing,\ m \not\subseteq R \}$$

in G_M. Further define $M^* := M' \cup \{ m^* \}$ for an artificial movebound m^* with $m^* := R \setminus \bigcup_{m' \in M'} m'$.

In this context any (strongly) M^*-homogeneous partition R_i of R is weakly M-homogeneous.

Proof. For $m' \in M'$ we have $R_i \cap m' \in \{\varnothing, R_i\} \subseteq \{\varnothing, R_i, m'\}$ for any R_i by strong M'-homogeneity of R_i.

Otherwise $m \in M \setminus M'$, for which we claim $R_i \cap m \in \{\varnothing, m\}$ for any R_i: As $m \notin M'$, for any $m' \in M'$ we have $(m', m) \notin E_M$ by choice of M'. By Definition 5.6 of G_M, for any $m' \in M'$ thus either $m' \cap m = \varnothing$ or $m \subseteq m'$. By definition this is also the case for m^*. Consequently m is strongly M^*-homogeneous. If $R_i \cap m = \varnothing$ there is nothing to show. Otherwise $m \subseteq R_i$ as both R_i and m are strongly M^*-homogeneous. ■

Note that by choice of m^* in Proposition 5.7, $R \subseteq \bigcup_{m \in M^*} m$. Thus a strongly M^*-homogeneous partition of R exists.

Next we extend the motivation for Definition 5.6 of G_M inductively. This establishes a lower bound on the cardinality of any weakly M-homogeneous partition:

Lemma 5.8. *Consider a set of movebounds M, a rectilinear shape $R \subseteq \bigcup_{m \in M} m$ and $S, M' \subseteq M$ as defined in Proposition 5.7. Further let R_i for $i \in [\![n]\!]$ be any weakly M-homogeneous partition of R. In this case $R_i \cap m' \in \{R_i, \varnothing\}$ for all $m' \in M'$ and $i \in [\![n]\!]$.*

Proof. Let $m' \in M'$. We assume $m' \cap R \neq \varnothing$ as otherwise there is nothing to show. By definition of M' there is a path $P = (m_0, \ldots, m_k, m')$ in G_M where $m_0 \in S$. We use induction on the length of a shortest such S-m'-path P.

For the induction basis $m_0 = m'$ and thus $m' \in S$. By definition of S we have $m' \setminus R \neq \varnothing$ which implies $m' \cap R_i \neq m'$ for any $R_i \subseteq R$. As R_i is a weakly M-homogeneous partition, $R_i \cap m \in \{R_i, \varnothing\}$.

In the induction step $R_i \cap m_k \in \{R_i, \varnothing\}$ for any R_i. By choice of P as shortest S-m'-path, we have $m' \notin S$. As $m' \cap R \neq \varnothing$, this implies $m' \subseteq R$. By the edge $(m_k, m') \in E_M$, we obtain $\varnothing \neq m_k \cap m' \subseteq m' \subseteq R$ and $m' \not\subseteq m_k$.

As R_i for $i \in [\![n]\!]$ are a partition of R, there is an R_j with $R_j \cap m_k \cap m' \neq \varnothing$. By induction hypothesis $R_j \cap m_k = R_j$. As $m' \not\subseteq m_k$, $m' \setminus m_k \neq \varnothing$ by which $m' \setminus R_j \neq \varnothing$ i.e. $R_j \cap m' \neq m'$. As $R_j \cap m' \in \{\varnothing, m', R_j\}$ due to weak movebound homogeneity, we must have $R_j \cap m' = R_j$. As R_i for $i \in [\![n]\!]$ are a partition of R and particularly disjoint to R_j, there is no R_i with $R_i \cap m' = m'$. Thus $R_i \cap m' \in \{\varnothing, R_i\}$. ■

We can use this lower bound of Lemma 5.8 to analyze the construction of Proposition 5.7. As it turns out, this weakly M-homogeneous partition in fact has minimum cardinality:

Theorem 5.9. *Consider a set of movebounds M and a rectilinear shape $R \subseteq \bigcup_{m \in M} m$. The weakly M-homogeneous partition constructed in Proposition 5.7 has minimum cardinality.*

Proof. Let M^* as defined in Proposition 5.7. By Lemma 5.8 any weakly M-homogeneous partition of R is strongly M^*-homogeneous. By Definition 5.2 strongly M^*-homogeneous partitions are unique and particularly have unique cardinality. Consequently the (unique) cardinality of a strongly M^*-homogeneous partition of R is a lower bound on the minimum cardinality of any weakly M-homogeneous partition of R. Since the partition of Proposition 5.7 attains this lower bound, it has minimum cardinality. ■

Using Theorem 5.9 we can compute weakly M-homogeneous partitions as fast as strongly ones:

Corollary 5.10. *Consider a set of movebounds M and a rectilinear shape $R \subseteq \bigcup_{m \in M} m$. We can compute a weakly M-homogeneous partition of R with minimum cardinality*

$$\mathcal{O}(\rho \log \rho + |M|\omega(\log \rho + \log \omega))$$

where ρ denotes the size of representations for R and all movebounds $m \in M$ and ω is the size of a representation of the strongly M-homogeneous partition of R.

As we discuss shortly, we did not implement the algorithm of Corollary 5.10. Instead we make use of a simpler approach which suffices in practice. Thus Corollary 5.10 is not central, which is why we only outline essential ideas of a technical proof here:

Proof (sketch). We can adapt the sweep-line algorithm of Proposition 5.3 to compute a graph $G = (M, E)$ with the following properties: G has $\mathcal{O}(\omega)$ edges and for any $m_0, m_1 \in M$ there is a m_0-m_1-path in G if and only if there is one in G_M. Thus we can compute $M' \subseteq M$ reachable from S (cf. Proposition 5.7) in G instead of G_M which takes $\mathcal{O}(\omega)$ running time.

Finally by Theorem 5.9 it is sufficient to compute a strongly M^*-homogeneous partition (cf. Proposition 5.7). As $R \subseteq \bigcup_{m \in M^*} m$, this partition exists. It can be computed in the stated running time by Proposition 5.3. ■

As mentioned, in practice we avoid making our sweep-line implementation even more involved. Instead of M', we work with a superset N with $M' \subseteq N$ which is easier to compute: We again start with the M-homogeneous partition R_i for $i \in [\![n]\!]$ of A. Afterwards we determine connected components in a modified, undirected movebound graph $G'_M := \left(M, E'_M\right)$ where

$$E'_M := \{\{m_0, m_1\} : m_i \in M, m_0 \neq m_1 \text{ and } m_0 \cap m_1 \neq \varnothing\}.$$

By traversing all R_i and their involved movebounds, we can identify N as the nodes reachable from S in G'_M. In particular, we thereby can avoid constructing G'_M explicitly. Using the well-known union-find data structure (Tarjan 1975) this takes $\mathcal{O}(\omega \cdot \alpha(|M|))$, where α denotes the inverse Ackermann function.

Recall that by Definition 5.6 the movebound graph G_M is directed and only introduces edges under additional requirements. Consequently $M' \subseteq N$ as claimed.

With the usual artificial movebound, we can actually determine a strongly N^*-homogeneous partition from R_i. This can be done by sorting all rectangles in representations of all R_i and has already been elaborated in the second part of the proof of Lemma 5.4. Since $M' \subseteq N$, the result of this operation is strongly M^*-homogeneous and thus in fact weakly M-homogeneous by Proposition 5.7.

Since $M' \subseteq N$, the result of our implementation is not necessarily a partition of minimum cardinality. But this de facto is not a huge restriction, since the circumstances of extra movebounds in $N \setminus M'$ happens rarely in practice, in particular not at all in the experiments presented in Section 6.2.

5.1.3 Weakly Movebound Homogeneous Regions

We can use Corollary 5.10 in each window $w \in W$ to find a set of regions $R(w)$ that is minimum for modelling partitioning accurately globally. This can be achieved using the approach of Lemma 5.4 while substituting the strongly M-homogeneous partition by a weakly one.

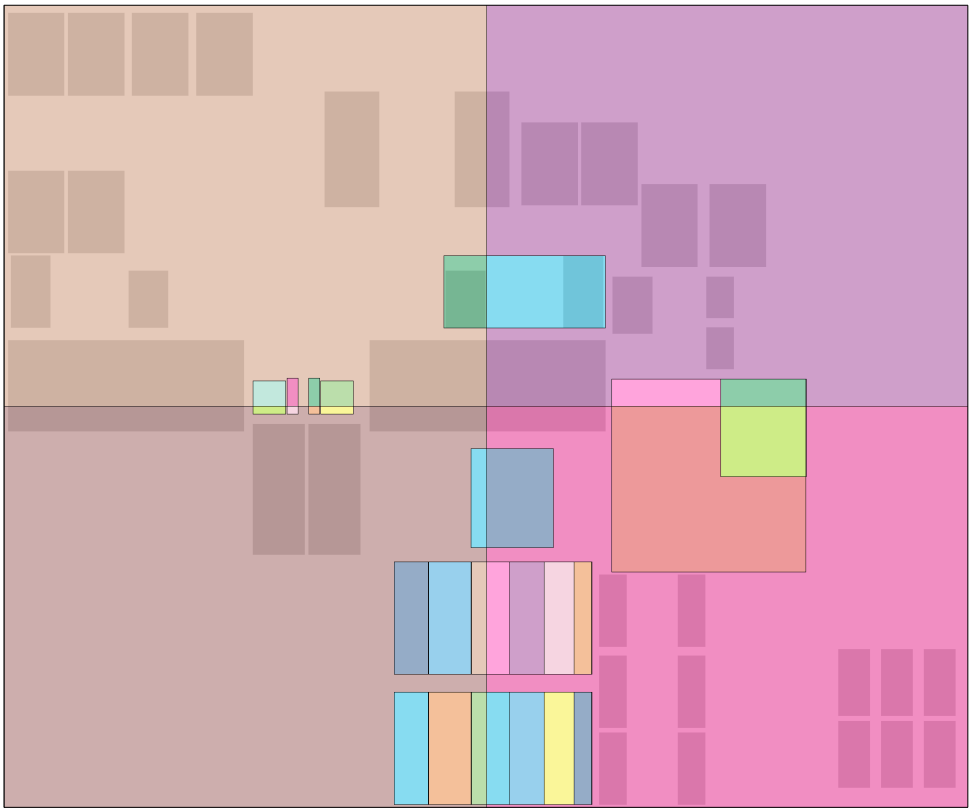


Figure 5.5: Overview of 70 regions in partitioning level 1 on Mia generated based on weakly M-homogeneous partitions of the windows.

This improves the number of created regions drastically. To illustrate this, we reconsider the instance Mia for which different region creation algorithms have been depicted in Figures 5.3 and 5.4. The 70 regions generated based on weakly M-homogeneous partitions of the windows are shown in Figure 5.5. As initially intended, we can clearly identify large groups of former regions that could be elided by making use of weak M-homogeneity.

Although the output of this approach is optimum in the sense of Theorem 5.9, it can not be computed quickly. Computing regions so far requires querying movebounds and blockages for each $w \in W$. Due to the recursive nature of partitioning based global placement, we can have millions of windows – especially in the last levels and especially on chips with large chip area.

But we can avoid using any quad-trees for this purpose whatsoever. Instead we exploit the grid structure of the windows W, which we have not done at all so far.

Proposition 5.11. *Consider a set of movebounds M in chip area A and a set of grid windows W subdividing A. We can partition all $w \in W$ into weakly M-homogeneous regions $R(w)$ with minimum cardinality in*

$$\mathcal{O}\Big(\mu \log \mu + |M|\gamma(\log \mu + \log \gamma) + (\gamma + |M|) \log|W| + \omega\Big)$$

running time, where ω is the size of representations of the output $R(w)$ for all $w \in W$, μ is the size of representations for all movebounds $m \in M$ and γ is the size of a representation for the (strongly) M-homogeneous partition of A.

Proof. Let $S := \{ m \in M : \nexists w \in W \text{ with } m \subseteq w \}$ and $M' \subseteq M$ the movebounds reachable from S in G_M. S and M' are the union of all S_w and M'_w used for finding weakly M-homogeneous partitions of w in Theorem 5.9. Further we specify the artificial movebound m^* by $m^* := A \setminus \bigcup_{m \in M'} m$ and denote $M^* := M \cup \{ m^* \}$.

Let H_i for $i \in [\![n]\!]$ be a strongly M^*-homogeneous partition of A. By Theorem 5.9, for each $w \in W$ the partition $H_w := \{ H_i \cap w : H_i \cap w \neq \varnothing, i \in [\![n]\!] \}$ is weakly M-homogeneous and has minimum cardinality. We will define regions $R(w)$ according to the partition H_w of $w \in W$.

We do so by using the grid structure of W as follows: For any point $p \in A$, we can find a window $w \in W$ containing p by binary search in $\mathcal{O}(\log|W|)$ running time. If p is exactly on the border between neighboring windows, there can be up to four such windows.

We can determine whether $m \in S$ by checking $m \subseteq w$ for a single $w \in W$: If such a w exists, it is a window closest to the center of m. Thus we detect $m \in S$ by checking all those constantly many closest windows. This takes $\mathcal{O}(\mu + |M| \log|W|)$ running time for all movebounds together.

Next we use Corollary 5.10 to compute the graph G with $\mathcal{O}(\gamma)$ edges containing a m_0-m_1-path if and only if the movebound graph G_M does so. Thus we determine M' as the movebounds reachable from S in G in $\mathcal{O}(\gamma)$ running time. By Corollary 5.10 computing G and all H_i can be done in $\mathcal{O}(\mu \log \mu + |M| \gamma (\log \mu + \log \gamma))$ running time.

Afterwards for each rectangle $h \in H_i$ and each $i \in [\![n]\!]$, we traverse all windows w intersecting h. Again we can find those windows by binary search exploiting the grid structure of W in $\mathcal{O}(\log|W|)$ running time. For each such pair (h, w) we either introduce a new region in $R(w)$ or add $h \cap w$ to r of an existing region $r \in R(w)$ for H_i. This consumes $\mathcal{O}(\gamma \log|W| + \omega)$ running time overall. ∎

In order to complete the construction of regions, we also need to consider blockages, which Proposition 5.11 does not do so far. We have two options to incorporate blockages:

It would be possible to subtract blockages B from the overall chip image A before applying Proposition 5.11. This is an elegant solution, but increases both γ and ω in the running time. In particular this can increase the complexity of the decomposition of w into r for $r \in R(w)$, even for $w \in W$ not intersecting B.

In order to avoid this effect, we first apply Proposition 5.11 and afterwards subtract blockages from regions. On the one hand, this introduces the overhead of finding these regions that intersect B. On the other hand, we thereby avoid any side effects on regions disjoint from B. This approach has the following running time:

Lemma 5.12. *Consider a rectilinear shape B of blockages represented by rectangles b_i for $i \in [\![n]\!]$. Let $R(w)$ be regions for all windows $w \in W$ of a grid. We can compute $r \setminus B$ for all $r \in R(w)$ and $w \in W$ in*

$$\sum_{i \in [\![n]\!]} \left(\log|W| + \sum_{r \in R(w) : w \cap b_i \neq \varnothing} \rho_r \right)$$

running time, where ρ_r denotes the size of a representation of r.

Proof. We process each b_i one after another. For each $i \in [\![n]\!]$ we compute the set $W_i := \{ w \in W : w \cap b_i \neq \varnothing \}$ in $\mathcal{O}(\log|W|)$ running time using binary search on W. This argument has already been discussed in the proof of Proposition 5.11. Subsequently we iterate all $r \in R(w)$ for $w \in W_i$ and subtract b_i from r in $\mathcal{O}(\rho_r)$ running time. ∎

Both Proposition 5.11 and Lemma 5.12 reflect the running time of our actual implementation. We now discuss two purely theoretical improvements that are not used in practice:

First of all, we can avoid binary search entirely: All windows, regions and blockages have integer coordinates. For each dimension separately, finding the grid line next to a given coordinate $x \in \mathbb{Z}$ thus can be accomplished in $\mathcal{O}(1)$ using a lookup table. This lookup table though is very large, namely linear in width and height of A (which is exponential in the input). Practical experiments showed that the overhead of populating the lookup table outweighs the speedup for querying windows. Thus we use binary search both in theory and practice.

Secondly Lemma 5.12 always traverses all regions of the windows W_i while actually only those intersecting b_i would be sufficient. Querying those regions efficiently is not an easy task. Falling back to one quad-tree for each window $w \in W$ storing region areas for $r \in R(w)$ is not an option: r is a rectilinear shape which can not be stored in general and storing $\mathrm{BB}(r)$ instead still leads to regions considered unnecessarily. Moreover the overhead of managing separate quad-trees for each $w \in W$ would be drastic – especially in the last levels of global placement.

Finally the overhead is expected to be minor overall: In the first levels, there are few regions altogether and even considering all of them for each b_i would be manageable. In contrast to this we have millions of regions in the final levels of global placement. But here each window consists of very few regions, usually even only a single one. Thus the number of unnecessarily considered regions remains small. This is why we use the approach in practice exactly as described in Lemma 5.12.

Now we want to compare the region construction in Lemma 5.4 against Proposition 5.11 plus Lemma 5.12: Clearly the latter combination finds strictly superior regions for partitioning. It computes regions $R(w)$ with weakly M-homogeneous areas with minimum cardinality. Even though we do not guarantee minimum cardinality in practice, we still demonstrated to significantly reduce the number of regions.

Besides output quality, also the required running time is preferable: For a start, the running time can be stated precisely as it does not use quad-trees any longer. Furthermore both Proposition 5.11 and Lemma 5.12 avoid all operations done for each and every $w \in W$ and $r \in R(w)$ respectively. As a consequence this combination is very valuable in practice – especially in the last levels of global placement. A detailed comparison of different region construction paradigms is presented in Section 6.2.

Finally we'd like to close this section by a few practical remarks:

BonnPlace supports non-uniform target densities. This is modeled by certain **density-rectangles** $d \subseteq A$ specifying a deviating target density inside d. We incorporate these similarly as blockages are handled in Lemma 5.12. The only difference is that we alter the capacity of $r \in R(w)$ instead of reducing r.

Furthermore computing regions for partitioning not only requires to calculate capacities only. In addition we also need to find the center of gravity of r for each region r. This can easily be computed during a final step as the weighted average of centers of $a \in r$ weighted by a's area. But this is computationally difficult as it averages products of 32-bit and 64-bit integers (the areas). Those products can not be represented by 64-bit integers which is the maximum available to us. In order to overcome this, we compute those products as floating point numbers and make use of the compensated summation method of Kahan (1965). This makes the error of summing n floating point numbers independent of n (Higham 1993).

Algorithm 5.1: Self-Stabilizing BonnPlace

Input : Cells C.
Output: Locations pos(c) for all cells $c \in C$.

```
1  pos :≡ 0, iter := 0
2  while not BreakCondition(pos, iter) do
3      foreach c ∈ C do
4          Connect c to a new pin at position pos(c) via a virtual net of weight
           0.01 · iter.
5      Partitioning-based BonnPlaceGlobal
6      BonnPlaceLegalization
7      Delete virtual nets
8      if timing_optimization_enabled then
9          BonnLayerAssignment
10         BonnRefinePlace
11         BonnPlaceLegalization
12     if routability_driven_placement_enabled then
13         CongestionAvoidance
14     Store current placement as pos, iter := iter + 1
```

5.2 Self-Stabilizing BonnPlace

We now present the self-stabilizing BonnPlace framework (Brenner et al. 2015) for Global Placement. This framework is a versatile algorithm based on various components of the BonnTools suite (Korte, Rautenbach and Vygen 2007) developed at the Research Institute for Discrete Mathematics of the University of Bonn. It can optimize netlength as well as timing properties (Section 5.3) and routability (Section 5.4).

Self-stabilizing BonnPlace combines the two strongly opposing placement strategies described in Section 2.4: force-directed and partitioning-based placement. In essence we perform multiple **iterations** of global placement followed by (standard cell) legalization. In our case this is BonnPlaceGlobal plus BonnPlaceLegalization.

Each iteration learns from previous ones. The insights gained in past iterations are transferred purely via virtual nets which we call **forces**. We thereby artificially pull cells towards some position derived from former iterations. Algorithm 5.1 outlines this placement strategy.

This approach provides the flexibility to incorporate multiple optimizations for various placement objectives and constraints. We demonstrate how our placement scheme stabilizes towards a solution meeting opposing placement goals. This justifies the name Self-Stabilizing BonnPlace.

