Combinatorial Optimization in VLSI Design

Stephan Held, Bernhard Korte, Dieter Rautenbach, and Jens Vygen

Abstract VLSI design is probably the most fascinating application area of combinatorial optimization. Virtually all classical combinatorial optimization problems, and many new ones, occur naturally as subtasks. Due to the rapid technological development and major theoretical advances the mathematics of VLSI design has changed significantly over the last ten to twenty years. This survey paper gives an up-to-date account on the key problems in layout and timing closure. It also presents the main mathematical ideas used in a set of algorithms called BonnTools, which are used to design many of the most complex integrated circuits in industry.

Keywords. combinatorial optimization, VLSI, physical design, layout, placement, routing, timing optimization, clock tree synthesis

1. Introduction

The ever increasing abundance, role and importance of computers in every aspect of our lives is clearly a proof of a tremendous scientific and cultural development - if not revolution. When thinking about the conditions which made this development possible most people will probably first think mainly of technological aspects such as the invention and perfection of transistor technology, the possibility to fabricate smaller and smaller physical structures consisting of only a few atoms by now, and the extremely delicate, expensive yet profitable manufacturing processes delivering to the markets new generations of chips in huge quantities every couple of months. From this point of view the increase of complexity might be credited mainly to the skills of the involved engineering sciences and to the verve of the associated economic interests.

It is hardly conceived how important mathematics and especially mathematical optimization is for all parts of VLSI technology. Clearly, everybody will acknowledge that the physics of semiconductor material relies on mathematics and that, considered from a very abstract level, computer chips are nothing but intricate machines for the calculation of complex Boolean functions. Nevertheless, the role of mathematics is far from being fully described with these comments. Especially the steps of the design of a VLSI chip preceding its actual physical realization involve more and more mathematics. Many of the involved tasks which were done by the hands of experienced engineers until one or two decades ago have become so complicated and challenging that they can only be solved with highly sophisticated algorithms using specialized mathematics.

While the costs of these design and planning issues are minor compared to the investments necessary to migrate to a new technology or even to build a single new chip factory, they offer large potentials for improvement and optimization. This and the fact that the arising optimization problems, their constraints and objectives, can be

captured far more exactly in mathematical terms than many other problems arising in practical applications, make VLSI design one of the most appealing, fruitful and successful application areas of mathematics.

The Research Institute for Discrete Mathematics at the University of Bonn has been working on problems arising in VLSI design for more than twenty years. Since 1987 there exists an intensive and growing cooperation with IBM, in the course of which more than one thousand chips of IBM and its customers (microprocessor series, application specific integrated circuits (ASICs), complex system-on-a-chip designs (SoC)) have been designed with the so-called BonnTools. In 2005 the cooperation was extended to include Magma Design Automation. Some BonnTools are now also part of Magma's products and are used by its customers.

The term BonnTools [51] refers to complete software solutions which have been developed at the institute in Bonn and are being used in many design centers all over the world. The distinguishing feature of BonnTools is their innovative mathematics. With its expertise in combinatorial optimization [50,52,26] the institute was able to develop some of the best algorithms for the main VLSI design tasks: placement, timing optimization, distribution of the clocking signals, and routing. Almost all classical combinatorial optimization problems such as shortest paths, minimum spanning trees, maximum flows, minimum cost flows, facility location and so forth arise at some stage in VLSI design, and the efficient algorithms known in the literature for these problems can be used to solve various subproblems in the design flow. Nevertheless, many problems do not fit into these standard patterns and need new customized algorithms. Many such algorithms have been developed by our group in Bonn and are now part of the IBM design flow.

In this paper we survey the main mathematical components of BonnTools. It is a common feature of these components that they try to restrict the optimization space, i.e. the set of feasible solutions which the algorithms can generate, as little as possible. This corresponds to what is typically called a flat design style in contrast to a hierarchical design style. The latter simplifies a problem by splitting it into several smaller problems and restricting the solution space by additional constraints which make sure that the partial solutions of the smaller problems properly combine to a solution of the entire problem. Clearly, this can seriously deteriorate the quality of the generated solution.

While imposing as few unnecessary restrictions to the problems as possible, the BonnTools algorithms are always considered with respect to their theoretical as well as practical performance. Wherever possible, theoretical performance guarantees and rigorous mathematical proofs are established. The running time and practical behavior of the implemented algorithms is always a main concern, because the code is used for real practical applications.

The beauty and curse of applying mathematics to VLSI design is that problems are never solved once for good. By new technological challenges, new orders of magnitude in instance sizes, and new foci on objectives like the reduction of power consumption for portable devices or the increase of the productivity of the factories, new mathematical problems arise constantly and classical problems require new solutions. This makes this field most interesting not only for engineers, but also for mathematicians. See [53] for some interesting open problems.

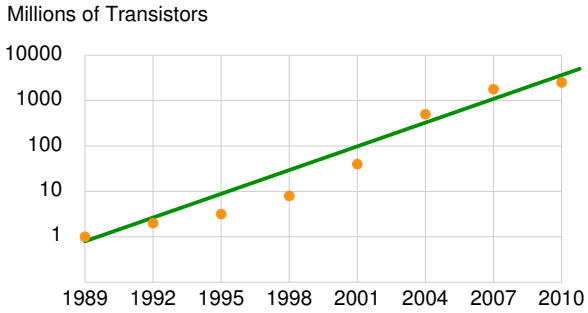

Figure 1. Number of transistors per logic chip

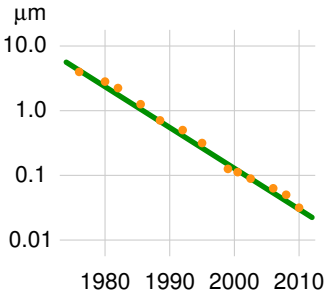

Figure 2. Feature sizes in micron

The paper is organized as follows. In the rest of this introduction we explain some basic terminology of VLSI technology and design. Then, in Section 2, we describe our placement tool BonnPlace and its key algorithmic ingredients. The placement problem is solved in two phases: global and detailed placement. Global placement uses continuous quadratic optimization and a new combinatorial partition algorithm (multisection). Detailed placement is based on a sophisticated minimum cost flow formulation.

In Section 3 we proceed to timing optimization, where we concentrate on the three most important topics: repeater trees, logic restructuring, and choosing physical realizations of gates (sizing and V_t-assignment). These are the main components of BonnOpt, and each uses new mathematical theory.

As described in Section 4, BonnCycleOpt further optimizes the timing and robustness by enhanced clock skew scheduling. It computes a time interval for each clock input of a memory element. BonnClock, our tool for clock tree synthesis, constructs clock trees meeting these time constraints and minimizing power consumption.

Finally, Section 5 is devoted to routing. Our router, BonnRoute, contains the first global router that directly considers timing, power consumption, and manufacturing yield, and is provably close to optimal. It is based on a new, faster algorithm for the min-max resource sharing problem. The unique feature of our detailed router is an extremely fast implementation of Dijkstra's shortest path algorithm, allowing us to find millions of shortest paths even for long-distance connections in very reasonable time.

1.1. A brief guided tour through VLSI technology

VLSI — very large-scale integrated — chips are by far the most complex structures invented and designed by man. They can be classified into two categories: memory chips and logic chips. In a memory chip transistors are packed into a rectangular array. For the design of such chips no advanced mathematics is needed since the individual storage elements (transistors) have to be arranged like a matrix. Logic chips have a very individual design where mathematical methods — as explained below — are essential.

Integrated circuits are around since 1959. Jack Kilby of Texas Instruments was one of its inventors. Since then the degree of integration grew exponentially. While the first integrated circuits had only a few transistors on a silicon chip, modern chips can have up to one million transistors per mm^2, i.e. a chip of 2 cm^2 total size can carry up to 2 billion transistors. The famous Moore's law, a rule of thumb proposed by Gordon Moore in 1965 [61] and updated in 1975, states that the number of transistors per chip doubles every 24 months (see Figure 1).

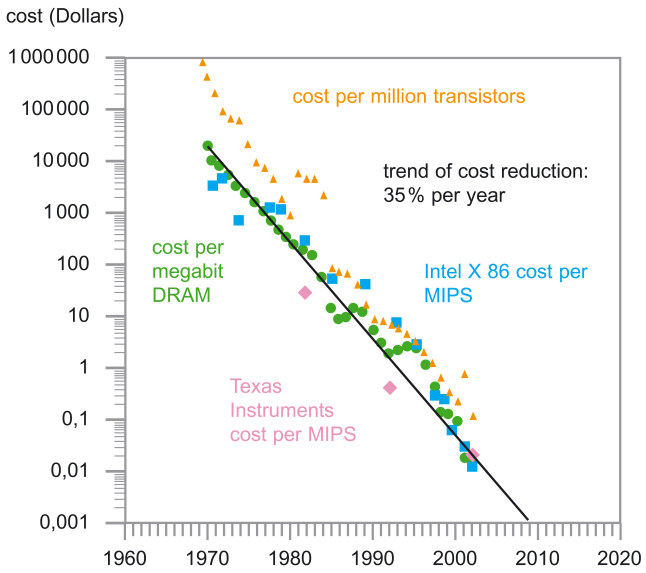

Figure 3. Trend of cost reduction in microelectronics

This empirical observation is true ever since. As the size of a chip remains almost constant (between 1 and 4 cm^2), the minimum feature size on a chip has to halve about every 4 years. See Figure 2 for the development of feature sizes on leading-edge computer chips.

It is frequently asked how this extremely rapid development of chip technology will continue. Technological as well as physical limitations have to be considered. However, technological limitations could be overruled so far by more sophisticated manufacturing approaches. Thus, the quite often predicted end of silicon technology is not yet in sight. Certainly, there are genuine physical limitations. Today less than 100 000 electrons are used to represent one bit, the absolute minimum is one. The switching energy of a single transistor amounts nowadays to 10 000 attojoule (atto = 10^{-18}). The lower bound derived from quantum physics is 0.000001 attojoule. Some experts believe that the limit of feature size is around 5 nanometers (today 32 nanometers) and they predict that such dimensions will be possible between 2020 and 2025. In any case, silicon technology will be alive for some further decades. There is some interesting (theoretical) research on quantum computing. However, nobody knows when such ideas can be used in hardware and for mass production.