5.2.1 General Framework

A key ingredient of our tool is the partitioning-based global placement algorithm BonnPlaceGlobal (Algorithm 5.1, line 5). As described in Section 5.1, BonnPlaceGlobal iterates analytical placement and flow-based partitioning in multiple levels.

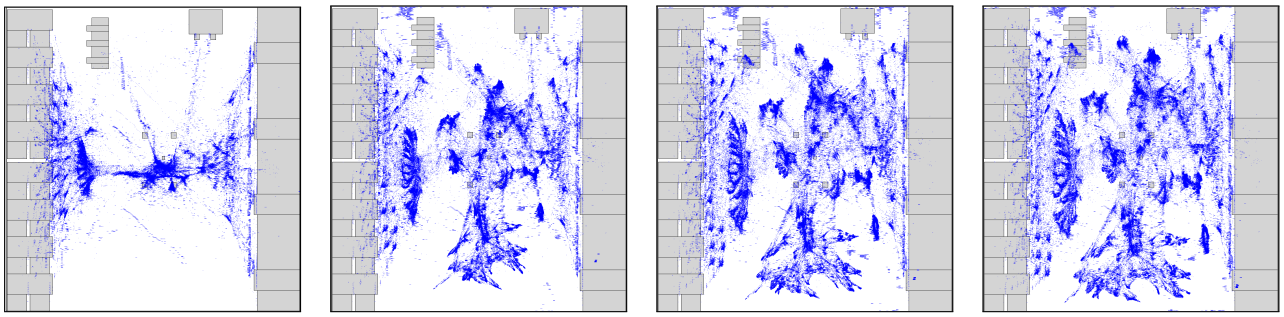
The analytical placement, which is computed in the very first level 0 of each iteration, is of particular interest. In this level the placement grid is trivial i.e. contains only a

single window. Referring to the notion of SimPL (Kim, Lee and Markov 2012, 2013), the remainder of BonnPlaceGlobal plus BonnPlaceLegalization can be seen as legalization of this initial placement.

Over the course of iterations, level 0 placements contain fewer overlaps (cf. Figures 5.6 and 5.7 and Section 6.3). The extended spreading predicts inevitable distortions for achieving legality. It also helps BonnPlaceGlobal anticipating the effects of partitioning decisions more accurately. On the other hand, those level 0 placements still optimize netlength on a global scale. Consequently in later iterations we can legalize these locations with fewer movement and smaller netlength increase.

BonnPlaceGlobal (Brenner, Struzyna and Vygen 2008; Struzyna 2011) optimizes (weighted) quadratic netlength using the clique-netmodel for nets with few pins and the star-netmodel otherwise (Brenner and Vygen 2001). For pure netlength minimization we instead make use of a combination of the bound-to-bound net model (B2B) with a superlinear distance scaling factor (Struzyna 2013). As of iteration one, we use the current placement to determine the outer pins of nets for the B2B net model. Thus we can avoid the computation of multiple QPs with B2B in order to stabilize the solution.

The result of BonnPlaceGlobal respects the target density accurately, but contains overlaps that need to be resolved locally. For this purpose we use BonnPlaceLegalization of Brenner (2012) (Algorithm 5.1, line 6). This algorithm minimizes quadratic movement by incrementally clearing overlaps similar to the Successive Shortest Path Algorithm (Edmonds and Karp 1972; Tomizawa 1971).



(a) Iteration 0 **(b)** Iteration 1 **(c)** Iteration 2 **(d)** Iteration 3

Figure 5.6: Spreading of cells on Renaud in the first analytical placement in the first four iterations. Each cell location is drawn as single blue point.

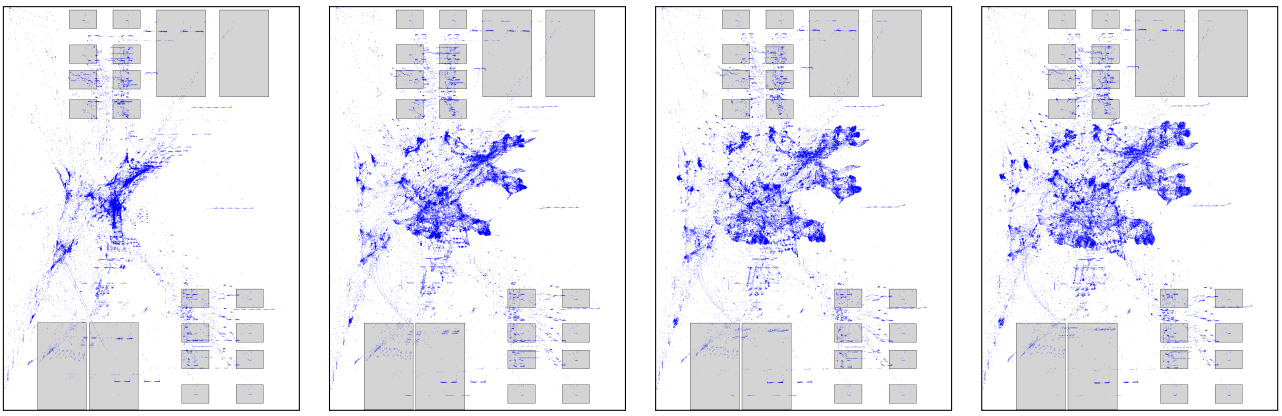
In the first iteration (iter 0) of our self-stabilizing global placer, we do not know any legal placement. Thus we skip introducing any forces whatsoever. Afterwards the forces (cf. Section 5.2.2) are respected throughout BonnPlaceGlobal i.e. they affect the solution in each level and not only the global analytical placement in level 0.

In contrast to other force-directed approaches, we spend more running time for a single iteration. On the other hand, each iteration not only results in a placement respecting all constraints (legality, movebounds, target density, blockages cf. Section 2.2) but also is of high quality (Struzyna 2011, 2013).

This allows us to reduce the number of iterations drastically: SimPL (Kim, Lee and Markov 2012, 2013) requires ∼50 iterations, ePlace (Lu, Chen et al. 2015) even more than 250. In contrast, for self-stabilizing BonnPlace depending on additional optimization objectives (Sections 5.3 and 5.4) 5 to 15 iterations suffice. We optimize elaborately and thus reliably in each iteration, which allows a steeper increase of forces.

5.2.2 Forces

In our placement algorithm, forces are used to transfer information and to stabilize the placement procedure. The higher the quality is of information that we get from earlier iterations, the greater is the efficiency of forces. We spend much effort on legalizing each analytical placement to a solution of high quality. Transferring this kind of collected information by means of forces is effective.



(a) Iteration 0 **(b)** Iteration 1 **(c)** Iteration 2 **(d)** Iteration 3

Figure 5.7: Spreading of cells on Franziska in the first analytical placement in the first four iterations. Each cell location is drawn as single blue point.

We add forces at the beginning of an iteration (Algorithm 5.1, lines 3 to 4). For this construction of forces we can use placement information pos stored at the end of the previous iteration (Algorithm 5.1, line 14). The first iter 0 is performed without forces. During each subsequent iteration iter > 0 of self-stabilizing BonnPlace, there is exactly one force pulling at each cell.

Forces are modeled as artificial two-terminal nets connecting any cell to some fixed position on the chip. Those nets are handled exactly the same way as regular nets. In particular artificial nets affect the distribution of cells during the analytical placements, but have no (direct) influence on other steps minimizing movement, e.g. flow-based partitioning.

For each cell c in the set of circuits C, the artificial force-net is connected to the center of c. The fixed pin of the force-net is located at pos(c), i.e. the location of c determined in the previous iteration. This position is called the **target point** of the force pulling at c. We have experimented with different target points incorporating further past iterations, but this could not improve practical results (Hoppmann 2014).

The weight of force nets increases in each iteration to stabilize the placement. Higher stability is expressed by decreasing cell movement between two consecutive iterations and converging analytical and legalized placement netlengths. On the one hand, forces have to be very effective. As we legalize analytical placements with high effort, we want to transfer as much of the information gained in previous iterations as possible. In particular we should induce a significant cell spreading in the analytical placement of subsequent iterations. On the other hand, force weights need to be chosen such that artificial nets are not predominant for netlength minimization. We need to provide guidance, while guaranteeing enough flexibility for our partitioning-based global placer BonnPlaceGlobal.

We achieve this by a weight of $0.01 \cdot \mathsf{iter}$. The average non-artificial net weight is 1. Thus our choice of force-weights defines a 1 %-increase in each iteration compared to the standard net weight. Different uniform and non-uniform force weights provided no benefits in practice for minimizing netlength (Hoppmann 2014) which is why we stick to this simple scheme. Increasing force-weights non-uniformly is of particular interest for optimizing timing characteristics and will be discussed in Section 5.4 in detail.

The impact of forces to the quadratic placement is shown in Figures 5.6 and 5.7 on two industrial designs. We see the analytical placement computed in level 0 of BonnPlaceGlobal, where we only have one window representing the whole chip area. We show the these solutions over the course of the first four iterations ($\mathsf{iter} \in [\![4]\!]$). The spreading is already significant, but far from giving a legal solution. This demonstrates how we are able to balance increased spreading and freedom for BonnPlaceGlobal. Although we are able to provide significant guidance and enforce stability, we still leave considerably many choices for this partitioning-based placer.

When a legal placement is found, we delete forces (Algorithm 5.1, line 7). In the following iteration, new forces will be computed.

5.2.3 Breaking Conditions

The general breaking condition for the overall loop is a user-specified maximum number of iterations – typically around 5 to 10. But the Self-Stabilizing BonnPlace framework is very flexible and allows further extensions.

In order to save running time, we can skip remaining iterations based on the cells' movement in the last iterations. As future iterations with higher force weights are expected to deviate less from previous placements, we anticipate to miss little improvement.

As we will demonstrated in Section 6.3, optimization objectives vary non-monotonically over the course of iterations. Thus we require the full number of iterations when aiming for the best quality possible.

On the other hand, this is not always the actual objective in chip design. Depending on the stage of the design process, sometimes placement algorithms are used to decide whether a given netlist can be placed routably or with specified slack. We can easily incorporate criteria to end our algorithm prematurely whenever a placement of sufficient quality is known.

5.3 Routability-Driven Placement

Besides minimizing netlength, our placer is able to improve the routability of a chip. The main idea is to dynamically reduce placement density wherever routing becomes difficult. We achieve lowered placement density by artificially increasing the size of cells in such sectors of the chip.

Our CongestionAvoidance routine (Algorithm 5.2) is called at the end of each iteration (Algorithm 5.1, line 13) of Self-Stabilizing BonnPlace. The result of this algorithm, namely new sizes for all cells, serves as input for the following iteration. Updated sizes of cells affect global density, i.e. inflated cells consume more capacity during flow-based partitioning. On the other hand, those size updates never change the actual physical outline of a cell.

CongestionAvoidance consist of two major components: congestion estimation (line 1) and density adjustment (lines 2 to 12). In the following we discuss both those steps.

Algorithm 5.2: CongestionAvoidance

Input : Grid G, cells C, locations pos(c) for $c \in C$, target/critical wire density tgt/crit.
Output: Adjusted size(c) for all $c \in C$.

```
1  Use BonnRouteGlobal as congestion estimation on G
2  while congested_edge_percentage < 10% and crit > tgt do
3      crit := max{tgt, crit − 5 %}
4  foreach tile window w ∈ G do
5      Compute criticality(w) w.r.t. crit
6  Choose a ∈ [0, 1]
7  foreach tile window w ∈ G do
8      foreach cell c ∈ w do
9          if criticality(w) > 0 then
10             size(c) := size(c) + a · inflate(c, w)
11         else if criticality(w) ≤ 0 then
12             size(c) := size(c) + shrink(c, w)
13 return size
```

5.3.1 Congestion Estimation

We use an actual global router for estimating congestion (Algorithm 5.2, line 1). Such an algorithm roughly decides how nets are routed, but avoids finding a routing that is actually legal w.r.t. various distance constraints. In particular it determines a prediction of the number of wires crossing certain areas of the chip. We measure congestion based on this global routing.

Note that during Self-Stabilizing BonnPlace (cf. Algorithm 5.1) CongestionAvoidance is always applied to a legal placement. A global routing engine works best on legal placements, as predicting congestion accurately is much harder otherwise. Thus also CongestionAvoidance benefits from the effort spent on legalizing the placement thoroughly.

The global router we rely on is BonnRouteGlobal (Müller, Radke and Vygen 2011). This tool considers a coarse three-dimensional grid that partitions the routing area on all layers. In this grid the global routing problem can be modeled as Min-Max Resource Sharing Problem where resource allocation (routing capacity along grid-edges) is balanced among customers (nets). Assuming individual nets can be routed near-optimally, this approach is proven to find near optimum solutions for the overall global routing problem very efficiently.

Moreover this algorithm is also very relevant in practice. In particular it is used in multiple stages of the IBM chip design process, most notably as first step of detailed routing (Ahrens et al. 2015; Gester et al. 2013).

For our congestion analysis we align the global routing grid to the placement grid. As a consequence BonnRouteGlobal estimates exactly the routing requirements caused by our partition of cells into windows. In order to limit running time, we avoid using the grid of the final placement level. Instead we use a coarser grid, more precisely of the

maximum level during which windows exceed a user-specified number of routing tracks.

Naturally we need a fast congestion estimation that can be computed frequently. For our purposes we also prefer a rough, pessimistic estimation over a more accurate and possibly optimistic one. This is due to the fact that we want to optimize our placement to be routable easily, rather than barely and with huge effort. Considering a pessimistic routing estimation more clearly shows where we need to improve the placement.

Thus we adjust standard BonnRouteGlobal in two ways for our purposes: On the one hand, we reduce the number of phases of the Resource Sharing Algorithm to a single one. On the other hand, we route each net with artificially restricted search space. Instead of leaving the freedom to route it arbitrarily (possibly with large detour to avoid congestion hotspots), we narrow the search space to the net's bounding box (plus few grid windows). Due to this, routing hotspots can be observed more clearly. Due to both choices, we lose theoretical guarantees but make this approach viable in practice.

5.3.2 Density Adjustments

With a congestion estimation at hand, Algorithm 5.2 adapts the density of the placement. This is done in two steps similar to Brenner and Rohe (2003): assigning each window of the grid a routing criticality (lines 2 to 5) and distributing inflations based on this criticality (lines 7 to 12). We now elaborate those steps.

In contrast to Brenner and Rohe (2003), CongestionAvoidance manages a dynamically adapting congestion target crit. We lower crit maximally towards tgt s.t. at least 10 % of the routing edges have a congestion of at least crit. Here tgt denotes a user-specified congestion target which is usually chosen around 80 % to 90 %.

Based on this congestion target crit and the actual estimated routing congestion, we determine routing edge criticalities. These edge criticalities in the interval $[0, 1]$ are derived by mapping congestion below crit to 0, above l to 1 and interpolating linearly in $[\mathsf{crit}, l]$. Here l denotes a user-specified congestion cutoff (usually 130 %).

Afterwards we designate $\mathsf{criticality(w)} \in [0, 1]$ to the grid windows (lines 4 to 5). These criticalities are chosen as the average of pin density in w and the edge criticalities of incident routing edges of w.

But inflations can not only be increased, but also reduced. In order to downsize cells in a window, all adjacent edges are required to have criticality 0. If this is the case, we similarly interpolate a routing edge uncriticality in $[-1, 0]$ where congestion crit is mapped to 0 and congestion 0 % to -1. Based on this, $\mathsf{criticality(w)} \in [-1, 0]$ is averaged as before. Reducing cell inflation is essential to recover space in uncongested sectors of the chip.

The distribution of inflations is performed in analogy to Brenner and Rohe (2003) (lines 7 to 12). At this point we have computed $\mathsf{criticality(w)} \in [-1, 1]$ for each window w. We then compute the area increment of w as

$$\mathsf{criticality(w)} \cdot \iota \cdot \sum_{c \in w} \mathrm{size}(c)$$

where ι is a user-specified inflation emphasis (usually 80 %). This area increment of w is then distributed to $c \in w$ proportionally to the number of pins of c. We thereby ensure that no cell is shrinked below its original size.

In order to ensure that the inflated cells can still be placed on the chip at all, we initially choose a maximum scalar $a \in [0, 1]$ for which this is possible. This scalar applies to all inflations and shrinkages of any cells.

Adapting the criticality threshold crit is essential. On the one hand, it results in cautiously computed inflations that targeted congestion hotspots precisely. On the other hand, we crucially aim at reducing congestion globally towards tgt, even below 100 %. Barely routable chips make detours inevitable, which has a negative effect on other design objectives. Thus it is necessary to optimize for maximum congestion tgt to make routing easier. Dynamically adapting our critical congestion crit is an important component for this. It is very effective in practice as we demonstrate in Section 6.4.2.

5.4 Timing-Aware Placement

The versatility of Self-Stabilizing BonnPlace is especially useful for optimizing timing characteristics (Algorithm 5.1, lines 8 to 11). Besides good routability, this is the most important objective during placement for modern technology nodes. Moreover optimizing this objective is required to evaluate timing-driven macro placements as obtained by BonnMacro (cf. Chapter 4).

Our framework applies two different methods to resolve timing problems: we optimize the assignment of nets to routing layers as well as locally move circuits purely based on timing properties.

Remember that we assume Self-Stabilizing BonnPlace is applied in a linearized timing model (cf. Section 2.2.2). Due to this timing optimization runs more efficiently.

As first stage of our internal timing optimization, we run BonnLayerAssignment (Algorithm 5.1, line 9). This module of the BonnTools suite specifically targets critical nets. It improves the delay of such nets by assigning those to higher layers and recommending wider wires. Both those changes lead to lowered resistance and thus improved timing.

Naturally especially critical nets compete for limited routing resources on the fastest layers. BonnLayerAssignment heuristically respects these routing capacities without having to compute an actual global routing. On the other hand, we can also operate this tool in an optimistic mode where such constraints are not applied. It makes sense to use this mode to obtain worst slack bounds e.g. for evaluating timing-driven macro placements.

As described in Section 2.2.2, the delay of long nets differs greatly between layers. Thus it is important to first assign long connections across the chip to the fastest layers. Only under this condition we can apply our local placement optimization based on timing.

For this purpose (Algorithm 5.1, line 10) we use another component of the BonnTools suite: BonnRefinePlace (Bock et al. 2015; Bock 2010). This tool relocates a predefined ratio of cells in the most critical nets to decrease the overall timing criticality. In this step, cells are moved such that critical paths are shortened and detours are avoided. This is achieved by a local search for timing optimal positions of single circuits. In order to overcome local minima, small clusters of neighboring critical cells can also be moved simultaneously. Typically this approach allows to shorten critical paths significantly in case it is sufficient to move only some of the cells.

The resulting positions of BonnRefinePlace are not necessarily legal. Using the aforementioned BonnPlaceLegalization (Brenner 2012) once again we can resolve all overlaps in the placement (Algorithm 5.1, line 11). As only a small fraction of cells was moved, legalization is efficient and causes little movement as well as degradation in timing.

These refined positions serve as force target points in the next iteration. Moreover we emphasize the importance of cells moved by BonnRefinePlace. To do so, we additionally increase the force weight by 50 % for those circuits. The raised force weights

persist in subsequent iterations until the standard force weight has caught up or the cell is moved once again.

With this weighting scheme, we achieve two contrary characteristics: On the one hand, our placer is strongly encouraged to retain similar positions for critical cells in the future. On the other hand, in subsequent iterations different cells might become the most timing critical. Thus our weight emphasis decays for cells having been formerly critical. We thereby can improve the timing characteristics of placement solutions in subsequent iterations.

Previous versions of Self-Stabilizing BonnPlace additionally updated net weights based on timing criticality (Brenner et al. 2015). This turned out to be harmful for modern technology nodes and thus is not done any longer.

All described steps for timing optimization change the placement or the layer assignment. Either way, these changes directly affect global routability. This is why it is more effective to estimate congestion at the end of an iteration in our placement scheme.

To optimize timing thoroughly, we usually perform 15 iterations of timing-driven Self-Stabilizing BonnPlace. Thereby we delay inclusion of BonnLayerAssignment to iteration 1 and BonnRefinePlace to 2. Gradually adding optimization steps helps in conjunction with CongestionAvoidance as particularly BonnLayerAssignment significantly changes global routability. Thus our placer benefits from the opportunity to adapt the placement accordingly.