The extreme dynamics of chip technology can be demonstrated by cost reduction over time. In Figure 3 the trend of cost reduction is shown from 1960 on. Green dots demonstrate the cost of 1 megabit of memory (DRAM), red triangles the cost of 1 million transistors in silicon technology, blue squares the cost of 1 MIPS (million instructions per second) computing power of Intel X86 processors and red diamonds show the cost of 1 MIPS for Texas Instruments processors. All these developments indicate an average reduction of cost of 35 % per year. There is no other industry or technology known with such huge figures over a period of 50 years.

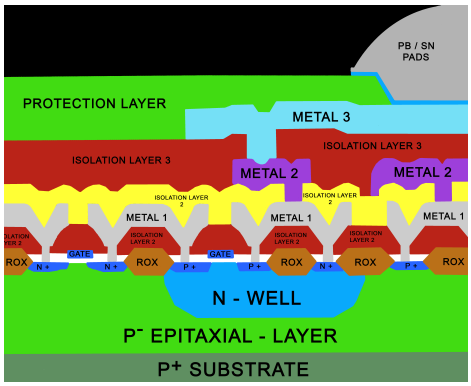

Figure 4. Schematic cross-sectional view of a chip

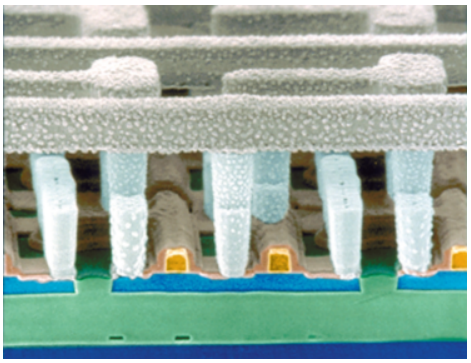

Figure 5. Microscopic view of a chip with aluminum wiring

Let us give some insight into the real structure of a chip. Figure 4 shows a schematic of a chip and its different layers. This diagram refers to an older technology with only three layers of metal interconnect. Modern chips have up to 12 layers for wiring signal nets between the circuits. The cross-sectional view reveals the different layers generated on the chip by lithographic processes. By doping with foreign atoms so-called wells (N-wells and P-wells) are generated on the silicon substrate of the wafer. According to the doping the regions have either a surplus (emitter zones) or a demand (collector zones) of electrons. The space between these regions is controlled by a gate. The gates can be charged with different electrical potentials. This will effect that the space underneath the gate is blocked or that electrons can move, which means that the transistor as an electronic switch is either closed or open.

Figure 5 displays the structure of a real chip, visualized by a scanning tunneling microscope. We can identify emitter, collector and gates with their connector pins and some horizontal part of the connecting wires. This picture shows an older technology with aluminum as metal for the interconnect. Figure 6 shows the same technology. Each layer contains horizontal and/or vertical wires, and adjacent layers are separated by an insulation medium. One can also see vias, i.e. the connections between different metal layers. Vias can be considered as little holes in the insulating layer, filled with metal. Since approximately ten years ago aluminum has been replaced by copper for the interconnect wiring (Figure 7). This permits faster signal transmission.

1.2. The VLSI design problem: a high-level view

Although all functions of a chip are composed of transistors and their interconnect, it is not useful to work on this level directly. Instead one uses a library for the design of a chip, where each element of the library represents a pre-designed configuration of several transistors, implementing a specific function. The elements of the library are called books.

Each book is a blueprint of a (smaller) integrated circuit itself: it contains a list of transistors and their layout, including internal interconnect. Most importantly, each book has at least one input and at least one output. Most books implement an elementary Boolean function, such as `and`, `or`, `invert`, `xor`, etc. For example, the output of an `and` is charged (logical 1) if both inputs are charged, and discharged (logical 0) otherwise. Other books implement registers.

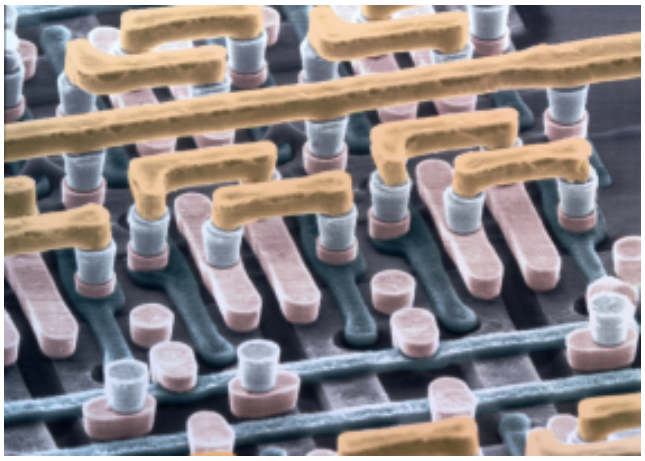

Figure 6. Layers of a chip visualized by electron microscopy

Figure 7. Microscopic view of a chip with copper-wiring (90 nm technology)

Such simple cells are called *standard cells*. Their blueprints have unit height, because on the chip these cells have to be aligned vertically into so called *cell rows*. Thereby, the power supply pins in each cell row are also aligned and power supply wires can be arranged in straight horizontal lines across the chip.

Finally, there are complicated (and larger) books that represent complex structures like memory arrays, adders, or even complete microprocessors that have been designed earlier and are re-used several times. With each book there is also pre-computed information about its timing behavior. A simple view is that we know how long it takes that a change of an input bit is propagated to each output (if it has any effect).

A chip can contain many instances of the same book. These instances are called circuits or cells. For example, a logic chip can contain millions of inverters, but a typical library contains only a few dozen books that are different implementations of the `invert` function. These books have different layouts and different timing behavior, although they all implement the same function. Each circuit has a set of pins, and each of these pins corresponds to an input or an output of the corresponding book.

The most important part of an instance of the VLSI design problem is a *netlist*, which consists of a set of circuits, their pins, a set of additional pins that are inputs or outputs of the chip itself (I/O ports), and a set of nets, which are pairwise disjoint sets of pins. The layout problem consists of placing these circuits within the chip area, without any overlaps, and connecting the pins of each net by wires, such that wires of different nets are well separated from each other. Placement (Section 2) is a two-dimensional problem (it is currently not possible to put transistors on top of each other), but routing (Section 5) is a three-dimensional problem as there are several (currently up to 12) layers that can be used for wiring. Of course there are additional rules for placement and routing that must be followed. Some of these are important for the nature of the problem and will be discussed later on, others are merely technical but cause no algorithmic problems; these will not be discussed in this paper.

Layout (placement and routing) is not the only interesting design task. Usually one of the main challenges is to meet timing constraints. In particular, all signals must be propagated in time not only through a single circuit, but also through long paths. In

a simple setting, we have an arrival time at each input of the chip, and also a latest feasible arrival time at each output. Moreover, each register is controlled by a periodic clock signal, and the signal to be stored in the register must arrive in time, and can then be used for further computations. To make this possible, one can replace parts of the netlist equivalently (the new parts must compute the same Boolean function as the old ones). While the task to implement a given Boolean function optimally by a netlist (logic synthesis) is extremely hard and more or less completely unsolved, we concentrate on replacing smaller parts of the netlist or restrict to basic operations (Section 3). Another possibility to speed up timing is to schedule the clock signals for all registers (Section 4), thereby trading timing constraints of paths. This is one of the few tasks which we can solve optimally, even for the largest VLSI instances.

2. Placement

A chip is composed of basic elements, called cells, circuits, boxes, or modules. They usually have a rectangular shape, contain several transistors and internal connections, and have at least two pins (in addition to power supply). The pins have to be connected to certain pins of other cells by wires according to the netlist. A net is simply a set of pins that have to be connected, and the netlist is the set of all nets.

The basic placement task is to place the cells legally — without overlaps — in the chip area. A feasible placement determines an instance of the routing problem, which consists of implementing all nets by wires. The quality of a placement depends on the quality of a wiring that can be achieved for this instance.

For several reasons it is usually good if the wire length (the total length of the wires connecting the pins of a net) is as short as possible. The power consumption of a chip grows with the length of the interconnect wires, as higher electrical capacitances have to be charged and discharged. For the same reason signal delays increase with the wire length. Critical nets should be kept particularly short.

2.1. Estimating net length

An important question is how to measure (or estimate) wire length without actually routing the chip. First note that nets are wired in different layers with alternating orthogonal preference direction. Therefore the ℓ_1-metric is the right metric for wire length. An exact wire length computation would require to find disjoint sets of wires for all nets (vertex-disjoint Steiner trees), which is an *NP*-hard problem. This even holds for the simplified problem of estimating the length of each net by a shortest two-dimensional rectilinear Steiner tree connecting the pins, ignoring disjointness and all routing constraints [31].

A simple and widely used estimate for the net length of a finite set $V \subset \mathbb{R}^2$ of pin coordinates (also called terminals) is the *bounding box* model BB, which is defined as half the perimeter of the bounding box:

$$\mathrm{BB}(V) = \max_{(x,y)\in V} x - \min_{(x,y)\in V} x + \max_{(x,y)\in V} y - \min_{(x,y)\in V} y$$

The bounding box net length is a lower bound for the minimum Steiner tree length and computable in linear time. It is widely used for benchmarking and often also as an

	BB	STEINER	MST	CLIQUE	STAR
BB	1	1	1	1	1
STEINER	$\frac{n-1}{\lceil\sqrt{n}\rceil+\left\lceil\frac{n}{\lceil\sqrt{n}\rceil}\right\rceil-2}$ $\cdots$ $\frac{\lceil\sqrt{n-2}\rceil}{2}+\frac{3}{4}$	1	1	$\begin{cases}\frac{9}{8} & (n=4)\\ 1 & (n\neq 4)\end{cases}$	1
MST	$\lfloor\frac{\sqrt{2n-1}+1}{2}\rfloor$ $\cdots$ $\frac{\sqrt{n}}{\sqrt{2}}+\frac{3}{2}$	$\frac{3}{2}$	1	$1+\Theta\left(\frac{1}{n}\right)$ $\cdots$ $\frac{3}{2}$	$\begin{cases}\frac{4}{3} & (n=3)\\ \frac{3}{2} & (n=4)\\ \frac{6}{5} & (n=5)\\ 1 & (n>5)\end{cases}$
CLIQUE	$\frac{\lceil\frac{n}{2}\rceil\lfloor\frac{n}{2}\rfloor}{n-1}$	$\frac{\lceil\frac{n}{2}\rceil\lfloor\frac{n}{2}\rfloor}{n-1}$	$\frac{\lceil\frac{n}{2}\rceil\lfloor\frac{n}{2}\rfloor}{n-1}$	1	1
STAR	$\lfloor\frac{n}{2}\rfloor$	$\lfloor\frac{n}{2}\rfloor$	$\lfloor\frac{n}{2}\rfloor$	$\frac{n-1}{\lceil\frac{n}{2}\rceil}$	1