For estimating macro placements, less iterations are sufficient. Usually we restrict ourselves to 5 iterations whereby we skip optimizing routability and run BonnLayerAssignment in the mode that disregards routability.

As a closing remark we want to point out the versatility of Self-Stabilizing BonnPlace. Incorporating further timing optimization techniques which deviate from previous placements even more significantly, e.g. logic optimization or latch clustering, seem promising for future evolutions of our framework.

Chapter 6

Experimental Results

This chapter is devoted to the performance of various placement algorithms that have previously been discussed. We examine our tools by experiments both on recent real-world instances from industry and on benchmark chips.

Section 6.1 provides an overview of all designs and the benchmarking environment. In Section 6.2 we revisit the running time effects of our improved region construction as discussed in Section 5.1. Afterwards we concentrate on Self-Stabilizing BonnPlace (cf. Sections 5.2 to 5.4): We start with an analysis of the basic mode in Section 6.3, before we discuss the congestion-aware mode in Section 6.4 and the joint timing-driven and congestion-aware variant in Section 6.5. Finally Section 6.6 analyzes our timing-driven extension of BonnMacro which has been proposed in Chapter 4.

6.1 Test Environment

An overview of all test instances considered in this chapter is presented in Table 6.1. All instances are real-world designs provided by our industrial partner IBM.

We test primarily on designs of the current technology nodes, but also include challenging older test cases. Moreover we consider designs with millions of cells, many macros and blockages as well as complicated movebounds.

Note that we consider multiple variants of one particular instance: Alex. Alex (v1) has extra movebounds arising from the design hierarchy and Alex (v2) contains many extra blockages.

For technical reasons not all instances provide both timing and routing environments, which is why it is impossible to analyze all approaches on the entire testbed. In the following chapters we consider subsets of the presented chips, for which the respective approaches seemed most promising.

Unless stated otherwise, all experiments have been conducted on an Intel Xeon E5-2699 v4 CPU which runs at 2.2 GHz base frequency. The Intel® Turbo Boost Technology is enabled with a maximum clock speed of 3.6 GHz. This processor has 22 physical cores providing 44 threads via hyper-threading. Our machine is equipped with 378 GB of main memory.

Our code was compiled using clang version 7.0.0 on CentOS release 7.6 using optimization level `O3`. All mixed-integer and linear programs are solved using IBM ILOG CPLEX Optimization Studio 12.6 (CPLEX).

Chip	T [nm]	C [10^3]	L	P [10^3]	N [10^3]	M	B	MD [%]	DB [10^3]	R	CA [mm^2]
Anna	14	2	800	203	76	5	24	83	10	15	2.5×2.5
Rosemarie	14	7	941	359	140		9	58	9	13	6.5×4.6
Ida	22	19		67	19		2		1	7	0.2×0.2
Leo	22	31	9	94	34		4	66	1	12	0.8×0.9
Beate	22	41		141	43		10		1	7	0.3×0.3
Sonja	14	53	5	170	55			10	3	11	0.3×0.4
Anja	14	66	244	86	37			56	3	15	2.0×4.1
August	14	73	18	176	67	6		26	3	7	0.7×0.4
Rolf	7	75		222	72				1	8	0.2×0.3
Jannik	14	126	21	360	126	4	4	54	43	13	0.7×1.7
Conni	14	134	99	494	164			66	275	11	1.1×1.3
Falk	14	169		488	167				1	13	0.3×0.3
Lisa	14	171	294	271	121		4	70	2	15	1.5×2.4
Benedikt	22	264	37	837	267		2	29	3	13	0.5×2.1
Renaud	45	352	33	1 132	352		4	33	1	7	1.0×1.1
Viktor	22	382	142	1 551	444	6	5				2.4×3.9
Franziska	22	386	20	1 336	412		2	21	686	13	1.3×1.7
Meinolf	22	393	1	1 494	437		2	1	2	7	1.0×0.8
Iris	22	430	11	1 577	474		2	9	1 224	7	1.0×0.9
Gautier	45	1 362	2 926	5 136	1 478		4	29	14 101	8	4.0×6.0
Regina	14	1 385	37	4 266	1 495	1	2	22	3 384	15	1.9×1.4
Mia	14	1 499	35	4 625	1 625	84		20	3 710	15	1.8×1.5
Valentin	14	1 585	265	4 912	1 693	57		41	4 077	15	2.2×1.7
Rafael	32	1 946	7 008	5 983	1 920		1 982	51	8 487	11	3.2×2.7
Elias	14	5 132	372	16 262	5 408		24	23	19 395	15	2.5×2.5
Alex	14	8 484	3 262	24 906	8 955		380	35	16 222	15	6.1×4.6
Alex (v1)	14	8 484	3 262	24 906	8 955	361	380	35	16 222	15	6.1×4.6
Alex (v2)	14	8 484	3 262	24 906	8 955		301K	56	16 222	15	6.1×4.6

Table 6.1: Overview of our testbed: For each chip the technology T in [nm], the number of cells C, non-standard cells L, pins P, nets N, movebounds M, blockages B, distance bounds DB and routing layers R. Moreover the fraction MD of the chip area covered by macros in [%] as well as the chip area CA in [mm^2].

6.2 Partitioning with Movebounds

We first examine the impact of different region construction strategies for BonnPlaceGlobal as discussed in Section 5.1. We compare different approaches during Global Placement where all macros have fixed positions. In this context we do not consider self-stabilizing yet, i.e. perform only a single iteration.

We compare three different approaches: The baseline has been presented in Lemma 5.4. In contrast to the original implementation of Struzyna (2011) it has been improved to compute the (strongly) movebound homogeneous partition in polynomial time using Proposition 5.3.

An improved version of this strategy is referenced as M-homogeneous. It computes the movebound homogeneous partition similarly, but in contrast only introduces a single region for each element of this partition. Moreover it manages blockages more efficiently by exploiting the grid structure of windows using Lemma 5.12.

The final approach relies on a weakly M-homogeneous partition of windows. This has been proposed in Proposition 5.11 and includes the same blockage handling mechanism as for the M-homogeneous case.

In a first analysis we reconsider the instance Mia which has already been visualized multiple times before (cf. Figures 5.1 to 5.5, pages 79 and 85). This instance is of particular interest as it has the most challenging movebound structure. Figure 5.2 highlights the intricately overlapping movebound configurations.

The mentioned approaches on this particular instance are compared in Figure 6.1. We analyze different properties of all three mentioned approaches in the respective levels of BonnPlaceGlobal. Since level 0 consists of a single, very efficient QP computation only (without partitioning), the shown analysis starts with level 1.

On the left we see the number of introduced regions. The exponential increase in regions is intrinsic to recursive partitioning and particularly underlined due to the logarithmic scaling. The right of Figure 6.1 visualizes the total wall clock time spent in the respective levels for each strategy.

During the first levels of placement we see a drastic reduction in number of regions. The running time improvement obtained by considering fewer regions is even more obvious. The running time spent in level 2 in the baseline was 2:14 h, which could be reduced to 29 min and 15 min with the strongly and weakly movebound homogeneous approaches. The substantial running time improvements are very much expected, since the number of regions super-linearly contributes to the running time complexity of solving min-cost flow problems in partitioning.

In the last levels of BonnPlaceGlobal the difference is marginal for two main reasons: For once windows become very small and thus intersect few movebounds. Consequently the constructed regions of all approaches are more and more similar until they coincide for all approaches in the final level. On the other hand, our placer processes multiple windows in parallel. This parallelization works best if there are sufficiently many windows and compensates for marginal improvements on single, small instances.

A comparison on a larger testbed with more chips is presented in Table 6.2. All instances are ordered by increasing number of cells. For each chip we compare the same region construction variants baseline (base), M-homogeneous (hom) and weakly M-homogeneous (whom) as in Figure 6.1 in separate rows.

In the columns of Table 6.2 we show the total wall clock time of BonnPlaceGlobal (BPG WT), the wall clock time contribution of computing regions (BPG RC) as well as

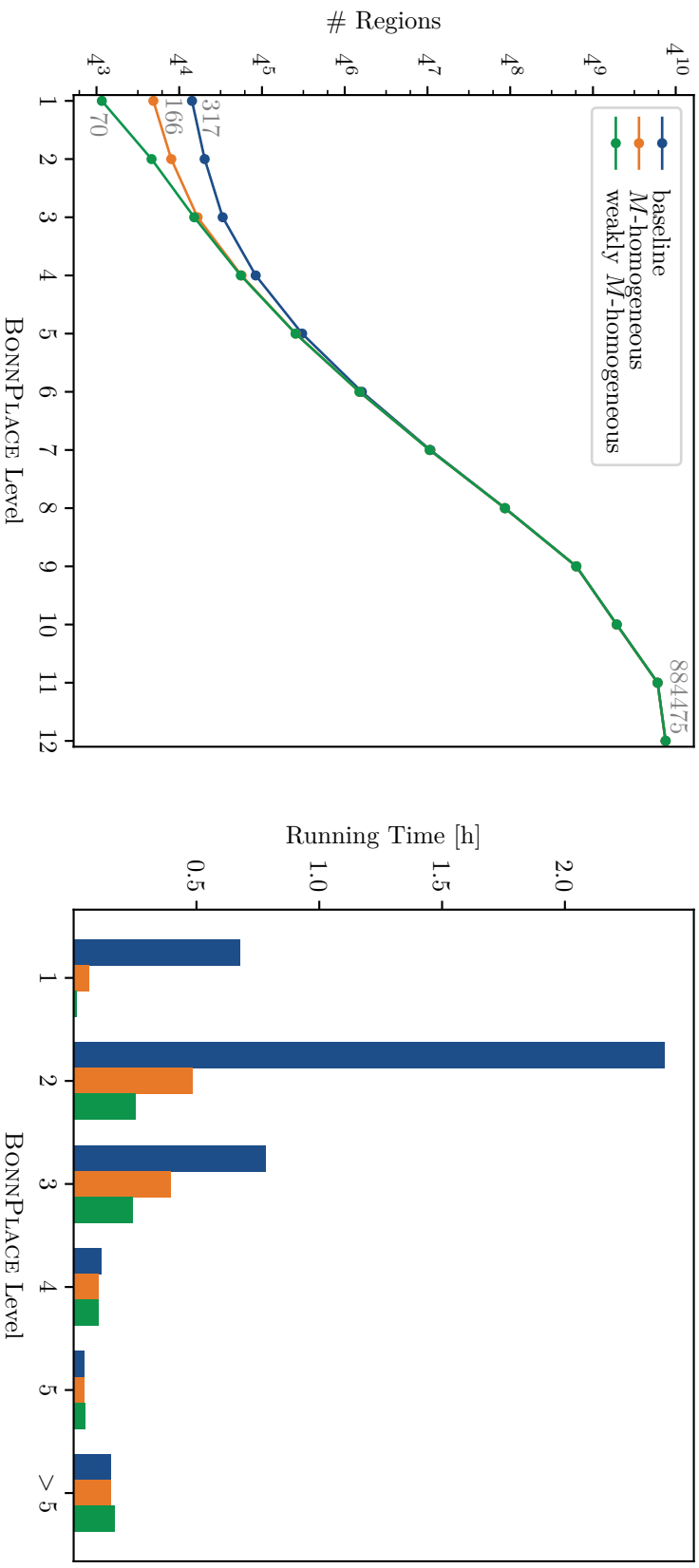


Figure 6.1: Details on Mia: We compare BonnPlaceGlobal with different regions. The left picture shows the numbers of regions in each level while we display the resulting running times on the right. Colors match in both visualizations.

Chip	Run	BPG WT [H:M:S]	BPG WT [%]	BPG RC [M:S]	BPG RC [%]	Netlength [m]	Netlength [%]
Ida	base	12		0		0.34	
	hom	12	+0.76	0	−36.54	0.34	+0.13
	whom	13	+8.54	0	−32.69	0.34	−0.02
Leo	base	16		0		1.81	
	hom	17	+5.52	0	−2.20	1.80	−0.46
	whom	19	+16.87	0	−17.86	1.80	−0.60
Beate	base	20		0		0.51	
	hom	22	+6.70	0	−30.00	0.51	−0.42
	whom	22	+10.11	0	−34.44	0.51	−0.07
Sonja	base	29		0		0.92	
	hom	30	+3.29	0	−28.35	0.92	+0.12
	whom	32	+10.64	0	−27.10	0.92	+0.08
Jannik	base	01:31		3		4.60	
	hom	01:27	−4.40	2	−4.61	4.61	+0.34
	whom	01:32	+1.28	2	−15.38	4.63	+0.59
Lisa	base	01:27		44		14.91	
	hom	54	−38.03	8	−81.18	14.89	−0.11
	whom	51	−40.55	6	−85.55	14.88	−0.20
Benedikt	base	02:53		0		7.29	
	hom	02:56	+1.86	0	−21.93	7.31	+0.25
	whom	03:00	+4.12	0	−21.49	7.28	−0.13
Renaud	base	05:36		0		7.41	
	hom	04:33	−18.68	0	−27.01	7.47	+0.90
	whom	04:37	−17.51	0	−28.07	7.48	+1.04
Viktor	base	05:54		9		20.49	
	hom	06:06	+3.35	5	−45.50	20.50	+0.07
	whom	05:47	−1.89	5	−46.03	20.47	−0.09
Gautier	base	11:53		7		236.90	
	hom	11:42	−1.49	4	−44.62	236.91	0.00
	whom	13:02	+9.76	4	−38.45	237.04	+0.06
Mia	base	04:11:51		13		27.22	
	hom	01:15:40	−69.96	8	−35.94	27.83	+2.25
	whom	50:43	−79.86	10	−21.91	27.20	−0.04
Rafael	base	21:28		21		133.87	
	hom	21:24	−0.24	9	−53.20	133.99	+0.09
	whom	22:38	+5.50	10	−49.93	134.13	+0.20
Alex	base	02:40:50		20:52		205.75	
	hom	02:20:07	−12.88	01:03	−94.96	205.99	+0.12
	whom	01:39:38	−38.05	01:07	−94.65	205.76	+0.01
Alex (v1)	base	12:42:21		11:20		204.51	
	hom	02:38:40	−79.19	01:50	−83.76	204.20	−0.15
	whom	01:56:32	−84.71	01:23	−87.78	204.15	−0.18
Alex (v2)	base	03:10:20		51:25		209.51	
	hom	02:19:40	−26.62	01:20	−97.40	209.30	−0.10
	whom	01:39:49	−47.56	01:15	−97.55	209.18	−0.16
Summary	hom		−22.16		−61.47		+0.20
	whom		−26.91		−62.68		+0.03

Table 6.2: Region construction strategies during BonnPlaceGlobal. We compare three runs: a baseline run (base) against one with homogeneous (hom) as well as with weakly homogeneous (whom) regions. Columns present

- BPG WT: Total wall clock time of BonnPlaceGlobal
- BPG RC: Contained wall clock time of region construction
- WL: Final bounding box wirelength in [m]

All metrics are compared in [%] against the baseline. Improvements in green, marginal deviations in blue, degradations in red. Summaries as geometric mean of the ratios of the deviations.

the final bounding box netlength (Netlength). The comparison columns in [%] indicate the changes relative to the baseline version. We finally summarize these deviations from the baseline as the geometric mean of the ratios.

On small instances the baseline version is fastest overall (BPG WT). But at the same time the absolute increase is only a few seconds and thus minor overall. Moreover wall clock time is not a deterministic metric. It usually fluctuates around 5 % by itself especially for a highly parallel application like BonnPlaceGlobal.

On larger instances the total wall clock time significantly improves. The simplification by M-homogeneous regions defines an important improvement that is even surpassed by considering weakly M-homogeneous regions. Running time is most drastically improved on instances with many movebounds, e.g. Mia or Alex (v1). This is expected since we specifically targeted those kinds of instances. Here our new region variants dramatically speed up flow-based partitioning as we already demonstrated in Figure 6.1.

But not only the total wall clock time improves due to faster flow-based partitioning. Also the total wall clock time to construct all regions in all levels combined decreases. As the BRG RC column indicates, the tendency is very clear: Both improved region construction versions result in significant and similar speed-ups. This can almost entirely be attributed to Lemma 5.12 using which quad-trees can be avoided by exploiting the windows' grid structure. This effect can be observed most explicitly on instances with large chip area, e.g. Rafael or Alex (v2).

Netlength on the other hand marginally increases. In general we expect a netlength increase since partitioning is modeled with fewer regions, which is why movement costs are expressed less accurately. At the same time netlength often fluctuates up to 0.5 % on single instances, e.g. with slightly different placement parameters. The netlength penalty of our proposed regions thus is well below a usual variation overall.

Overall weakly M-homogeneous regions provide a very significant running time improvement that outweighs marginal netlength increase.

6.3 Self-Stabilizing Behavior

In this section we analyze the performance of the basic Self-Stabilizing BonnPlace framework. For that reason we do not yet enable congestion mitigation or timing optimization techniques. This basic version of our placement algorithm has been introduced in Section 5.2.

In Figure 6.2, we demonstrate the self-stabilizing behavior of BonnPlace on four selected instances (Falk, Renaud, Meinolf as well as Iris; cf. Table 6.1).

On the left, we present two netlength metrics for each design over the course of the first 5 self-stabilizing iterations. The lower curve specifies the bounding box netlength of our analytical QP placement. The same netlength metric is evaluated on the legal placement determined by BonnPlace starting with the respective QP solution as first analytical placement. This is visualized by the upper curve. Note that all absolute values have been normalized to the final, legal bounding box netlength in order to simplify the comparison.

For each chip, Figure 6.2 also presents a different metric in the right part. Here we visualize the total (linear) cell movement from the analytical placements to the final placements in each iteration.

We can clearly observe that the final netlength can be steadily improved during all iterations of Self-Stabilizing BonnPlace. The netlength reduction of up to 5.5 % is very significant.

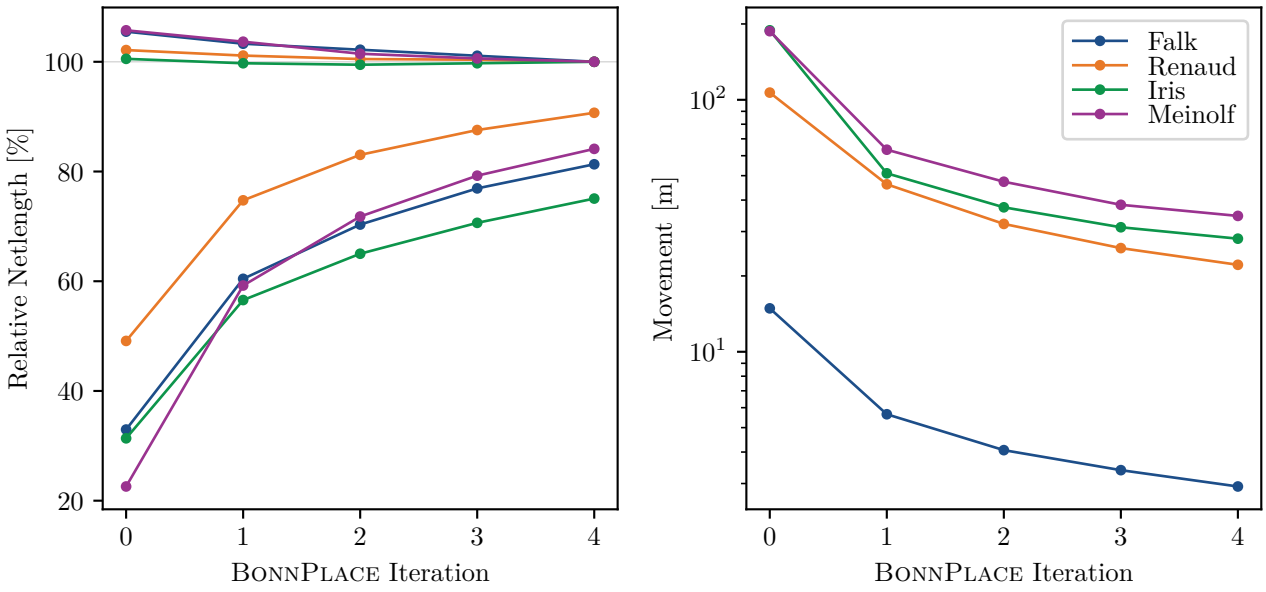


Figure 6.2: Netlength and movement during Self-Stabilizing BonnPlace: The left figure shows QP (lower curve) and legal netlength (upper curve) normalized to the final netlengths on each chip. On the right, we display total movement resulting from legalizing QP positions. Chip colors match in both figures.