Table 1. Worst-case ratios of major net models. Entry (r, c) is $\sup \frac{c(N)}{r(N)}$ over all point sets N with $|N| = n$. Here $c(N)$ denotes a net length in the model of column c and $r(N)$ in the model of row r.

objective function in placement. Other useful measurements are the clique model CLIQUE which considers all pin to pin connections of a net

$$\text{CLIQUE}(V) = \frac{1}{|V|-1} \sum_{\{(x,y),(x',y')\}\in\binom{V}{2}} (|x-x'|+|y-y'|),$$

and the star model which is the minimum length of a star connecting all sinks to an optimally placed auxiliary point. It can be shown that the clique model is the best topology-independent approximation of the minimum Steiner length [15]. Therefore we use it in our optimization framework, which we will present in Section 2.3.

Table 1 gives an overview on major net models and their mutual worst-case ratios. They were proved in [15], [42], and [75].

2.2. The placement problem

We now define the *Simplified Placement Problem*. It is called "simplified" as side constraints such as routability, timing constraints, decoupling capacitor densities, or nwell filling are neglected [1]. Moreover, wire length is estimated by the bounding box model. Nevertheless this formulation is very relevant in practice. Net weights are incorporated to reflect timing criticalities and can be interpreted as Lagrange multipliers corresponding to delay constraints. Other constraints can be dealt with by adjusting densities.

[1] Readers who are not acquainted with these terms might just think of additional constraints.

SIMPLIFIED PLACEMENT PROBLEM

Instance:

- a rectangular chip area $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$
- a set of rectangular blockages
- a finite set C of (rectangular) cells
- a finite set P of pins, and a partition $\mathcal{N}$ of P into nets
- a weight $w(N) > 0$ for each net $N \in \mathcal{N}$
- an assignment $\gamma : P \to C \cup \{\square\}$ of the pins to cells
  [pins p with $\gamma(p) = \square$ are fixed; we set $x(\square) := y(\square) := 0$]
- offsets $x(p), y(p) \in \mathbb{R}$ of each pin $p \in P$

Task: Find a position $(x(c), y(c)) \in \mathbb{R}^2$ of each cell $c \in C$ such that

- each cell is contained in the chip area,
- no cell overlaps with another cell or a blockage,

and the weighted net length

$$\sum_{N \in \mathcal{N}} w(N) \operatorname{BB}(\{(x(\gamma(p)) + x(p), y(\gamma(p)) + y(p)) \mid p \in N\})$$

is minimum.

A special case of the *Simplified Placement Problem* is the QUADRATIC ASSIGNMENT PROBLEM (QAP), which is known to be one of the hardest combinatorial optimization problems in theory and practice (for example, it has no constant-factor approximation algorithm unless $P = NP$ [72]).

Placement typically splits into global and detailed placement. Global placement ends with an infeasible placement, but with overlaps that can be removed by local moves: there is no large region that contains too many objects. The main objective of global placement is to minimize the weighted net length. Detailed placement, or legalization, takes the global placement as input and legalizes it by making only local changes. Here the objective is to ensure the previously neglected constraints while minimizing the perturbation of the global placement.

The global placement algorithm developed in [92,96,12,13] has two major components: quadratic placement and multisection.

At each stage the chip area $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is partitioned by coordinates $x_{\min} = x_0 \le x_1 \le x_2 \le \ldots \le x_{n-1} \le x_n = x_{\max}$ and $y_{\min} = y_0 \le y_1 \le y_2 \le \ldots \le y_{m-1} \le y_m = y_{\max}$ into an array of regions $R_{ij} = [x_{i-1}, x_i] \times [y_{j-1}, y_j]$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Initially, $n = m = 1$. Each movable object is assigned to one region (cf. Figure 8).

In the course of global placement, columns and rows of this array, and thus the regions, are subdivided, and movable objects are assigned to subregions. After global placement, these rows correspond to cell rows with the height of standard cells, and the columns are small enough so that no region contains more than a few dozen movable objects. On a typical chip in 32 nm technology we have, depending on the library and die size, about 10 000 rows and 2 000 columns.