This is achieved via additional spreading in the analytical placement. We observe this spreading as increasing netlength of the QP solution. This additional information can subsequently be utilized by flow-based partitioning and results in the netlength improvements overall.

Note that in contrast to SimPL-based approaches (Kim, Lee and Markov 2012, 2013) our analytical placement is not necessarily a lower bound for the final netlength. Since we optimize *quadratic* netlength here, the netlength of the legal placement can be smaller than the one of the illegal QP.

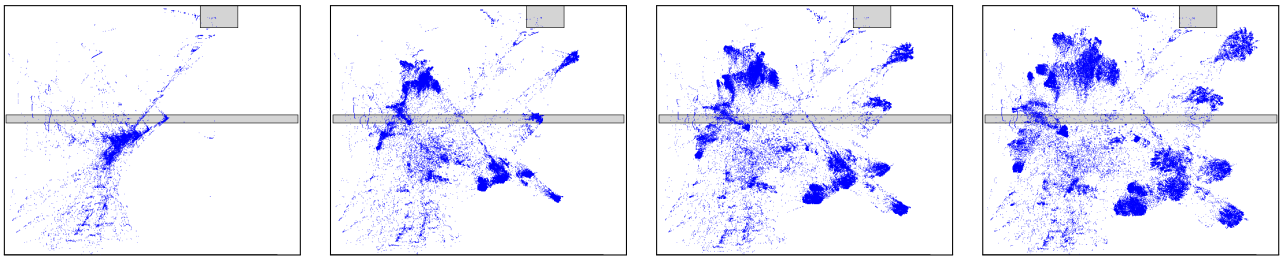
Moreover our analytical placement with forces results in a particularly good form of spreading. Over the course of iterations, it predicts the legal placement better and better. This can be seen in the drastic and steady movement decreases throughout Self-Stabilizing BonnPlace on the right of Figure 6.2. Note that the scale is logarithmic here.

This effect is also demonstrated visually in the QP solutions presented in Figure 5.6 (page 89) on Renaud, Figure 5.7 (page 90) on Franziska as well as Figure 6.3 on Meinolf. The presented analysis justifies the name of Self-Stabilizing BonnPlace.

6.4 Congestion-Driven Standard Cell Placement

Now we analyze the congestion-driven mode of Self-Stabilizing BonnPlace. For this purpose we only enable congestion mitigation and turn timing optimization off.

We consider results on industrial designs in Section 6.4.1 before we address the 2012 Design Automation Conference (DAC) placement contest benchmark suite (Viswanathan et al. 2012) in Section 6.4.2.



(a) Iteration 0 **(b)** Iteration 1 **(c)** Iteration 2 **(d)** Iteration 4

Figure 6.3: Initial QP placements on Meinolf in the first three and the final iterations.

6.4.1 Industrial Designs

The first comparison is presented in Figure 6.4. Here we analyze our placer in the first 8 self-stabilizing iterations. All placements have been computed with identical default parameters for congestion mitigation, i.e. target cell-density of 75 %, a congestion target wACE4 of 90 % and maximum inflation of 80 % per grid window and iteration. For all five industrial designs tested on, we know that finding a routable placement is not easy.

The left part is similar to Figure 6.2: For each design, we have two curves. The lower curve again denotes the bounding box netlength of our initial QP solutions. In contrast to that, the upper curve specifies the routed netlength as determined by our global router at the end of the respective iterations. In order to compare different chips, all absolute values are scaled to the final (routed) netlength.

On the right, we present how the congestion evolves over the course of iterations. This is measured in terms of wACE4 [%]. The **wiring-area average congestion estimate wACE4** is a standard congestion metric which was first proposed by Wei et al. (2012). It is defined as

$$\text{wACE4} \;:=\; \frac{1}{4} \cdot \sum_{p \,\in\, \{0.5,\,1,\,2,\,5\}} \text{ACE}(p)$$

where $\text{ACE}(p)$ is the average congestion estimate on the p % of the most congested global routing edges weighted by wiring usage.

As a first observation, we clearly see that routability significantly improved during Self-Stabilizing BonnPlace. We observe that the wACE4 decreases quickly and is already below 100 % on all designs after only 4 iterations. On a single instance (Iris) wACE4 increases in intermediate iterations. This is an effect of placing too many circuits in small alleys between macros, which we can effectively avoid in a next iteration. On all other instances BonnPlace continues to improve wACE4 steadily in further iterations.

Recall that our congestion mitigation technique is applied as a final step at the end of each iteration. Thus the initial high wACE4 values in iteration 0 are expected, since during placement in this iteration no inflations have been applied yet. From iteration 1 onward we thus observe that our congestion mitigation approach effectively targets wACE4.

The behavior of Self-Stabilizing BonnPlace concerning netlength changes in congestion-driven mode (cf. Figure 6.2). On some instances the final (in this case routed) netlength increases while wACE4 is reduced significantly (e.g. Rolf or Iris). This is in an effect of our congestion mitigation technique via artificial cell inflations that reduces cell density and thus can require longer nets.

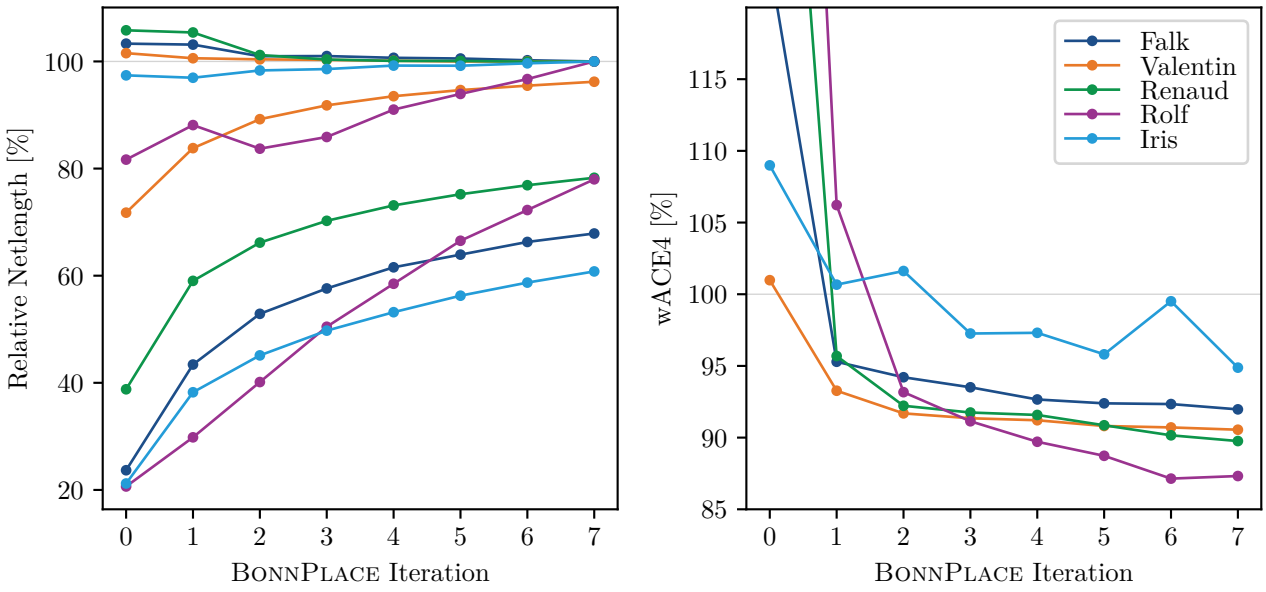


Figure 6.4: Netlength and wACE4 development during congestion-driven Self-Stabilizing BonnPlace: The left figure shows QP (lower curve) and routed netlength (upper curve) normalized to the final netlengths on each chip. On the right, we display the congestion. Chip colors match in both figures.

Interestingly this behavior can not be observed on all chips. On all other designs, netlength even decreases while cells are being artificially spread in order to avoid congestion. This indicates that we apply cell inflations very cautiously and we often are able to compensate for this type of extra spreading with regard to routed netlength. This can also be verified, as the scaling factor a of the CongestionAvoidance routine (Algorithm 5.2, line 6 on page 92) is always at $a = 1.0$.

Note that Figure 6.4 is normalized to routed netlength, while Figure 6.2 in contrast considers bounding box netlength as baseline. This is also the reason why seemingly the spreading of our QP diminishes e.g. on Iris. But this is actually only a matter of normalization. In fact our additional cell spreading is also reflected in our initial analytical placements (e.g. 5.87 m with compared to 5.60 m without congestion mitigation on Iris in iteration 4).

The behavior of our congestion mitigation technique on one particular design is also visualized in Figure 6.6. Here we see the routing congestion on Renaud at the end of the first two and the last iterations of Self-Stabilizing BonnPlace. The global routing edges are colored according to Figure 6.5.

Apparently our placer can quickly dissolve the initial hotspots indicated in purple and red (cf. Figure 6.5). Consequently the significantly improved congestion at the end of iteration 1 is a result of a single application of CongestionAvoidance at the end of iteration 0. As we can see, routability is further improved until a wACE4 of 89.76 % is reached in iteration 7.

Note that BonnPlace does not operate on the congestion analysis presented in Figure 6.6. As discussed in Section 5.3.1 (page 92), we rather only work with a rough estimate that is faster to compute and indicates routing hotspots more clearly. Figure 6.6 in contrast shows a detailed congestion analysis of a complete global routing. This

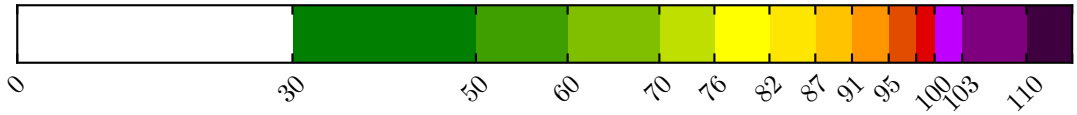


Figure 6.5: Legend for congestion visualizations; values in [%].

estimation is trustworthy and very accurate, but has only been computed for reasons of comparison.

Even though this very instance Renaud is from the former 45 nm-node only, it has been a challenging test case for BonnPlace for many years. Previous versions of BonnPlace (Struzyna 2013) have not been capable of finding any placement which was even close to routable. But with the proposed enhancements in Self-Stabilizing BonnPlace we can solve this difficult instance well.

We demonstrated that BonnPlace is capable of effectively mitigating congestion on real-world industrial designs.

6.4.2 DAC 2012 Placement Contest

In this section we compare Self-Stabilizing BonnPlace to other state-of-the-art placement tools on the 2012 Design Automation Conference (DAC) placement contest benchmark suite (Viswanathan et al. 2012).

All our placements are finally routed by the official contest router NCTUgr (Liu et al. 2010; Liu, Li and Koh 2012) in regular mode with default parameters. Despite using NCTUgr for a final estimation, we continue to consult BonnRouteGlobal (Müller, Radke and Vygen 2011) throughout our algorithm (cf. Section 5.3.1, page 92).

We compare the official contest metric, i.e. the **scaled wirelength sWL** defined as

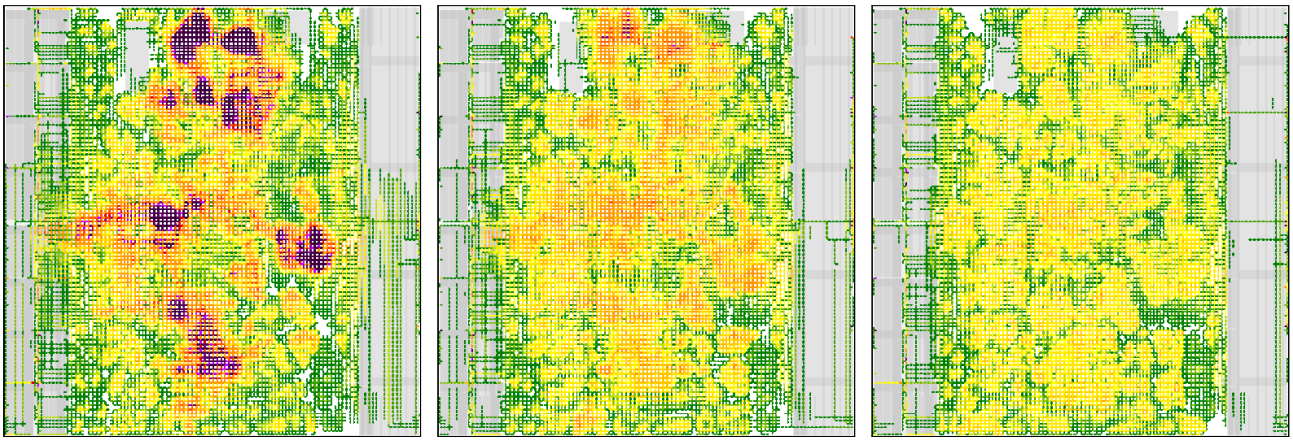
$$\mathrm{sWL} := \mathrm{HPWL} \cdot \left(1 + \frac{3}{100}(\mathrm{RC} - 100)\right),$$

where $\mathrm{RC} := \max\{100, \mathrm{wACE4}\}$ denotes the routing congestion. HPWL stands for the (unscaled) linear half-perimeter wiring length. We report the (scaled) wirelengths and routing congestion as determined by the official benchmark evaluation scripts.

All benchmarks were placed with identical parameter settings. We ran our placer with a target density of 90 % for four iterations. After each iteration, we updated our inflations limiting the cell size increment to 50 % of the original cells' size for each grid window individually.

Only for these benchmarks, we consider the bound-to-bound net model (B2B) with a super-linear distance scaling factor for our analytical placements (cf. Struzyna 2013). This setup proved beneficial for these DAC benchmark designs, while on real-world instances the additional spreading due to quadratic netlength minimization is favorable – especially for timing optimization. As of iteration one, we use the current placement to determine the outer pins of nets for the B2B net model. Thus we do not have to run multiple QPs with B2B in order to stabilize the solution.

The results are summarized in Table 6.3. We compare ourselves to NTUPlace4 (Hsu et al. 2011), the best participant of the contest, and to the tool of Cong et al. (2013), which achieved the best results on these benchmarks so far. We report the scaled wirelength (sWL) both as an absolute value and relative in [%] to the scaled wirelength produced by our Self-Stabilizing BonnPlace. Moreover for NTUPlace4 and our tool, we report



(a) Iteration 0 **(b)** Iteration 1 **(c)** Iteration 7

Figure 6.6: Congestion on Renaud in the first two and the last iterations. Coloring according to Figure 6.5.

the routing congestion (RC). Note that the routing congestion of Cong et al. (2013) has not been published.

On average NTUPlace4 produces 4.04 % more scaled wirelength than our tool while the algorithm of Cong et al. (2013) produces 3.59 % more scaled wirelength. We get the best published results on six out of the ten benchmark designs. Moreover we reduce the routing congestion RC on all chips attaining the minimum possible in eight out of ten cases.

Our congestion mitigation technique is indeed very cautious; the scaling factor a (cf. Section 5.3) is consistently at $a = 1.0$. We inflate the movable cells' area by 30.72 % on average. This increases HPWL by 3.84 % on average over the course of our iterations. In contrast to our experiments in Section 6.4, we here start with a higher target density, which is why a netlength increase due to congestion mitigation is expected. Despite of this, we are able to rapidly accomplish a drastically improved routability and hence optimize this essential placement objective well.

It is not trivial to compare running times properly. The official benchmark was performed on 64 −bit Intel Xeon CPU X7560 running at 2.27 GHz in single threaded mode. Our machine for these experiments is about 45 % faster and our placer is implemented to run highly parallel. In this setting, the former best placer by Cong et al. (2013) needed on average more than 225 % of our running time on eight threads. Our running time includes the invocations of BonnRouteGlobal.

6.5 Timing-Driven Standard Cell Placement

Now we analyze the combined congestion-driven and timing-driven mode of Self-Stabilizing BonnPlace. In contrast to earlier versions of our placer (Brenner et al. 2015), we will target both objectives at the same time.

Our timing-aware global placer introduced in Section 5.4 was run with default parameters for timing optimization. These settings include the same congestion mitigation parameters as already discussed in Section 6.4. Furthermore we run 15 iterations in total, where BonnLayerAssignment is applied from iteration 1 onward and BonnRefinePlace is used starting with iteration 2.

Chip	Self-Stabilizing BonnPlace (Brenner et al. 2015)			Cong et al. (2013)		NTUPlace4 (Hsu et al. 2011)		
	sWL [$\times 10^8$]	RC [%]	Inf [%]	sWL [$\times 10^8$]	[%]	sWL [$\times 10^8$]	[%]	RC [%]
sb2	6.05	100.17	46.42	6.14	+1.49	6.24	+3.14	100.68
sb3	3.22	100.00	58.03	3.60	+11.64	3.62	+12.26	103.53
sb6	3.37	100.00	31.77	3.40	+0.81	3.42	+1.54	101.21
sb7	4.07	100.00	19.23	3.95	−2.96	3.99	−2.10	100.68
sb9	2.35	100.00	26.51	2.50	+6.31	2.55	+8.25	102.48
sb11	3.44	100.02	15.97	3.40	−1.25	3.42	−0.62	100.02
sb12	2.80	100.01	41.57	3.04	+8.38	3.12	+11.21	100.02
sb14	2.26	100.00	15.53	2.45	+8.32	2.26	−0.26	100.07
sb16	2.65	100.00	31.85	2.74	+3.38	2.80	+5.79	102.39
sb19	1.51	100.00	20.30	1.51	−0.21	1.53	+1.18	100.61
Avg		100.02	30.72		+3.59		+4.04	101.17

Table 6.3: Results of congestion-driven Self-Stabilizing BonnPlace on the DAC 2012 placement contest benchmark suite (Viswanathan et al. 2012). We present the following metrics:

- sWL: Scaled wirelength (official contest metric)
- RC: Routing congestion
- Inf: Artificial cell inflation by our algorithm

All metrics are compared in [%] against BonnPlace. Improvements in green, marginal deviations in blue, degradations in red. Averages are the geometric means of the ratios of the deviations.

In Table 6.4 we illustrate results obtained by Self-Stabilizing BonnPlace with this configuration. We analyze four real-world designs from recent and older technology nodes.

To evaluate our approach we list multiple relevant metrics for each iteration: To begin with, we specify the bounding box netlength NL in [m] as well as the routing congestion C measured in terms of wACE4 in [%]. For timing criteria we analyze worst slack WS in [ps] and the path-based FOM variant (abbreviated as FOM in Table 6.4) measured in [ps]. FOM is measured with respect to a slack target of 5 ps. All metrics were gathered at the end of the respective iteration, after timing optimization on a legal placement.

The results show that over the course of iterations, we are able to improve timing behavior significantly. From iteration 0 to 14 our placement flow reduces FOM by 97 % on average over the four chips. Using iteration 1 and 2 as baseline, we still improve FOM by 93 % and 72 % respectively. Also the worst slack SLK is considerably improved across all designs.

If we take a closer look at the results, we can justify the configuration of our flow: After iteration 0 we often face significant routing congestion, e.g. on Rolf or Falk. Similar to the behavior analyzed in Section 6.4.1, considering one extra iteration with our congestion mitigation already significantly improves routability.