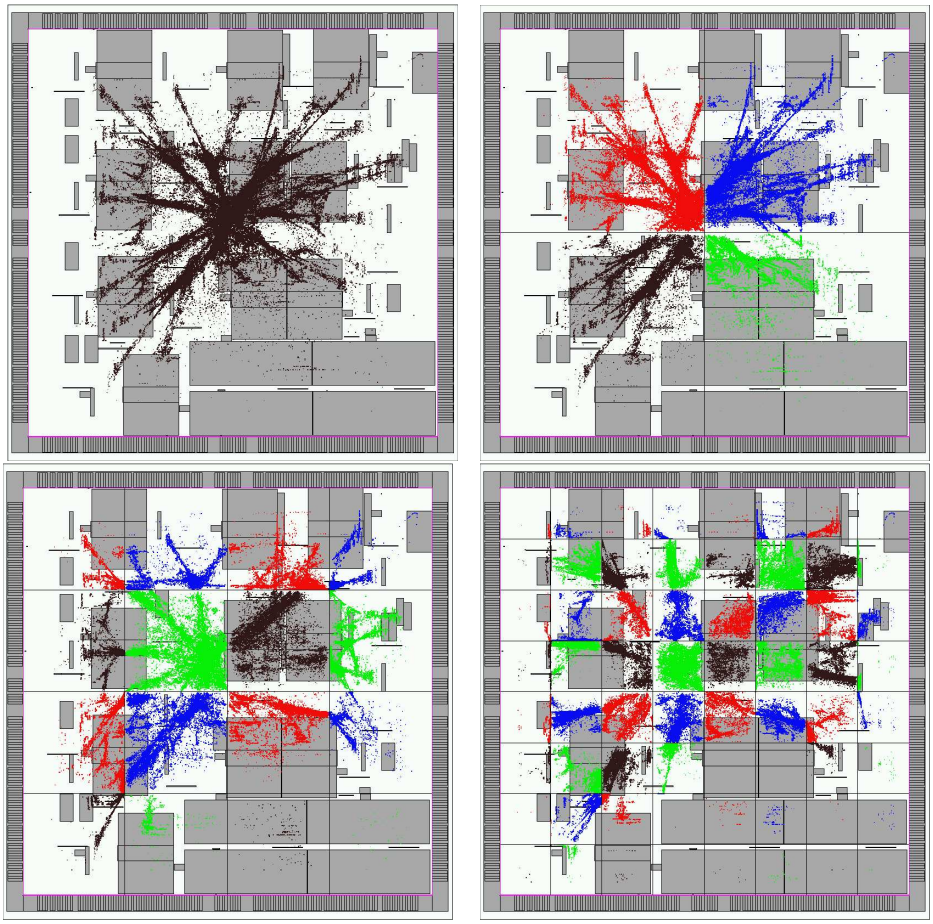

Figure 8. The initial four levels of the global placement with 1, 4, 16, and 64 regions. Colors indicate the assignment of the movable objects to the regions. The large grey objects are fixed and serve as blockages.

2.3. Quadratic placement

Quadratic placement means solving

$$\min \sum_{N \in \mathcal{N}} \frac{w(N)}{|N| - 1} \sum_{p,q \in N} (X_{p,q} + Y_{p,q}),$$

where $\mathcal{N}$ is the set of nets, each net N is a set of pins, $|N|$ is its cardinality (which we assume to be at least two), and $w(N)$ is the weight of the net, which can be any positive number. For two pins p and q of the same net, $X_{p,q}$ is the function

(i) $(x(c) + x(p) - x(d) - x(q))^2$ if p belongs to movable object c with offset $x(p)$, q belongs to movable object d with offset $x(q)$, and c and d are assigned to regions in the same column.

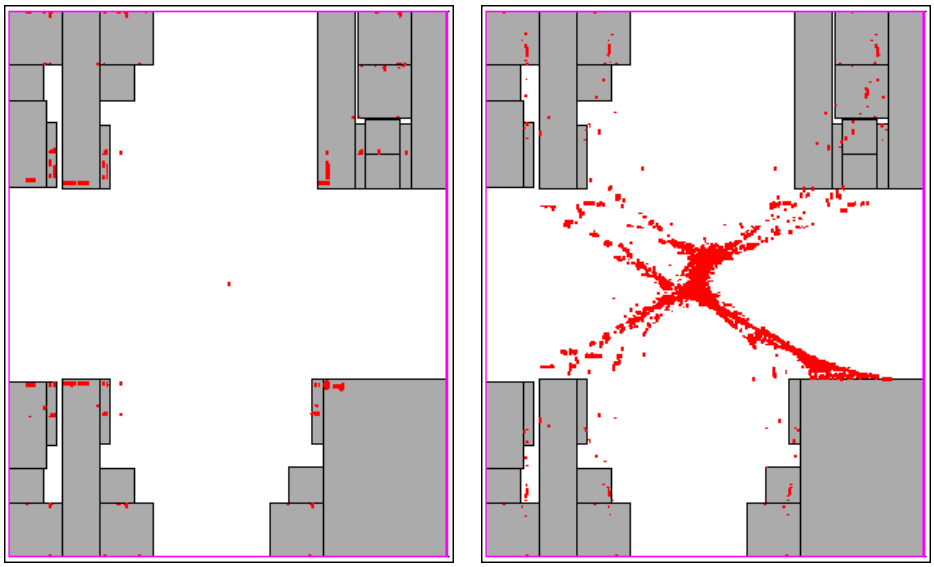

Figure 9. Minimizing the linear bounding box net length (left) gives hardly any information on relative positions compared to minimizing quadratic net length (right). As no disjointness constraints were considered yet, many cells share their position, especially on the left-hand side.