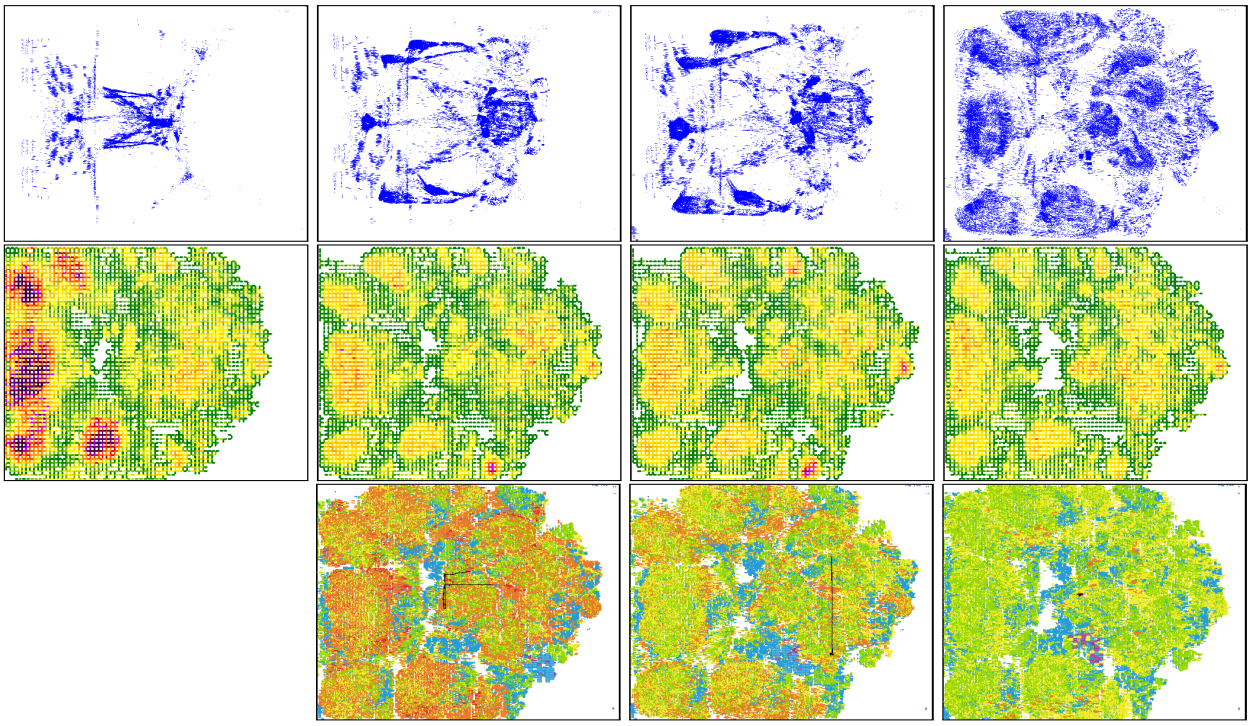
At the end of iteration 1 we perform BonnLayerAssignment for the first time.

It	August				Rolf				Falk				Benedikt			
	NL [m]	C [%]	SLK [ps]	FOM [ns]	NL [m]	C [%]	SLK [ps]	FOM [ps]	NL [m]	C [%]	SLK [ps]	FOM [ns]	NL [m]	C [%]	SLK [ps]	FOM [ns]
0	1.47	88.68	−368	−487	0.74	211.60	−119	−71 532	1.34	122.51	−126	−2 127	9.06	90.88	−1 090	−7 607
1	1.47	89.57	−257	−195	0.77	115.78	−116	−224 370	1.26	94.04	−42	−488	9.00	94.12	−470	−1 254
2	1.49	89.22	−253	−179	0.80	117.30	−18	−4 579	1.32	94.33	−20	−174	9.05	91.03	−315	−787
3	1.48	90.55	−309	−201	0.83	110.29	−18	−2 852	1.30	93.09	−16	−127	9.00	89.82	−317	−830
4	1.48	90.74	−269	−194	0.83	102.38	−11	−2 463	1.29	91.58	−18	−122	8.94	89.23	−337	−856
5	1.48	91.01	−284	−189	0.83	99.35	−16	−6 272	1.28	90.83	−17	−115	8.94	90.88	−341	−902
6	1.48	90.44	−263	−182	0.83	98.30	−12	−4 400	1.26	92.07	−16	−98	8.91	91.37	−337	−903
7	1.47	90.36	−248	−179	0.84	98.49	−11	−2 301	1.25	92.20	−15	−86	8.89	92.78	−345	−927
8	1.47	90.49	−259	−183	0.81	96.32	−8	−2 147	1.25	92.31	−15	−55	8.87	89.67	−361	−964
9	1.47	91.32	−246	−175	0.79	95.16	−8	−2 195	1.24	90.74	−13	−62	8.85	91.08	−309	−805
10	1.47	90.55	−245	−171	0.78	94.24	−12	−1 835	1.25	93.82	−12	−46	8.84	91.37	−297	−761
11	1.48	90.45	−250	−180	0.77	94.33	−7	−1 145	1.24	92.41	−13	−41	8.83	89.89	−299	−808
12	1.48	90.19	−252	−177	0.76	93.93	−9	−1 319	1.24	92.69	−13	−36	8.82	90.35	−292	−754
13	1.48	90.01	−258	−178	0.76	93.62	−6	−192	1.24	92.40	−11	−33	8.83	91.15	−326	−870
14	1.48	89.97	−262	−181	0.76	93.41	−5	−165	1.23	92.65	−11	−30	8.81	90.70	−298	−728

Table 6.4: Timing-driven Self-Stabilizing BonnPlace overview. We present the following metrics on August, Rolf, Falk and Benedikt:

- NL: Routed netlength in [m].
- C: Routing congestion wACE4 in [%].
- SLK: Worst slack in [ps].
- FOM: Path-based figure of merit in [ps], large values in [ns].

All values are measured based on the legal placement at the end of each of the 15 iterations (It).



(a) Iteration 0 **(b)** Iteration 1 **(c)** Iteration 2 **(d)** Iteration 14

Figure 6.7: Timing-Driven Self-Stabilizing BonnPlace overview on Falk: We present the first three and the final iterations in the columns. The rows show: QP positions at the start of the iteration, congestion at the end of the iteration (coloring according to Figure 6.5) and cells in the legal placement colored by slack (most timing critical path highlighted).

This assignment of critical nets to fast layers significantly improves worst slack and FOM on all instances apart from Rolf. But on this instance timing degradations are only a temporary effect of our congestion mitigation, that we can shortly compensate. Starting with iteration 2 we also incorporate BonnRefinePlace. Thus we especially improve the worst interconnections. This can be observed in terms of worst slack for example on Falk and Benedikt.

We also clearly observe that we target a wACE4 of 90 % as described in Section 5.3.2 (page 93). Once we reach this goal, we do not reduce cell density further but at the same time can preserve this wACE4.

Despite various additional steps in our timing-driven mode of Self-Stabilizing BonnPlace, we more or less preserve the behavior visualized in Figure 6.4 with respect to routed netlength and wACE4. Particularly on Rolf where netlength increases in intermediate iterations, we can see that our balancing of opposing placement objectives works well.

Finally note that the version of Benedikt considered in Table 6.4 differs from the one of Brenner et al. (2015). In the form at hand, Benedikt contains complicated paths whose gate delay is more than 120 % of the cycle time. Thus clearly negative slack can not be avoided here, even if all nets would have length 0. The variant of Brenner et al. (2015) re-optimized this logic to an extent where a worst slack of 0 ps was achievable.

The development of key metrics is exemplary depicted graphically in Figure 6.7 on

Falk. We present visualizations of iterations 0, 1, 2 and 14 in the columns. In the first and second row we show the initial analytical placement and the congestion of the respective iteration (as e.g. in Figures 6.3 and 6.6).

In the last row, we show our final placement of each iteration where cells are colored by slack and a critical path is highlighted in black. The coloring of slack is relative to the cycle time: More than 10 % slack compared to the cycle time is drawn in blue, positive slack in green, less than 10 % negative slack relative to the cycle time is drawn in yellow, up to 25 % in orange and below this in purple. In each such visualization, we also add a worst path where respective endpoints are connected by thin black lines. Note that our timing optimizations start at the end of iteration 1, which is why we omit timing statistics of the first iteration.

During the first two iterations we observe the behavior already discussed in Sections 6.3 and 6.4.1 what QP spreading and congestion is concerned. In particular the vast majority of critical routing problems can be resolved. Recall that we do not yet use BonnRefinePlace at the end of iteration 1 which is why the critical path contains significant detours.

At the end of iteration 2 we can observe two effects: On the one hand, new small routing hotspots arise (e.g. in the upper center). This is an effect of layer-assignment which we can resolve over the course of subsequent iterations. On the other hand, timing characteristics of this placement apparently are much better. This can be observed globally (fewer orange cells) as well as locally with a straightened, shorter worst path.

After the final iteration almost all cells reach the slack target, or at least almost the slack target. Cells on the most critical path are placed so closely together that it is even hard to spot this path connecting few orange boxes in the center of Falk. Note that the purple boxes are clock buffers (LCBs) who only have incident clock nets hidden from the timing environment. We achieve this placement while meeting the congestion targets.

We demonstrated that our Self-Stabilizing BonnPlace framework can balance different placement objectives and is capable of finding routable placements with good timing characteristics on real-world designs.

6.6 Mixed-Size Placement

Now we present practical results of timing-driven BonnMacro. For this purpose we analyze the quality of results in Section 6.6.1 before we specifically devote Section 6.6.2 to running times both of MIP solving and the proposed overall timing-driven BonnMacro flow.

6.6.1 Timing Results

In this section we analyze our timing-driven mixed-size placement approach. For this purpose we particularly consider real-world instances where macro placement is relevant for timing (cf. Table 6.1).

We consider all fairly large cells as macros. Note that this not necessarily includes all non-standard cells, e.g. latches or clock buffers (LCBs) usually are only a few circuit rows high and thus not considered as macros in this section.

Results of this approach are presented in Table 6.5. For each chip, we compare three different macro placements: First a baseline macro placement (base) used by our industrial partner at some stage of the design process. It not necessarily describes the finally produced layout, but it is the latest state known to us. Such placements are created mostly manually in many iterations by experienced designers.

Chip	Run	M-WS [ps]	WS [ps]	FOM [ns]	FOM [%]	NL [m]	NL [%]
Anna	base	−130	−272	−631		41.03	
	BM	−175	−272	−1 667	+164.22	39.74	−3.16
	BM TD	−145	−298	−8 766	+1 289.24	67.78	+65.18
Rosemarie	base	−890	−957	−62 213		88.78	
	BM	−818	−818	−53 442	−14.10	74.87	−15.66
	BM TD	−794	−893	−96 805	+55.60	227.19	+155.92
Sonja	base	−22	−22	−28		0.96	
	BM	−29	−30	−15	−45.44	0.93	−2.96
	BM TD	−21	−21	−13	−54.15	0.93	−3.42
Anja	base	−56	−112	−59		34.49	
	BM	−109	−112	−58	−1.80	30.71	−10.95
	BM TD	−29	−102	−16	−73.02	41.20	+19.46
Jannik	base	−105	−105	−557		4.61	
	BM	−126	−149	−871	+56.42	5.62	+21.89
	BM TD	−89	−100	−683	+22.73	5.98	+29.78
Conni	base	−183	−255	−777		6.22	
	BM	−200	−245	−672	−13.55	4.73	−23.97
	BM TD	−178	−183	−505	−34.97	6.51	+4.64
Lisa	base	−159	−174	−2 089		14.87	
	BM	−187	−201	−3 573	+71.03	19.45	+30.83
	BM TD	−148	−154	−1 605	−23.19	24.69	+66.06
Benedikt	base	−181	−250	−480		7.14	
	BM	−181	−246	−487	+1.60	7.27	+1.86
	BM TD	−210	−281	−572	+19.29	8.22	+15.12
Franziska	base	−36	−44	−56		9.65	
	BM	−28	−35	−17	−70.33	9.26	−4.04
	BM TD	−14	−19	−8	−85.84	9.69	+0.38
Regina	base	−80	−125	−270		36.59	
	BM	−82	−102	−171	−36.80	35.09	−4.08
	BM TD	−80	−105	−236	−12.59	36.29	−0.80
Mia	base	−62	−64	−50		24.53	
	BM	−53	−62	−30	−40.51	24.45	−0.32
	BM TD	−71	−71	−42	−14.90	25.69	+4.74
Elias	base	−108	−248	−1 059		83.63	
	BM	−162	−248	−2 469	+133.11	84.14	+0.61
	BM TD	−102	−248	−1 825	+72.30	89.05	+6.47

Table 6.5: Designer macro placement (base) compared with BonnMacro (BM) and timing-driven BonnMacro (BM TD). Columns denote:

- M-WS: Distance bound model worst slack in [ps].
- WS: Actual worst slack in [ps] determined by timing engine.
- FOM: Path-based figure of merit in [ns].
- NL: Final bounding box netlength of legal placement in [m].

All metrics are compared against the baseline. Improvements in green, marginal deviations in blue, degradations in red.

The second macro placement compared (BM) is the result of standard (timing unaware) BonnMacro. More precisely this includes a shredded placement of macros and subsequent macro legalization. For each macro we try different packing modes (cf. Section 4.2) and solve each resulting local instance of Rectangle Packing with Spark (Funke 2011; Funke, Hougardy and Schneider 2016). The packing modes considered to this end allow up to 3 unconstrained and 10 constrained macros, which empirically leads to good solutions in terms of netlength. Such a legal placement is not post-optimized any further.

Finally we also compare our timing-driven extension of BonnMacro (BM TD). This approach has been described in Section 4.3 in great detail. It starts with the identical shredded input placement as (timing unaware) BonnMacro, but models packing locally as Timing-Driven Rectangle Packing instances.

During timing-driven macro legalization we consider two extra timing-driven packing modes. For each such packing mode we solve a Timing-Driven Rectangle Packing instance via a MIP-formulation (cf. Section 4.3.3). The first allows up to 5 unconstrained and 20 constrained macros, up to 50 000 induced bounds for timing neighbors and a (deterministic) running time limit of roughly 2 min per MIP. The second mode in contrast restricts these numbers to 4, 16, 25 000 and 30 s respectively. For each macro we select the first timing-driven legalization solution obtained within our running time limit. In case both timing-driven packing modes time out, we fall back to a solution with minimum movement as determined by the aforementioned (timing unaware) packing modes.

We solve all MIPs with the barrier interior point method with multiple threads. In order to save running time, we solve MIPs with FOM and movement objective up to 0.5 % relative gap. Moreover we fix spatial relations of the FOM optimal solution for movement minimization.

Since timing-driven macro legalization can deviate further from the shredded input placement, the resulting macro placements can also be much denser. In order to compensate for this effect, we apply the whitespace post-optimization as discussed in Section 4.4. We use this algorithm to introduce small gaps of up to 4 circuit rows between macros while imposing minimal additional movement.

The quality of a macro placement can only be assessed based on a legal placement of the complete netlist. Therefore we evaluate all three macro placements in the same way: We fix the macro positions and place the remaining cells legally using 5 iterations of Self-Stabilizing BonnPlace (cf. Section 5.2).

In order to analyze our timing model of distance bounds more directly, we disregard routability. This allows to consider a more optimistic layer assignment matching underlying assumptions of our model. Thus we enable timing-optimization for Self-Stabilizing BonnPlace in a special congestion-unaware mode and do not perform congestion mitigation.

The resulting three legal placements for the entire netlist are compared in Table 6.5. We list the distance bound model worst slack M-WS in [ps] as well as the actual worst slack in [ps] as computed by the timing engine of IBM. In addition we show the actual path-based FOM, which we present in [ns] in order to simplify comparison of large numbers. Finally we also analyze the bounding box netlength of the placements in [m].

Before we discuss the difference in quality of the three placements, we note that our implementation of the distance bound model worst slack is optimistic in practice, i.e. an upper bound on the actual slack as shown in Proposition 3.3 (page 21). Given the various

sources of possible inaccuracies indicated in Figure 3.1, the distance bound slack correlates fairly well with the actual slack.

Furthermore we observe that on all but 3 instances timing-driven BonnMacro (BM TD) finds the placements with best distance bound slacks. On Anna this macro placement is only outperformed by the baseline. This design almost entirely consists of macros (highest macro density in Table 6.1). Recall that BonnMacro legalizes macros locally in decreasing priority (cf. Section 4.2, page 50). This is why on this particular instance, we have to pay some slack once small macros have to be packed between large ones with limited flexibility.

But the instances Benedikt and Mia require further explanations: Both have medium macro density (cf. Table 6.1). Thus they particularly require macro positions between which standard cells can be placed easily in order to allow enough flexibility for subsequent timing optimization. This requirement is only considered indirectly by BonnMacro as minimization of the deviation from the shredded input placement (in terms of movement). Timing unaware BonnMacro demonstrates that this type of movement objective is reasonable and therefore finds a placement with very competitive distance bound slack.

Timing-driven BonnMacro in contrast is encouraged to deviate from the input placement to maximize slack with respect to yet illegal standard cell positions (particularly for latches). Later, once such macro positions are completed to legal placements of the entire netlist, some detours can not be avoided. This is why the final latch positions are not optimum with respect to distance bound slack. The same effect is also documented by comparing the slack in our model directly after macro legalization (based on illegal standard cell positions). Here these slacks have been much better (Benedikt: $-129\,\text{ps}$, Mia: $-37\,\text{ps}$). This overly optimistic evaluation explains the observed behavior on these two designs.

We refer to Section 6.6.3 for further ideas how to overcome both weaknesses in future evolutions of timing-driven BonnMacro.

On the remaining instances, timing-driven BonnMacro not only consistently finds positions with best distance bound slack, but also outperforms both other placements in terms of actual worst slack. While this is not surprising in comparison to (timing unaware) BonnMacro, it certainly was not expected relative to a hand-crafted baseline.

FOM is only considered as secondary objective by timing-driven BonnMacro. Nevertheless this approach finds significantly better placements in that regard on many instances. Most interestingly, this particularly again applies in comparison to the baseline. Moreover this holds for the great majority of instances, but particularly not for Anna and Rosemarie. These designs stand out, since both violate the premises of the distance bounds timing model (cf. Assumption 3.6; both contain non-endpoint macro-internal timing nodes). Consequently the actual slack deviates a lot from the distance bound slack, which explains why FOM is not optimized well here.

In terms of netlength, Table 6.5 shows the expected trend: Regular BonnMacro targets this objective well and outperforms the baseline on the majority of instances. Timing-driven BonnMacro on the other hand is expected to have higher netlength, since movement is only considered as third tie-breaker objective. The instances with the most drastic netlength increase are those with the highest macro densities. Here denser macro placements complicate placing standard logic, which is reflected in higher netlength.

The described effects are also visualized in Figure 6.8. Here the placements are colored by the distance bounds slacks (cells without any bounds drawn in light gray). Moreover we can compare the most critical distance bounds in the baseline and timing-driven

(a) Baseline Placement **(b)** Timing-Driven BonnMacro

Figure 6.8: Distance bound slacks on Lisa: Cells colored by slack. Overall critical bounds drawn in black.

BonnMacro placements. Note that the respective critical bounds are not necessarily identical. These connections also overall define the actual worst slack. BonnMacro can improve the overall critical connection as it aligns critical macros side by side (e.g. in the upper right of Figure 6.8(b)).

We demonstrated that BonnMacro can optimize various macro placement objectives and particularly often computes solutions with very competitive worst slack. Such macro placements not necessarily are perfect but can provide valuable guidance for designers.

6.6.2 Running Time

Now we analyze the running time of timing-driven macro placement. We first evaluate the difference between various MIP formulations that have been proposed in Section 4.3.3, before we discuss running time contributions of various components of the proposed timing-driven macro placement flow.

Running Time of MIP Solving

Initially we compare the total time for solving different MIP formulations in Table 6.6. Therefore we analyze the deterministic solving times of CPLEX. This metric deterministically measures the computational effort of solving a MIP and does not depend on the current load of the machine. This metric empirically correlates well with the actual wall clock time under benchmarking conditions.

Chip	Run	S [%]	F [%]	M [%]
Anna	MA	−99.37	−55.90	−72.05
	MA NL	−99.87	−72.37	−93.12
Rosemarie	MA	−52.47	+6.56	−46.35
	MA NL	−93.97	−66.53	−91.97
Sonja	MA	−68.02	+60.62	−3.55
	MA NL	−47.14	−86.17	−64.27
Anja	MA	−43.26	−13.76	−1.60
	MA NL	−12.79	−20.66	−49.19
Jannik	MA	−87.17	−85.92	−89.19
	MA NL	−94.61	−95.20	−96.53
Conni	MA	−81.89	+42.75	−9.42
	MA NL	−89.00	−62.53	−89.51
Lisa	MA	−45.05	−48.75	−48.95
	MA NL	−49.67	+14.98	−61.44
Benedikt	MA	+2.30	+4.04	−53.09
	MA NL	+14.23	−4.08	−53.99
Franziska	MA	−72.69	+12.16	−12.80
	MA NL	−88.02	−92.66	−89.43
Regina	MA	−98.41	−98.88	−96.89
	MA NL	−97.82	−99.60	−98.56
Mia	MA	−95.67	−93.58	−93.41
	MA NL	−93.95	−97.51	−98.46
Elias	MA	−92.24	−82.06	−84.93
	MA NL	−93.78	−94.36	−96.40
Summary	MA	−85.28	−61.65	−68.65
	MA NL	−89.86	−85.56	−90.49

Table 6.6: Deterministic CPLEX running time comparison for solving MIPs with slack (S), FOM (F) and movement (M) objective. We show comparisons in [%] with the baseline MIP formulation. MA stands for a formulation based on Manhattan Arcs; MA NL in addition avoids extra variables for measuring bound lengths. All comparisons are summarized as geometric means of the ratios of the deviations.