(ii) $(x(c) + x(p) - v)^2$ if p belongs to movable object c with offset $x(p)$, c is assigned to region $R_{i,j}$, q is fixed at a position with x-coordinate u, and $v = \max\{x_{i-1}, \min\{x_i, u\}\}$.

(iii) $(x(c) + x(p) - x_i)^2 + (x(d) + x(q) - x_{i'-1})^2$ if p belongs to movable object c with offset $x(p)$, q belongs to movable object d with offset $x(q)$, c is assigned to region $R_{i,j}$, d is assigned to region $R_{i',j'}$, and $i < i'$.

(iv) 0 if both p and q are fixed.

$Y_{p,q}$ is defined analogously, but with respect to y-coordinates, and with rows playing the role of columns.

In its simplest form, with $n = m = 1$, quadratic placement gives coordinates that optimize the weighted sum of squares of Euclidean distances of pin-to-pin connections (cf. the top left part of Figure 8). Replacing multi-terminal nets by cliques (i.e. considering a connection between p and q for all $p, q \in N$) is the best one can do as CLIQUE is the best topology-independent net model (see Section 2.1). Dividing the weight of a net by $|N| - 1$ is necessary to prevent large nets from dominating the objective function. Splitting nets along cut coordinates as in (ii) and (iii), first proposed in [92], partially linearizes the objective function and reflects the fact that long nets will be buffered later.

There are several reasons for optimizing this quadratic objective function. Firstly, delay along unbuffered wires grows quadratically with the length. Secondly, quadratic placement yields unique positions for most movable objects, allowing one to deduce much more information than the solution to a linear objective function would yield (see Figure 9). Thirdly, as shown in [98], quadratic placement is stable, i.e. almost invariant to small netlist changes. Finally, quadratic placement can be solved extremely fast.

To compute a quadratic placement, first observe that the two independent quadratic forms, with respect to x- and y-coordinates, can be solved independently in parallel.

Moreover, each row and column can be considered separately and in parallel. Each quadratic program is solved by the conjugate gradient method with incomplete Cholesky pre-conditioning. The running time depends on the number of variables, i.e. the number of movable objects, and the number of nonzero entries in the matrix, i.e. the number of pairs of movable objects that are connected. As large nets result in a quadratic number of connections, we replace large cliques, i.e. connections among large sets of pins in the same net that belong to movable objects assigned to regions in the same column (or row when considering y-coordinates), equivalently by stars, introducing a new variable for the center of a star. This was proposed in [92] and [12].

The running time to obtain sufficient accuracy grows slightly faster than linearly. There are linear-time multigrid solvers, but they do not seem to be faster in practice. We can compute a quadratic placement within at most a few minutes for 5 million movable objects. This is for the unpartitioned case $n = m = 1$; the problem becomes easier by partitioning, even when sequential running time is considered.

It is probably not possible to add linear inequality constraints to the quadratic program without a significant impact on the running time. However, linear equality constraints can be added easily, as was shown by [47]. Before partitioning, we analyze the quadratic program and add center-of-gravity constraints to those regions whose movable objects are not sufficiently spread. As the positions are the only information considered by partitioning, this is necessary to avoid random decisions. See also [18] for a survey on analytical placement.

2.4. Multisection

Quadratic placement usually has many overlaps which cannot be removed locally. Before legalization we have to ensure that no large region is overloaded. For this global placement has a second main ingredient, which we call multisection.

The basic idea is to partition a region and assign each movable object to a subregion. While capacity constraints have to be observed, the total movement should be minimized, i.e. the positions of the quadratic placement should be changed as little as possible. More precisely we have the following problem.

MULTISECTION PROBLEM

Instance:

- Finite sets C (cells) and R (regions),
- sizes $size : C \to \mathbb{R}_{\geq 0}$,
- capacities $cap : R \to \mathbb{R}_{\geq 0}$ and
- costs $d : C \times R \to \mathbb{R}$.

Task: Find an assignment $g : C \to R$ with
$\sum_{c \in C : g(c) = r} size(c) \leq cap(r)$ (for all $r \in R$)
minimizing the total cost $\sum_{c \in C} d(c, r)$.

This partitioning strategy has been proposed in [92] for $k = 4$ and ℓ_1-distances as costs as the QUADRISECTION PROBLEM, and was then generalized to arbitrary k and costs in [12]. It is a generalization of the ASSIGNMENT PROBLEM where the sizes and capacities are all 1. To decide whether a solution of the MULTISECTION PROBLEM exists is *NP*-complete (even for $|R| = 2$) since it contains the decision problem PARTITION. For

our purpose it suffices to solve the fractional relaxation which is known as the HITCHCOCK TRANSPORTATION PROBLEM. Here each $c \in C$ can be assigned fractionally to several regions:

HITCHCOCK TRANSPORTATION PROBLEM

Instance:
- Finite sets C (cells) and R (regions),
- sizes $size : C \to \mathbb{R}_{\geq 0}$,
- capacities $cap : R \to \mathbb{R}_{\geq 0}$ and
- costs $d : C \times R \to \mathbb{R}$.

Task: Find a fractional assignment $g : C \times R \to \mathbb{R}_+$ with
$\sum_{r \in R} g(c, r) = size(c)$ for all $c \in C$ and
$\sum_{c \in C} g(c, r) \leq cap(r)$ for all $r \in R$ minimizing
$\sum_{c \in C} \sum_{r \in R} g(c, r)d(c, r)$.