We compare three different MIP formulations for Timing-Driven Rectangle Packing: As baseline we consider mixed-integer program 4.1 (page 63).

This baseline is compared with two different enhancements. Note that for technical reasons we implemented improvements of Section 4.3.3 in reverse order they have been presented. The first variant (MA) exploits the geometry of Manhattan arcs. Similarly to program 4.3 (page 65), this only requires 4 inequalities for any set of parallel distance bounds and one extra variable for the length of a bound. The second variant (MA NL) avoids this very variable (as already elaborated in program 4.2, page 64) and exactly denotes mixed-integer program 4.3.

All versions use an identical timing-driven packing mode with different constants than those of Section 6.6.1 (no timeout, 3 unconstrained and 10 constrained rectangles, 30 000 induced distance bounds). Note that we compare aggregated total deterministic CPLEX solving running times of all MIPs considered throughout timing-driven BonnMacro. Due to the nature of this algorithm, this not necessarily compares identical Timing-Driven Rectangle Packing instances. Nonetheless the results show a strong tendency.

Table 6.6 separately lists the times for solving the MIP with slack (S), FOM (F) and movement (M) objectives. Note that we only show the relative running times compared to the baseline in [%] since the actual absolute values are not very meaningful.

The comparisons in Table 6.6 very clearly show drastic running time improvements. Despite rare outliers, the overall time spent for CPLEX is significantly reduced on all chips and for all objectives. The best MIP formulation MA NL reduces the MIP solving running time by more than 85 % on average for all objectives. Without these improvements more generous packing modes as used in Section 6.6.1 would not have been applicable.

Seemingly the majority of the running time improvement stems from exploiting Manhattan arcs. But actually, both the baseline and MA optimized FOM-MIPs with fixed relations only. This is not done in MA NL any longer, which is why improvements of MA NL in FOM-MIPs particularly stand out. Moreover many instances that using MA became slower than the baseline, significantly speed up with MA NL.

In addition, the presented improvements do not only result from speeding up few instances with enormous running times. Also the maximum (deterministic CPLEX) time spent for solving single instances decreases almost identically as the totals presented in Table 6.6. This holds across all considered chips and objectives.

We also tried to solve our MIPs even faster by providing MIP starts for FOM and movement MIPs. These can be inferred for the coordinate variables of rectangles from the previously solved programs. But this actually turned out to be harmful since CPLEX became slower with these initial assignments.

In summary, it therefore can be said that mixed-integer program 4.3 (Proposition 4.6, page 65) provides an essential improvement. This is why it has been used in all timing-driven BonnMacro experiments, particularly those discussed in the following section.

Timing-Driven BonnMacro-Flow

Now we analyze the running times of our proposed timing-driven macro placement flow.

Figure 6.9 visualizes the running times of different components in this flow. The shown data refers to the BM TD runs of Table 6.5 on the same instances. We compare accumulated total wall clock running times of these experiments.

Chronologically the first contribution is Self-Stabilizing BonnPlace with shredded macros denoted by ■ BP (A). Recall that we run 3 iterations here using a special clustering for macro fragments.

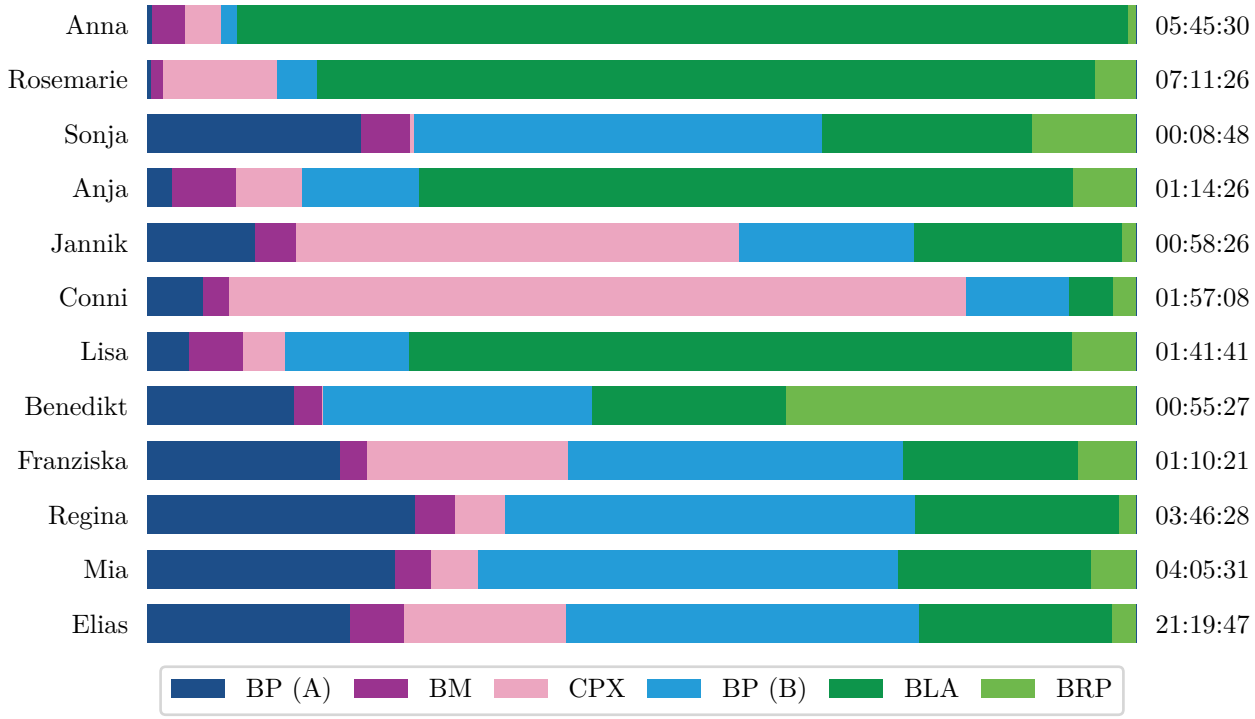


Figure 6.9: Running times of timing-driven BonnMacro components: Self-Stabilizing BonnPlace with shredded macros BP (A) and as evaluation with fixed macros BP (B), BonnMacro framework (BM) and MIP solving with CPLEX (CPX), timing optimizations BonnLayerAssignment (BLA) and BonnRefinePlace (BRP). Running time totals in [H:M:S].

Afterwards we display the contribution of the BonnMacro framework (■ BM). This not only accounts for the overhead of selecting local packing instances, but also in our timing-driven mode for library analysis, loading of distance bounds (Proposition 3.15) and construction of mixed-integer programs. The total time of solving these MIPs with CPLEX is summarized by ■ CPX.

The resulting legal macro placement is afterwards considered as fixed. We evaluate timing characteristics when this solution is extended to all standard cells. Therefore we perform 5 iterations of timing-aware Self-Stabilizing BonnPlace denoted by ■ BP (B). This step utilizes (congestion unaware) layer assignment using BonnLayerAssignment (■ BLA) as well as local placement changes using BonnRefinePlace (■ BRP).

We summarize the respective total running times on the right of Figure 6.9 in [H:M:S].

Figure 6.9 clearly shows that MIP solving consumes a significant amount of running time during timing-driven BonnMacro. The classical (timing unaware) rectangle packing solutions based on Spark (Funke, Hougardy and Schneider 2016) are also computed during BM as additional alternatives. But this type of macro legalization runs so quickly that it is even dominated by library analysis and distance bound loading in our timing driven mode.

But apart from that the timing evaluation of a macro placement using Self-Stabilizing BonnPlace still dominates the running time on the majority of test cases. Even BonnLayerAssignment alone is more running time intensive than CPLEX on all

but two designs. Even though CPLEX running time could likely be reduced slightly by further parameter tuning, this would not affect the overall running time significantly.

On the contrary we also explored larger packing instance parameters. Recall that we considered instances with up to 5 unconstrained, 20 constrained macros and up to 50 000 distance bounds for the experiments presented so far. But as expected larger mixed-integer programs hardly pay off: On the one hand, doubling the number of distance bounds on Mia increases running time by 1293 %. On this instance CPLEX even fails to complete a first cut-introducing phase within 1 hour of deterministic running time for 10 % of the packing instances. On the other hand, considering up to 7 unconstrained and 30 constrained macros on Rosemarie results in 1644 % more running time.

The deterministic CPLEX time limit only is exceeded on 0.7 % of all Timing-Driven Rectangle Packing instances. In 92 % of these cases, we still find a satisfactory solution and can continue: Either the next timing-driven packing mode with reduced constants succeeds or the gap between the best known integer and fractional solutions is sufficiently small (below 5 %). For the most part the time limit is reached on FOM MIPs. This justifies our choice of the timeout to avoid huge running times caused by few instances.

We demonstrated that our Timing-Driven Rectangle Packing instances are selected reasonably and considering larger constants can hardly be justified.

6.6.3 Outlook

At this point we want to indicate a few potential future extensions for BonnMacro. All are inspired by current weaknesses and could help improve the quality of results.

A first issue is posed by relevant macro-internal non-endpoint timing nodes. Those can not be handled well using the model of slack based on distance bounds. Apportioning slack to paths before and after such nodes would be possible upfront, but likely requires many iterations until such estimates are reasonable everywhere.

On the other hand, small subsets of such nodes could be included into our formulation as mixed-integer program. For this purpose we could propagate arrival times directly within such a MIP similar to actual virtual timing. Such nodes could also be included lazily and on demand once they become timing critical. Since arrival time propagation expectedly is much more expensive than simply bounding distances with Manhattan arcs, such an extension must be used cautiously.

In order to improve the space management for standard logic, two approaches are possible: On the one hand, we could optimize slack and FOM while restricting movement artificially (overall movement and/or individual movement of single macros). Consequently macro positions can deviate less from the shredded input positions which often leads to reduced detours and netlength (cf. timing unaware BonnMacro). On the other hand, such a restricted solution space might not be sufficient for optimizing timing properties. This could be compensated by iterating this approach with a new shredded placement guided by former iterations.

Furthermore logic could also be included in the macro legalization directly. Therefore we could cluster standard cells into few large groups which could be respected in local packing instances as rectangles with flexible aspect ratio. Similar approaches have already been applied for floorplanning (Adya and Markov 2003). Using excellent clusters, many detours might be avoidable which should result in better netlength. At the same time such an approach also provides the flexibility of incorporating a direct congestion mitigation into macro placement. Therefore similar techniques as used in Self-Stabilizing BonnPlace can be applied to the respective clusters.

Finally locally suboptimal solutions likely can be avoided by a post-optimization: Similar to the applied macro legalization, we simply could continue to solve local packing instances. This might help improve the slack of smaller macros that are legalized at the very last (as discussed on Anna). Such an approach requires a careful tradeoff between additional running time and quality.

Summary

Placement is an important step in chip design, the process of finding physical layouts for electronic computer chips. The basic task during placement is to arrange the building blocks of the chip, the circuits, disjointly within a given chip area. Since placement serves as baseline for multiple further optimizations, its quality has far reaching effects on the overall solution. The main challenges in placement originate in requirements of the subsequent design flow. This particularly includes finding positions that result in short circuit interconnections, that can be routed easily and that ensure all signals arrive in time. In this thesis we investigate various aspects of these challenges.

We mostly focus on the relatively few largest circuits on a chip, the macros. Due to the size of macros, their placement pre-determines many characteristics of the chip. But at the same time macro placement quality can hardly be assessed without placing all circuits. In order to still optimize timing, we propose a new timing model on macros which is based on distance bounds. This model can be evaluated for positions of macros together with only few extra cells. We prove that it bounds the worst slack that can be achieved by extending macro positions to all circuits. Yet it is accurate enough to predict these timing traits under certain assumptions that are usually met in practice.

We show how this model can be computed efficiently and how equivalent but smaller representations can be obtained. Packing rectangles disjointly remains strongly NP-hard under slack maximization in our timing model. Despite of this we develop an algorithm that solves special cases in $\mathcal{O}(nm)$ time where n and m denote the number of rectangles and distance bounds.

The proposed timing model is also incorporated into BonnMacro. This is the macro placement component of the placement framework BonnPlace developed at the Research Institute for Discrete Mathematics of the University of Bonn. We extend the paradigm of legalizing macros locally to optimize timing. Using efficient formulations as mixed-integer programs relatively large local subinstances can be handled in practice in reasonable time. This results in the first timing-aware macro placement tool.

In addition, we provide multiple enhancements for the partitioning-based standard cell placer BonnPlaceGlobal of BonnPlace. We find a provably smallest model of partitioning as minimum-cost flow problem via reduction to a reachability problem in graphs. Thus we can avoid running time intensive instances using an efficient sweep-line pre-processing and a thoughtful implementation. Moreover we propose the new global placement flow Self-Stabilizing BonnPlace. This approach combines partitioning-based BonnPlaceGlobal with a force-directed placement framework. It provides the flexibility to optimize the involved objectives routability and timing during placement.

The performance of our placement tools is confirmed on a large variety of real-world designs provided by our industrial partner IBM. We reduce running time of BonnPlaceGlobal by up to 85 % on single instances and by 27 % on average on a large

testbed. Using Self-Stabilizing BonnPlace, we achieve the best published results on the 2012 DAC congestion-driven placement contest (Viswanathan et al. 2012). Moreover this framework finds easily routable placements for challenging designs, even when simultaneously optimizing timing objectives. Self-Stabilizing BonnPlace and timing-driven BonnMacro can be combined to a mixed-size placement flow. This combination often finds placements with a very competitive worst slack and even outperforms solutions that have been determined manually by experienced designers.

Bibliography

G. M. Adelson-Velsky and E. M. Landis (1962). "An algorithm for the organization of information". Trans. by J. R. Myron. In: *Soviet Mathematics – Doklady* 3, pp. 1259–1263 (cit. on p. 77).

S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa and I. L. Markov (2004). "Unification of Partitioning, Placement and Floorplanning". In: *Proceedings of the 2004 International Conference on Computer-Aided Design*. ICCAD (San Jose, California, USA), pp. 550–557. DOI: 10.1109/ICCAD.2004.1382639 (cit. on p. 12).

S. N. Adya and I. L. Markov (2003). "Fixed-outline Floorplanning: Enabling Hierarchical Design". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11.6, pp. 1120–1135. DOI: 10.1109/TVLSI.2003.817546 (cit. on pp. 14, 119).

S. N. Adya and I. L. Markov (2005). "Combinatorial Techniques for Mixed-size Placement". In: *ACM Transactions on Design Automation of Electronic Systems* 10.1, pp. 58–90. DOI: 10.1145/1044111.1044116 (cit. on pp. 12, 48, 50).

S. N. Adya and X. Yang (2008). "Congestion-Driven Physical Design". In: *Handbook of Algorithms for Physical Design Automation*. Ed. by C. J. Alpert, D. P. Mehta and S. S. Sapatnekar, pp. 447–466 (cit. on p. 17).

M. Ahrens, M. Gester, N. Klewinghaus, D. Müller, S. Peyer, C. Schulte and G. Téllez (2015). "Detailed Routing Algorithms for Advanced Technology Nodes". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.4, pp. 563–576. DOI: 10.1109/TCAD.2014.2385755 (cit. on p. 92).

C. J. Alpert, J. Hu, S. S. Sapatnekar and C. N. Sze (2006). "Accurate Estimation of Global Buffer Delay Within a Floorplan". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.6, pp. 1140–1145. DOI: 10.1109/TCAD.2005.855889 (cit. on p. 10).

C. J. Alpert, D. P. Mehta and S. S. Sapatnekar, eds. (2008). *Handbook of Algorithms for Physical Design Automation* (cit. on pp. 8, 10, 16).

A. Armando, C. Castellini and E. Giunchiglia (1999–1999). "SAT-Based Procedures for Temporal Reasoning". In: *Recent Advances in AI Planning. 5th European Conference on Planning*. ECP (Durham, UK). Ed. by S. Biundo and M. Fox, pp. 97–108. DOI: 10.1007/10720246_8 (cit. on p. 63).

R. Bayer (1972). "Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms". In: *Acta Informatica* 1.4, pp. 290–306. DOI: 10.1007/BF00289509 (cit. on p. 77).

R. Bellman (1958). "On a routing problem". In: *Quarterly of Applied Mathematics* (16), pp. 87–90. DOI: 10.1090/qam/102435 (cit. on pp. 37, 38).

P. Berman and B. DasGupta (1992). "Approximating the rectilinear polygon cover problems". In: *Proceedings of the Fourth Canadian Conference on Computational Geometry* (St. John's, Newfoundland, Canada), pp. 229–235 (cit. on p. 59).

M. Bern and P. Plassmann (1989). "The Steiner problem with edge lengths 1 and 2". In: *Information Processing Letters* 32.4, pp. 171–176. DOI: 10.1016/0020-0190(89)90039-2 (cit. on p. 8).

A. Bock, S. Held, N. Kämmerling and U. Schorr (2015). "Local Search Algorithms for Timing-driven Placement Under Arbitrary Delay Models". In: *Proceedings of the 52nd Design Automation Conference*. DAC (San Francisco, California, USA), Art. No. 29. DOI: 10.1145/2744769.2744867 (cit. on p. 94).

A. Bock (2010). "Postoptimierung durch Umplatzierung und Gate Sizing". Diploma Thesis. Research Institute for Discrete Mathematics, University of Bonn (cit. on p. 94).

A. Bortfeldt (2013). "A reduction approach for solving the rectangle packing area minimization problem". In: *European Journal of Operational Research* 224.3, pp. 486–496. DOI: 10.1016/j.ejor.2012.08.006 (cit. on p. 14).

U. Brenner (2007). *An Effective Algorithm for Mixed-Size VLSI Placement*. Technical Report 07970. Research Institute for Discrete Mathematics, University of Bonn (cit. on pp. 12, 47, 53).

U. Brenner (2012). "VLSI Legalization with Minimum Perturbation by Iterative Augmentation". In: *Proceedings of the 2012 Conference on Design, Automation and Test in Europe*. DATE (Dresden, Germany), pp. 1385–1390. DOI: 10.1109/DATE.2012.6176579 (cit. on pp. 89, 94).

U. Brenner, A. Hermann, N. Hoppmann and P. Ochsendorf (2015). "BonnPlace: A Self-Stabilizing Placement Framework". In: *Proceedings of the 2015 International Symposium on Physical Design*. ISPD (Monterey, California, USA), pp. 9–16. DOI: 10.1145/2717764.2717778 (cit. on pp. 75, 76, 88, 95, 107, 108, 110).

U. Brenner and A. Rohe (2003). "An Effective Congestion-Driven Placement Framework". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22.4, pp. 387–394. DOI: 10.1109/TCAD.2003.809662 (cit. on pp. 16, 93).

U. Brenner, M. Struzyna and J. Vygen (2008). "BonnPlace: Placement of Leading-Edge Chips by Advanced Combinatorial Algorithms". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.9, pp. 1607–1620. DOI: 10.1109/TCAD.2008.927674 (cit. on pp. 12, 16, 47, 75, 76, 89).

U. Brenner and J. Vygen (2001). "Worst-Case Ratios of Networks in the Rectilinear Plane". In: *Networks. An International Journal* 38.3, pp. 126–139. DOI: 10.1002/net.1031 (cit. on p. 89).

U. Brenner and J. Vygen (2008). "Analytical Methods in Placement". In: *Handbook of Algorithms for Physical Design Automation*. Ed. by C. J. Alpert, D. P. Mehta and S. S. Sapatnekar, pp. 327–346 (cit. on p. 15).

M. Burstein and M. N. Youssef (1985). "Timing Influenced Layout Design". In: *Proceedings of the 22nd Design Automation Conference*. DAC (Las Vegas, Nevada, USA), pp. 124–130. DOI: 10.1109/DAC.1985.1585923 (cit. on p. 17).

T. F. Chan, J. Cong, J. R. Shinnerl, K. Sze and M. Xie (2006). "mPL6: Enhanced Multilevel Mixed-Size Placement". In: *Proceedings of the 2006 International Symposium on Physical Design*. ISPD (San Jose, California, USA), pp. 212–214. DOI: 10.1145/1123008.1123055 (cit. on pp. 12, 16).

T. F. Chan, K. Sze, J. R. Shinnerl and M. Xie (2007). "mPL6: Enhanced Multilevel Mixed-Size Placement with Congestion Control". In: *Modern Circuit Placement: Best Practices and Results*. Ed. by G.-J. Nam and J. Cong, pp. 247–288. DOI: 10.1007/978-0-387-68739-1_10 (cit. on pp. 12, 16).

Y.-C. Chang, Y.-W. Chang, G.-M. Wu and S.-W. Wu (2000). "B*-Trees: A New Representation for Non-slicing Floorplans". In: *Proceedings of the 37th Design Automation Conference*. DAC (Los Angeles, California, United States), pp. 458–463. DOI: 10.1145/337292.337541 (cit. on p. 14).

T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese and A. B. Kahng (1992). "Zero Skew Clock Routing with Minimum Wirelength". In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 39.11, pp. 799–814. DOI: 10.1109/82.204128 (cit. on p. 39).

H. Chen, C.-K. Cheng, N.-C. Chou, A. B. Kahng, J. MacDonald, P. Suaris, B. Yao and Z. Zhu (2003). "An Algebraic Multigrid Solver for Analytical Placement with Layout Based Cluster-

ing". In: *Proceedings of the 40th Design Automation Conference*. DAC (Anaheim, California, USA), pp. 794–799. DOI: 10.1109/DAC.2003.1219127 (cit. on p. 16).

T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen and Y.-W. Chang (2008). "NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs With Preplaced Blocks and Density Constraints". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.7, pp. 1228–1240. DOI: 10.1109/TCAD.2008.923063 (cit. on pp. 12, 17).

T.-C. Chen and Y.-W. Chang (2008). "Packing Floorplan Representations". In: *Handbook of Algorithms for Physical Design Automation*. Ed. by C. J. Alpert, D. P. Mehta and S. S. Sapatnekar, pp. 203–238 (cit. on p. 14).

T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang and T.-Y. Liu (2008). "MP-Trees: A Packing-Based Macro Placement Algorithm for Modern Mixed-Size Designs". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.9, pp. 1621–1634. DOI: 10.1109/TCAD.2008.927760 (cit. on p. 14).

C.-L. E. Cheng (1994). "RISA: Accurate and Efficient Placement Routability Modeling". In: *Proceedings of the 1994 International Conference on Computer-Aided Design*. ICCAD (San Jose, California, USA), pp. 690–695. DOI: 10.1109/ICCAD.1994.629897 (cit. on p. 16).

J. Cong, L. Guojie, K. Tsota and X. Bingjun (2013). "Optimizing Routability in Large-Scale Mixed-Size Placement". In: *Proceedings of the 18th Asia and South Pacific Design Automation Conference*. ASP-DAC (Yokohama, Japan), pp. 441–446. DOI: 10.1109/ASPDAC.2013.6509636 (cit. on pp. 106–108).

J. Cong, M. Romesis and J. R. Shinnerl (2005). "Robust Mixed-size Placement Under Tight White-space Constraints". In: *Proceedings of the 2005 International Conference on Computer-Aided Design*. ICCAD (San Jose, California, USA), pp. 165–172. DOI: 10.1109/ICCAD.2005.1560058 (cit. on p. 12).

J. C. Culberson and R. A. Reckhow (1988). "Covering Polygons is Hard". In: *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pp. 601–611. DOI: 10.1109/SFCS.1988.21976 (cit. on p. 59).

K. Doll, F. M. Johannes and K. J. Antreich (1994). "Iterative Placement Improvement by Network Flow Methods". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.10, pp. 1189–1200. DOI: 10.1109/43.317462 (cit. on pp. 12, 48).

A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak and M. Wiesel (1984). "Chip Layout Optimization Using Critical Path Weighting". In: *Proceedings of the 21st Design Automation Conference*. DAC (Albuquerque, New Mexico, USA), pp. 133–136. DOI: 10.1109/DAC.1984.1585786 (cit. on p. 17).

J. Edmonds and R. M. Karp (1972). "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *Journal of the ACM (JACM)* 19.2, pp. 248–264. DOI: 10.1145/321694.321699 (cit. on p. 89).

H. Eisenmann and F. M. Johannes (1998). "Generic Global Placement and Floorplanning". In: *Proceedings of the 35th Design Automation Conference*. DAC (San Francisco, California, USA), pp. 269–274. DOI: 10.1145/277044.277119 (cit. on pp. 12, 15, 17).

W. C. Elmore (1948). "The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers". In: *Journal of Applied Physics* 19.1, pp. 55–63. DOI: 10.1063/1.1697872 (cit. on p. 9).

C. Engels (2013). "Algorithms for Macro Placement". Master's Thesis. Research Institute for Discrete Mathematics, University of Bonn (cit. on pp. 50, 57, 58, 73).

P. Fernando and S. Katkoori (2008). "An Elitist Non-Dominated Sorting Based Genetic Algorithm for Simultaneous Area and Wirelength Minimization in VLSI Floorplanning". In: *21st International Conference on VLSI Design* (Hyderabad, India), pp. 337–342. DOI: 10.1109/VLSI.2008.97 (cit. on p. 14).

R. A. Finkel and J. L. Bentley (1974). "Quad Trees A Data Structure for Retrieval on Composite Keys". In: *Acta Informatica* 4.1, pp. 1–9. DOI: 10.1007/BF00288933 (cit. on p. 78).

L. R. Ford (1956). *Network Flow Theory*. Tech. rep. P-923. Rand Corporation, Santa Monica (cit. on p. 37).

D. S. Franzblau (1989). "Performance Guarantees on a Sweep-Line Heuristic for Covering Rectilinear Polygons with Rectangles". In: *SIAM Journal on Discrete Mathematics* 2.3, pp. 307–321. DOI: 10.1137/0402027 (cit. on p. 59).

M. L. Fredman and R. E. Tarjan (1987). "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms". In: *Journal of the ACM (JACM)* 34.3, pp. 596–615. DOI: 10.1145/28869.28874 (cit. on p. 33).

J. Funke (2011). "Netzlängenoptimale Platzierung von VLSI-Chips". German. Diploma Thesis. Research Institute for Discrete Mathematics, University of Bonn (cit. on pp. 52, 58, 113).

J. Funke, S. Hougardy and J. Schneider (2016). "An Exact Algorithm for Wirelength Optimal Placements in VLSI Design". In: *Integration. the VLSI Journal* 52, pp. 355–366. DOI: 10.1016/j.vlsi.2015.07.001 (cit. on pp. 14, 47, 52, 58, 63, 71, 113, 118).

T. Gao, P. M. Vaidya and C. L. Liu (1992). "A Performance Driven Macro-Cell Placement Algorithm". In: *Proceedings of the 29th Design Automation Conference* (Anaheim, California, USA), pp. 147–152. DOI: 10.1109/DAC.1992.227845 (cit. on p. 14).

M. R. Garey and D. S. Johnson (1975). "Complexity Results for Multiprocessor Scheduling under Resource Constraints". In: *SIAM Journal on Computing* 4, pp. 397–411. DOI: 10.1137/0204035 (cit. on pp. 25, 27).

M. R. Garey and D. S. Johnson (1978). ""Strong" NP-Completeness Results: Motivation, Examples, and Implications". In: *Journal of the ACM (JACM)* 25.3, pp. 499–508. DOI: 10.1145/322077.322090 (cit. on p. 25).

M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte and J. Vygen (2013). "BonnRoute: Algorithms and Data Structures for Fast and Good VLSI Routing". In: *ACM Transactions on Design Automation of Electronic Systems* 18.2, Art. No. 32. DOI: 10.1145/2442087.2442103 (cit. on pp. 8, 92).

A. V. Goldberg and A. V. Karzanov (2004). "Maximum skew-symmetric flows and matchings". In: *Mathematical Programming* 100.3, pp. 537–568. DOI: 10.1007/s10107-004-0505-z (cit. on p. 45).

L. J. Guibas and R. Sedgewick (1978). "A Dichromatic Framework for Balanced Trees". In: *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*. FOCS (Ann Arbor, Michigan, USA), pp. 8–21. DOI: 10.1109/SFCS.1978.3 (cit. on p. 77).

P.-N. Guo, C.-K. Cheng and T. Yoshimura (1999). "An O-Tree Representation of Non-Slicing Floorplan and Its Applications". In: *Proceedings of the 36th Design Automation Conference*. DAC (New Orleans, Louisiana, United States), pp. 268–273. DOI: 10.1109/DAC.1999.781324 (cit. on p. 14).

T. Hamada, C.-K. Cheng and P. M. Chau (1993). "Prime: A Timing-driven Placement Tool Using a Piecewise Linear Resistive Network Approach". In: *Proceedings of the 30th Design Automation Conference*. DAC (Dallas, Texas, USA), pp. 531–536. DOI: 10.1145/157485.165015 (cit. on p. 17).

M. Hanan (1966). "On Steiner's Problem with Rectilinear Distance". In: *SIAM Journal on Applied Mathematics* 14.2, pp. 255–265. DOI: 10.1137/0114025 (cit. on pp. 56, 78).

X. He, T. Huang, L. Xiao, H. Tian, G. Cui and E. F. Y. Young (2011). "Ripple: An Effective Routability-driven Placer by Iterative Cell Movement". In: *Proceedings of the 2011 International Conference on Computer-Aided Design*. ICCAD (San Jose, California, USA), pp. 74–79. DOI: 10.1109/ICCAD.2011.6105308 (cit. on p. 15).

M. R. Hestenes and E. Stiefel (1952). "Methods of conjugate gradients for solving linear systems". In: *Journal of Research of the National Institute of Standards and Technology* 49.1, pp. 409–439 (cit. on p. 75).

N. J. Higham (1993). "The Accuracy of Floating Point Summation". In: *SIAM Journal on Scientific Computing* 14.4, pp. 783–799. DOI: 10.1137/0914050 (cit. on p. 87).

R. B. Hitchcock, G. L. Smith and D. D. Cheng (1982). "Timing Analysis of Computer Hardware". In: *IBM Journal of Research and Development* 26.1, pp. 100–105. DOI: 10.1147/rd.261.0100 (cit. on p. 9).

N. Hoppmann (2014). "Self-Stabilizing BonnPlace – A Self-Stabilizing Global Placement Framework". Master's Thesis. Research Institute for Discrete Mathematics, University of Bonn (cit. on pp. 90, 91).

W. Hou, H. Yu, X. Hong, Y. Cai, W. Wu, J. Gu and W. H. Kao (2001). "A New Congestion-Driven Placement Algorithm Based on Cell Inflation". In: *Proceedings of the 6th Asia and South Pacific Design Automation Conference*. ASP-DAC (Yokohama, Japan), pp. 605–608. DOI: 10.1145/370155.370560 (cit. on p. 16).

M.-K. Hsu, V. Balabanov and Y.-W. Chang (2013). "TSV-Aware Analytical Placement for 3-D IC Designs Based on a Novel Weighted-Average Wirelength Model". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32.4, pp. 497–509. DOI: 10.1109/TCAD.2012.2226584 (cit. on p. 15).

M.-K. Hsu, S. Chou, T.-H. Lin and Y.-W. Chang (2011). "Routability-driven Analytical Placement for Mixed-size Circuit Designs". In: *Proceedings of the 2011 International Conference on Computer-Aided Design*. ICCAD (San Jose, California, USA), pp. 80–84. DOI: 10.1109/ICCAD.2011.6105309 (cit. on pp. 16, 17, 106, 108).

B. Hu and M. Marek-Sadowska (2005). "Multilevel Fixed-Point-Addition-Based VLSI Placement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.8, pp. 1188–1203. DOI: 10.1109/TCAD.2005.850802 (cit. on p. 16).

J. Hu, J. A. Roy and I. L. Markov (2010). "Completing High-quality Global Routes". In: *Proceedings of the 2010 International Symposium on Physical Design*. ISPD (San Francisco, California, USA), pp. 35–41. DOI: 10.1145/1735023.1735035 (cit. on p. 16).

M. A. B. Jackson and E. S. Kuh (1989). "Performance-driven Placement of Cell Based IC's". In: *Proceedings of the 26th Design Automation Conference*. DAC (Las Vegas, Nevada, USA), pp. 370–375. DOI: 10.1145/74382.74444 (cit. on p. 17).

A. Janiak, A. Kozik and M. Lichtenstein (2010). "New perspectives in VLSI design automation: deterministic packing by Sequence Pair". In: *Annals of Operations Research* 179.1, pp. 35–56. DOI: 10.1007/s10479-008-0460-9 (cit. on p. 14).

M. Jerrum (1985). *Complementary Partial Orders and Rectangle Packing*. Tech. rep. CSR-190-85. University of Edinburgh, Department of Computer Science. 10 pp. (cit. on p. 14).

W. Kahan (1965). "Further Remarks on Reducing Truncation Errors". In: *Communications of the ACM* 8.1, p. 40. DOI: 10.1145/363707.363723 (cit. on p. 87).

A. B. Kahng and Q. Wang (2005). "Implementation and Extensibility of an Analytic Placer". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.5, pp. 734–747. DOI: 10.1109/TCAD.2005.846366 (cit. on pp. 12, 16).

R. M. Karp (1972). "Reducibility Among Combinatorial Problems". In: *Complexity of Computer Computations*. The IBM Research Symposia Series, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9 (cit. on pp. 8, 25).

R. M. Karp (1978). "A characterization of the minimum cycle mean in a digraph". In: *Discrete Mathematics* 23.3, pp. 309–311. DOI: 10.1016/0012-365X(78)90011-0 (cit. on pp. 32, 37).

R. M. Karp and J. B. Orlin (1981). "Parametric shortest path algorithms with an application to cyclic staffing". In: *Discrete Applied Mathematics* 3.1, pp. 37–45. DOI: 10.1016/0166-218X(81)90026-3 (cit. on p. 33).

A. Khatkhate, C. Li, A. R. Agnihotri, M. C. Yildiz, S. Ono, C.-K. Koh and P. H. Madden (2004). "Recursive Bisection Based Mixed Block Placement". In: *Proceedings of the 2004 International Symposium on Physical Design* (Phoenix, Arizona, USA), pp. 84–89. DOI: 10.1145/981066.981084 (cit. on p. 12).

M.-C. Kim, N. Viswanathan, C. J. Alpert, I. Markov and S. Ramji (2012). "MAPLE: Multilevel Adaptive PLacEment for Mixed-Size Designs". In: *Proceedings of the 2012 International Symposium on Physical Design*. ISPD (Napa, California, USA), pp. 193–200. DOI: 10.1145/2160916.2160958 (cit. on pp. 12, 15).

M.-C. Kim, J. Hu, D.-J. Lee and I. Markov (2011). "A SimPLR Method for Routability-driven Placement". In: *Proceedings of the 2011 International Conference on Computer-Aided Design*. ICCAD (San Jose, California, USA), pp. 67–73. DOI: 10.1109/ICCAD.2011.6105307 (cit. on pp. 15, 16).

M.-C. Kim, D.-J. Lee and I. L. Markov (2012). "SimPL: An Effective Placement Algorithm". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.1, pp. 50–60. DOI: 10.1109/TCAD.2011.2170567 (cit. on pp. 15, 16, 89, 103).

M.-C. Kim, D.-J. Lee and I. L. Markov (2013). "SimPL: An Algorithm for Placing VLSI Circuits". In: *Communications of the ACM* 56.6, pp. 105–113. DOI: 10.1145/2461256.2461279 (cit. on pp. 15, 89, 103).

M.-C. Kim and I. L. Markov (2012). "ComPLx: A Competitive Primal-dual Lagrange Optimization for Global Placement". In: *Proceedings of the 49th Design Automation Conference*. DAC (San Francisco, California, USA), pp. 747–752. DOI: 10.1145/2228360.2228496 (cit. on pp. 12, 15).

T. I. Kirkpatrick and N. R. Clark (1966). "Pert as an Aid to Logic Design". In: *IBM Journal of Research and Development* 10.2, pp. 135–141. DOI: 10.1147/rd.102.0135 (cit. on p. 9).

J. M. Kleinhans, G. Sigl, F. M. Johannes and K. J. Antreich (1991). "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 10.3, pp. 356–365. DOI: 10.1109/43.67789 (cit. on p. 15).

W. Kocay and D. Stone (1993). "Balanced network flows". In: *Bulletin of the Institute of Combinatorics and its Applications* 7, pp. 17–32 (cit. on p. 45).

W. Kocay and D. Stone (1995). "An Algorithm for Balanced Flows". In: *Journal of Combinatorial Mathematics and Combinatorial Computing* 19, pp. 3–32 (cit. on p. 45).

T. Kong (2002). "A Novel Net Weighting Algorithm for Timing-driven Placement". In: *Proceedings of the 2002 International Conference on Computer-Aided Design*. ICCAD (San Jose, California, USA), pp. 172–176. DOI: 10.1145/774572.774597 (cit. on p. 17).

R. E. Korf, M. D. Moffitt and M. E. Pollack (2010). "Optimal rectangle packing". In: *Annals of Operations Research* 179.1, pp. 261–295. DOI: 10.1007/s10479-008-0463-6 (cit. on pp. 14, 63).

B. H. Korte and J. Vygen (2018). *Combinatorial Optimization. Theory and Algorithms*. 6th ed. Vol. 21. 698 pp. DOI: 10.1007/978-3-662-56039-6 (cit. on pp. 4, 22).

B. Korte, D. Rautenbach and J. Vygen (2007). "BonnTools: Mathematical Innovation for Layout and Timing Closure of Systems on a Chip". In: *Proceedings of the IEEE* 95.3, pp. 555–572. DOI: 10.1109/JPROC.2006.889373 (cit. on pp. 1, 75, 88).

H. W. Lenstra (1983). "Integer Programming with a Fixed Number of Variables". In: *Mathematics of Operations Research* 8.4, pp. 538–548. DOI: 10.1287/moor.8.4.538 (cit. on p. 72).

Z. Li, D. A. Papa, C. J. Alpert, S. Hu, W. Shi, C. Sze and Y. Zhou (2010). "Ultra-Fast Interconnect Driven Cell Cloning For Minimizing Critical Path Delay". In: *Proceedings of the 2010 International Symposium on Physical Design*. ISPD (San Francisco, California, USA), pp. 75–82. DOI: 10.1145/1735023.1735047 (cit. on p. 39).

J.-M. Lin and Y.-W. Chang (2005). "TCG: A Transitive Closure Graph-Based Representation for General Floorplans". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 13.2, pp. 288–292. DOI: 10.1109/TVLSI.2004.840760 (cit. on p. 14).

J.-M. Lin, Y.-W. Chang and S.-P. Lin (2003). "Corner Sequence - A P-Admissible Floorplan Representation With a Worst Case Linear-Time Packing Scheme". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 11.4, pp. 679–686. DOI: 10.1109/TVLSI.2003.816137 (cit. on p. 14).

T. Lin and C. Chu (2014). "POLAR 2.0: An Effective Routability-Driven Placer". In: *Proceedings of the 51nd Design Automation Conference*. DAC (San Francisco, California, USA), Art. No. 123. DOI: 10.1145/2593069.2593181 (cit. on pp. 15, 17).

T. Lin, C. Chu, J. R. Shinnerl, I. Bustany and I. Nedelchev (2015). "POLAR: A High Performance Mixed-Size Wirelengh-Driven Placer With Density Constraints". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.3, pp. 447–459. DOI: 10.1109/TCAD.2015.2394383 (cit. on p. 15).

F. Liu and P. Feldmann (2008–2008). "MAISE: An Interconnect Simulation Engine for Timing and Noise Analysis". In: *9th International Symposium on Quality Electronic Design*. ISQED (San Jose, California, USA), pp. 621–626. DOI: 10.1109/ISQED.2008.4479809 (cit. on p. 10).

W.-H. Liu, W.-C. Kao, Y.-L. Li and K.-Y. Chao (2010). "Multi-threaded Collision-aware Global Routing with Bounded-length Maze Routing". In: *Proceedings of the 47th Design Automation Conference*. DAC (Anaheim, California, USA), pp. 200–205. DOI: 10.1145/1837274.1837324 (cit. on p. 106).

W.-H. Liu, Y.-L. Li and C.-K. Koh (2012). "A Fast Maze-free Routing Congestion Estimator with Hybrid Unilateral Monotonic Routing". In: *Proceedings of the 2012 International Conference on Computer-Aided Design*. ICCAD (San Jose, California, USA), pp. 713–719 (cit. on p. 106).

J. Lu, H. Zhuang, P. Chen, H. Chang, C. Chang, Y. Wong, L. Sha, D. Huang, Y. Luo, C. Teng and C. Cheng (2015). "ePlace-MS: Electrostatics-Based Placement for Mixed-Size Circuits". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.5, pp. 685–698. DOI: 10.1109/TCAD.2015.2391263 (cit. on pp. 12, 16).

J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng and C.-K. Cheng (2015). "ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method". In: *ACM Transactions on Design Automation of Electronic Systems* 20.2, Art. No. 17. DOI: 10.1145/2699873 (cit. on pp. 12, 16, 89).

T. Luo, D. Newmark and D. Z. Pan (2006). "A New LP Based Incremental Timing Driven Placement for High Performance Designs". In: *Proceedings of the 43rd Design Automation Conference*. DAC, pp. 1115–1120. DOI: 10.1145/1146909.1147190 (cit. on p. 17).

I. L. Markov, J. Hu and M.-C. Kim (2015). "Progress and Challenges in VLSI Placement Research". In: *Proceedings of the IEEE* 103.11, pp. 1985–2003. DOI: 10.1109/JPROC.2015.2478963 (cit. on pp. 16, 17).

N. Megiddo (1983). "Applying Parallel Computation Algorithms in the Design of Serial Algorithms". In: *Journal of the ACM (JACM)* 30.4, pp. 852–865. DOI: 10.1145/2157.322410 (cit. on p. 33).

F. Michaelis (2015). "Verbesserungen von Algorithmen für die Platzierung von Macros". German. Bachelor's Thesis. Research Institute for Discrete Mathematics, University of Bonn (cit. on pp. 53, 58).

M. D. Moffitt, A. N. Ng, I. L. Markov and M. E. Pollack (2006). "Constraint-Driven Floorplan Repair". In: *Proceedings of the 43rd Design Automation Conference*. DAC, pp. 1103–1108. DOI: 10.1145/1146909.1147188 (cit. on p. 14).

E. F. Moore (1959). "The shortest path through a maze". In: *Proceedings of the International Symposium on Switching Theory*. Vol. II, pp. 285–292 (cit. on p. 37).

D. Müller, K. Radke and J. Vygen (2011). "Faster Min–Max Resource Sharing in Theory and Practice". In: *Mathematical Programming Computation* 3.1, pp. 1–35. DOI: 10.1007/s12532-011-0023-y (cit. on pp. 17, 92, 106).

H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani (1996). "VLSI Module Placement Based on Rectangle-Packing by the Sequence-Pair". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15.12, pp. 1518–1524. DOI: 10.1109/43.552084 (cit. on p. 14).

S. Nakatake, K. Fujiyoshi, H. Murata and Y. Kajitani (1996). "Module Placement on BSG-structure and IC Layout Applications". In: *Proceedings of the 1996 International Conference on Computer-Aided Design*, pp. 484–491. DOI: 10.1109/ICCAD.1996.569870 (cit. on p. 14).

G.-J. Nam (2006). "ISPD 2006 Placement Contest: Benchmark Suite and Results". In: *Proceedings of the 2006 International Symposium on Physical Design*. ISPD (San Jose, California, USA), pp. 167–167. DOI: 10.1145/1123008.1123042 (cit. on p. 7).

G.-J. Nam, C. J. Alpert, P. Villarrubia, B. Winter and M. Yildiz (2005). "The ISPD 2005 Placement Contest and Benchmark Suite". In: *Proceedings of the 2005 International Symposium on Physical Design*. ISPD (San Francisco, California, USA), pp. 216–220. DOI: 10.1145/1055137.1055182 (cit. on p. 7).

G.-J. Nam and J. Cong, eds. (2007). *Modern Circuit Placement: Best Practices and Results*. DOI: 10.1007/978-0-387-68739-1_10 (cit. on p. 16).

W. Naylor, R. Donelly and L. Sha (2001). "Non-linear optimization system and method for wire length and delay optimization for an automatic electric circuit placer". US6301693B1 (cit. on pp. 15, 16).

H. Onodera, Y. Taniguchi and K. Tamaru (1991). "Branch-and-bound Placement for Building Block Layout". In: *Proceedings of the 28th Design Automation Conference* (San Francisco, California, USA), pp. 433–439. DOI: 10.1145/127601.127708 (cit. on p. 14).

R. H. J. M. Otten (1998). "Global Wires: Harmful?" In: *Proceedings of the 1998 International Symposium on Physical Design*. ISPD (Monterey, California, USA), pp. 104–109. DOI: 10.1145/274535.274550 (cit. on pp. 9, 10).

D. Z. Pan, B. Halpin and H. Ren (2008). "Timing-Driven Placement". In: *Handbook of Algorithms for Physical Design Automation*. Ed. by C. J. Alpert, D. P. Mehta and S. S. Sapatnekar, pp. 423–446 (cit. on p. 17).

C. L. Ratzlaff and L. T. Pillage (1994). "RICE: Rapid Interconnect Circuit Evaluation Using AWE". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.6, pp. 763–776. DOI: 10.1109/43.285250 (cit. on p. 10).

J. A. Roy, S. N. Adya, D. A. Papa and I. L. Markov (2006). "Min-Cut Floorplacement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.7, pp. 1313–1326. DOI: 10.1109/TCAD.2005.855969 (cit. on pp. 12, 16, 17).

J. A. Roy and I. L. Markov (2007). "Seeing the Forest and the Trees: Steiner Wirelength Optimization in Placement". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26.4, pp. 632–644. DOI: 10.1109/TCAD.2006.888260 (cit. on p. 17).

S. S. Sapatnekar (2004). *Timing* (cit. on p. 10).

H. E. Scarf (1981a). "Production Sets with Indivisibilities, Part I: Generalities". In: *Econometrica* 49.1, pp. 1–32. DOI: 10.2307/1911124 (cit. on p. 72).

H. E. Scarf (1981b). "Production Sets with Indivisibilities, Part II: The Case of Two Activities". In: *Econometrica* 49.2, pp. 395–423. DOI: 10.2307/1913318 (cit. on p. 72).

J. Schneider (2009). "Macro Placement in VLSI-Design". Diploma Thesis. Research Institute for Discrete Mathematics, University of Bonn (cit. on pp. 47, 55, 57).

D. Shmoys and É. Tardos (1993). "An approximation algorithm for the generalized assignment problem". In: *Mathematical Programming* 62, pp. 461–474. DOI: 10.1007/BF01585178 (cit. on p. 76).

J. Silvanus and J. Vygen (2017). *Few Sequence Pairs Suffice. Representing All Rectangle Placements*. arXiv: 1708.09779 [math.CO] (cit. on p. 14).

P. Spindler, U. Schlichtmann and F. M. Johannes (2008). "Kraftwerk2 – A Fast Force-Directed Quadratic Placement Approach Using an Accurate Net Model". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.8, pp. 1398–1411. DOI: 10.1109/TCAD.2008.925783 (cit. on pp. 12, 15, 76).

A. Srinivasan, K. Chaudhary and E. S. Kuh (1992). "RITUAL: A Performance Driven Placement Algorithm". In: *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 39.11, pp. 825–840. DOI: 10.1109/82.204130 (cit. on p. 17).

M. Struzyna (2010). "Flow-based Partitioning and Fast Global Placement in Chip Design". Dissertation. Research Institute for Discrete Mathematics, University of Bonn (cit. on p. 77).

M. Struzyna (2011). "Flow-based partitioning and position constraints in VLSI placement". In: *Proceedings of the 2011 Conference on Design, Automation and Test in Europe*. DATE (Grenoble, France), pp. 1–6. DOI: 10.1109/DATE.2011.5763100 (cit. on pp. 75–79, 81, 89, 99).

M. Struzyna (2013). "Sub-Quadratic Objectives in Quadratic Placement". In: *Proceedings of the 2013 Conference on Design, Automation and Test in Europe*. DATE (Grenoble, France), pp. 1867–1872. DOI: 10.7873/DATE.2013.372 (cit. on pp. 16, 75, 76, 89, 106).

T.-Y. Sun, S.-T. Hsieh, H.-M. Wang and C.-W. Lin (2006). "Floorplanning based on Particle Swarm Optimization". In: *Proceedings of the 2006 Annual Symposium on Emerging VLSI Technologies and Architectures*. ISVLSI (Karlsruhe, Germany), pp. 7–11. DOI: 10.1109/ISVLSI.2006.46 (cit. on p. 14).

S. Sutanthavibul, E. Shragowitz and J. B. Rosen (1990). "An Analytical Approach to Floorplan Design and Optimization". In: *Proceedings of the 27th Design Automation Conference* (Orlando, Florida, USA), pp. 187–192. DOI: 10.1145/123186.123255 (cit. on p. 62).

C. Szegedy (2005). "Some Applications of the Weighted Combinatorial Laplacian". Dissertation. Research Institute for Discrete Mathematics, University of Bonn (cit. on p. 17).

T. Takahashi (2000). "A New Encoding Scheme for Rectangle Packing Problem". In: *Proceedings of the 37th Design Automation Conference*. DAC (Los Angeles, California, United States), pp. 175–178. DOI: 10.1109/ASPDAC.2000.835091 (cit. on p. 14).

M. Tang and A. Sebastian (2005). "A Genetic Algorithm for VLSI Floorplanning Using O-Tree Representation". In: *Applications of Evolutionary Computing. Proceedings of the EvoWorkshops 2005*. Ed. by F. Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. D. Smith and G. Squillero, pp. 215–224. DOI: 10.1007/978-3-540-32003-6_22 (cit. on p. 14).

R. E. Tarjan (1975). "Efficiency of a Good But Not Linear Set Union Algorithm". In: *Journal of the ACM (JACM)* 22.2, pp. 215–225. DOI: 10.1145/321879.321884 (cit. on p. 84).

N. Tomizawa (1971). "On Some Techniques Useful for Solution of Transportation Network Problems". In: *Networks. An International Journal* 1.2, pp. 173–194. DOI: 10.1002/net.3230010206 (cit. on p. 89).

N. Viswanathan, C. J. Alpert, C. Sze, Z. Li and Y. Wei (2012). "The DAC 2012 Routability-Driven Placement Contest and Benchmark Suite". In: *Proceedings of the 49th Design Automation Conference*. DAC (San Francisco, California, USA), pp. 774–782. DOI: 10.1145/2228360.2228500 (cit. on pp. 103, 106, 108, 122).

N. Viswanathan, G.-J. Nam, C. J. Alpert, P. G. Villarrubia, H. Ren and C. Chu (2007). "RQL: Global Placement via Relaxed Quadratic Spreading and Linearization". In: *Proceedings of the 44th Design Automation Conference*. DAC (San Diego, California, USA), pp. 453–458. DOI: 10.1145/1278480.1278599 (cit. on p. 15).

N. Viswanathan, M. Pan and C. Chu (2006). "FastPlace 2.0: An Efficient Analytical Placer for Mixed-mode Designs". In: *Proceedings of the 11th Asia and South Pacific Design Automation*

Conference. ASP-DAC (Yokohama, Japan), pp. 195–200. DOI: 10.1145/1118299.1118354 (cit. on pp. 12, 15).

N. Viswanathan, M. Pan and C. Chu (2007). "FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion Control". In: *Proceedings of the 12th Asia and South Pacific Design Automation Conference*. ASP-DAC (Yokohama, Japan), pp. 135–140. DOI: 10.1109/ASPDAC.2007.357975 (cit. on pp. 12, 15, 16).

C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran and J. G. Hemmett (2006). "First-Order Incremental Block-Based Statistical Timing Analysis". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25.10, pp. 2170–2180. DOI: 10.1109/TCAD.2005.862751 (cit. on p. 10).

J. Vygen (2005). "Geometric quadrisection in linear time, with application to VLSI placement". In: *Discrete Optimization* 2.4, pp. 362–390. DOI: 10.1016/j.disopt.2005.08.007 (cit. on p. 76).

Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy, A. D. Huber, G. E. Tellez, D. Keller and S. S. Sapatnekar (2012). "GLARE: Global and Local Wiring Aware Routability Evaluation". In: *Proceedings of the 49th Design Automation Conference*. DAC (San Francisco, California, USA), pp. 768–773. DOI: 10.1145/2228360.2228499 (cit. on p. 104).

D. M. Wochnik (2017). "Verbesserte Algorithmen für die Platzierung von Makros". Master's Thesis. Research Institute for Discrete Mathematics, University of Bonn (cit. on pp. 56, 59, 73).

G. Wu and C. Chu (2017). "Two Approaches for Timing-Driven Placement by Lagrangian Relaxation". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36.12, pp. 2093–2105. DOI: 10.1109/TCAD.2017.2697947 (cit. on p. 17).

Z. Xiu and R. Rutenbar (2007). "Mixed-Size Placement with Fixed Macrocells using Grid-Warping". In: *Proceedings of the 2007 International Symposium on Physical Design*. ISPD (Austin, Texas, USA), pp. 103–110. DOI: 10.1145/1231996.1232019 (cit. on p. 16).

Y. Xu, Y. Zhang and C. Chu (2009). "FastRoute 4.0: Global Router with Efficient via Minimization". In: *Proceedings of the 14th Asia and South Pacific Design Automation Conference*. ASP-DAC (Yokohama, Japan), pp. 576–581. DOI: 10.1145/1509633.1509768 (cit. on p. 17).

J. Z. Yan, N. Viswanathan and C. Chu (2014). "An Effective Floorplan-Guided Placement Algorithm for Large-Scale Mixed-Size Designs". In: *ACM Transactions on Design Automation of Electronic Systems* 19.3, Art. No. 29. DOI: 10.1145/2611761 (cit. on p. 14).

E. F. Y. Young (2008). "Floorplan Representations". In: *Handbook of Algorithms for Physical Design Automation*. Ed. by C. J. Alpert, D. P. Mehta and S. S. Sapatnekar, pp. 185–202 (cit. on p. 14).

N. E. Young, R. E. Tarjan and J. B. Orlin (1991). "Faster Parametric Shortest Path and Minimum-Balance Algorithms". In: *Networks. An International Journal* 21.2, pp. 205–221. DOI: 10.1002/net.3230210206 (cit. on p. 33).

K. Zhong and S. Dutt (2002). "Algorithms for Simultaneous Satisfaction of Multiple Constraints and Objective Optimization in a Placement Flow with Application to Congestion Control". In: *Proceedings of the 2002 Design Automation Conference*. DAC (New Orleans, Louisiana, United States), pp. 854–859. DOI: 10.1109/DAC.2002.1012741 (cit. on p. 17).

H. Zhou and J. Wang (2004). "ACG-Adjacent Constraint Graph for General Floorplans". In: *Proceedings of the International Conference on Computer Design: VLSI in Computers and Processors*. ICCD (Austin, Texas, USA), pp. 572–575. DOI: 10.1109/ICCD.2004.1347980 (cit. on p. 14).

C. Zhuang, Y. Kajitani, K. Sakanushi and L. Jin (2002–2002). "An Enhanced Q-Sequence Augmented with Empty-Room-Insertion and Parenthesis Trees". In: *Proceedings of the 2002 Conference on Design, Automation and Test in Europe*. DATE (Paris, France), pp. 61–68. DOI: 10.1109/DATE.2002.998250 (cit. on p. 14).

Index

Page numbers in bold face (as in **42**) point to a definition or explanation of a keyword. In contrast, page numbers in serif face (as in 42) indicate an occurrence of a term.

Notation

$\triangle$	Above of spatial relation (Definition 2.14, page 12).
$\measuredangle_l(v_e)$	Angle $\theta \in [0, 2\pi]$ of $e = \{v, w\} \in \mathrm{E}(G)$ w.r.t. locations l (Definition 3.21, page 28).
$\mathrm{at}(v)$	Arrival time of node v in the fine timing graph (page 9).
$\triangledown$	Below of spatial relation (Definition 2.14, page 12).
b	Length constraints of distance bounds (Definition 3.4, page 22).
$\mathrm{BB}(P)$	Bounding box of finite $P \subseteq \mathbb{R}^2$ (Definition 2.5, page 4).
BBL	Bounding box netlength (Definition 2.13, page 7).
$\mathrm{A}(c)$	Cell shape of cell c (Definition 2.7, page 4).
$\square$	Chip image (Definition 2.6, page 4).
C	Circuits of a netlist (Definition 2.6, page 4).
γ	$\gamma\colon P \to C \cup \{\square\}$ (Definition 2.6, page 4).
$\mathrm{d_c}(P)$	Gate-internal delay of path P in the fine timing graph (page 10).
$\mathrm{d}(P, l)$	Delay of path P in the fine timing graph acc. to placement l (page 10).
$\mathrm{d_w}(P, l)$	Wiring delay of path P in the fine timing graph acc. to placement l (page 10).
$\delta^-(v)$	Set of entering edges of node v in a directed graph.
$\delta^+(v)$	Set of leaving edges of node v in a directed graph.
$\delta(v)$	Set of incident edges of node v in an undirected graph.
G_b	Shorthand for distance bounds $(G, \mathrm{b}, \mathrm{o})$ (Definition 3.4, page 22).
$\mathrm{D}(G)$	Distance graph for distance bounds G_b (Definition 3.28, page 33).
$\mathrm{dgc}(C)$	Distance graph cycle in $\mathrm{D}(G)$ for cycles C or edges in G (page 34).
f_λ^d	Distance graph function in direction d (Definition 3.28, page 33).
$\mathrm{E_c}$	Gate-internal edges of the fine timing graph (page 10).
$\mathrm{E}(G)$	Edges of graph G.
$\mathrm{E_w}$	Wiring edges of the fine timing graph (page 10).
$\triangleleft$	Left of spatial relation (Definition 2.14, page 12).
$\mathrm{msd}_Q(p)$	Maximum shifted point distance (Definition 3.38, page 39).
$\mathrm{mr}\,(G, f_\lambda)$	Optimum ratio of (G, f_λ) instance of the Minimum Ratio Cycle Problem (page 32).
μ	Movebound map $\mu\colon C \to M$ (Definition 2.11, page 6).
m'	Number of distinct, connected endpoint pairs (page 65).
m	Number of edges $m := \lvert\mathrm{E}(G)\rvert$ of a graph G.
N	Nets of a netlist (Definition 2.6, page 4).
n	Number of vertices $n := \lvert\mathrm{V}(G)\rvert$ of a graph G.
$\mathcal{O}(f)$	$\mathcal{O}$-Notation: $g \in \mathcal{O}(f)$ iff $\exists \alpha, \beta\colon \lvert g(n)\rvert \leq \alpha f(n) + \beta$.

o	Offsets of distance bounds (Definition 3.4, page 22).
$\Omega(f)$	Ω-Notation: $g \in \Omega(f)$ iff $f \in \mathcal{O}(g)$.
off (p)	Pin offset of pin p (Definition 2.7, page 4).
P	Pins of a netlist (Definition 2.6, page 4).
$l(c)$	Placement $l\colon C \to \mathbb{R}^2$ (Definition 2.8, page 4).
$\mathcal{P}(X)$	Power set of X.
$[\![n]\!]$	The first $n \in \mathbb{N}$ integers $\{\, i \in \mathbb{N} : 0 \leq i < n \,\} = [0, n) \cap \mathbb{N}$.
rat(v)	Required arrival time of node v in the fine timing graph (page 9).
$\overline{e}$	Reverse edge of $e \in \mathrm{D}(G)$ (page 33).
$\rhd$	Right of spatial relation (Definition 2.14, page 12).
σ_z^d	Indicator of coordinate $z \in \{\, \mathrm{x}, \mathrm{y} \,\}$ in direction d (Definition 3.28, page 33).
slack(v)	Slack at node v in the fine timing graph (page 9).
sWL	Scaled wirelength metric (page 106).
v_e	Shorthand for (v, e) for undirected edge e and $v \in e$ (page 24).
$\mathrm{V}(G)$	Vertices of graph G.
wACE4	Wiring-area average congestion metric (page 4).