A nice characteristic of the fractional problem is, that one can easily find an optimum solution with only a few fractionally assigned cells. Most cells can be assigned to a unique region as shown by Vygen [96]:

Proposition 1. *From any optimum solution g to the* HITCHCOCK PROBLEM *we can obtain another optimum solution $g^\star$ in $O(|C||R|^2)$ time that is integral up to $|R| - 1$ cells.*

Proof: W.l.o.g. let $R = \{1, \ldots, k\}$. Let g be an optimum solution. Define $\Phi(g) := |\{(c, r) \mid c \in C,\, r \in R,\, g(c, r) > 0\}|$.

Let G be the undirected graph with vertex set R. For every cell c that is not assigned integrally to one region, add an edge between the region i with least i and the region j with largest j that contain parts of c. (G may have parallel edges.) If $|E(G)| \leq k - 1$, we are done.

Otherwise G contains a circuit $(\{v_1, \ldots, v_j, v_{j+1} = v_1\}, \{\{v_i, v_{i+1}\} \mid i = 1, \ldots, j\})$. For each $i \in \{1, \ldots, j\}$ there is a $c_i \in C$ with $0 < g(c_i, v_i) < size(c_i)$ and $0 < g(c_i, v_{i+1}) < size(c_i)$ (here $v_{j+1} := v_1$). $c_1, \ldots, c_j$ are pairwise distinct. Hence for a sufficiently small $\epsilon > 0$ we have that g' and g'' are feasible fractional partitions, where $g'(c_i, v_i) := g(c_i, v_i) - \epsilon$, $g'(c_i, v_{i+1}) := g(c_i, v_{i+1}) + \epsilon$, $g''(c_i, v_i) := g(c_i, v_i) + \epsilon$, $g''(c_i, v_{i+1}) := g(c_i, v_{i+1}) - \epsilon$ ($i = 1, \ldots, j$) and $g'(c, r) := g''(c, r) := g(c, r)$ for $c \in C \setminus \{c_1, \ldots, c_j\}$ and $r \in R$.

The arithmetic mean of the objective function values of g' and g'' is precisely that of g, implying that g' and g'' are also optimum. If we choose ϵ as large as possible, $\Phi(g')$ or $\Phi(g'')$ is strictly smaller than $\Phi(g)$.

After $|C||R|$ iterations Φ must be zero. Note that each iterations can be performed in $O(|R|)$ time, including the update of G. □

The Hitchcock transportation problem can be modeled as a minimum cost flow problem as Figure 10 indicates. The fastest standard minimum cost flow algorithm runs in $O(n \log n(n \log n + kn))$ [67]. However, super-quadratic running times are too slow for VLSI instances. For the quadrisection case, where $k = 4$ and d is the ℓ_1-distance, there is a linear-time algorithm by Vygen [96]. The algorithm is quite complicated but very efficient in practice. Recently, Brenner [9] proposed an $O(nk^2(\log n + k \log k))$-algorithm for the

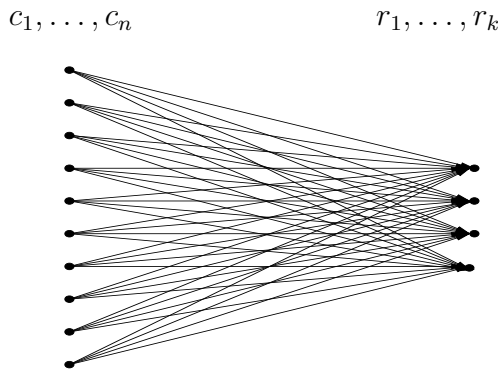

Figure 10. The Hitchcock transportation problem is a relaxation of the multisection problem. All arcs are oriented from left to right and are uncapacitated. The supply vertices on the left correspond to movable objects and have supply $size(c_1), \ldots, size(c_n)$. The demand vertices on the right correspond to subregions and have demand $cap(r_1), \ldots, cap(r_k)$. Arc costs are $d(c_i, r_j)$ for all $1 \leq i \leq n, 1 \leq j \leq k$. Note that $k \ll n$ in this application.

general case. This is extremely fast also in practice and has replaced the quadrisection algorithm of [96] in BonnPlace.

The idea is based on the well-known successive shortest paths algorithm (cf. [52]). Let the cells $C = \{c_1, c_2, \ldots, c_n\}$ be sorted by size $size(c_1) \geq size(c_2) \geq \cdots \geq size(c_n)$. We assign the objects in this order. A key observation is that for doing this optimally we need to re-assign only $O(k^2)$ previously assigned objects and thus can apply a minimum cost flow algorithm in a digraph whose size depends on k only. Note that k is less than 10 in all our applications, while n can be in the millions. The relevant results for our purposes are summarized in following theorem ([96],[9]).

Theorem 2. *The Hitchcock transportation problem with $|R| = 4$ and ℓ_1-distance can be solved in $O(n)$ time. The general case can be solved in $O(nk^2(\log n + k \log k))$ time, where $n = |C|$ and $k = |R|$.*

Figure 11 shows a multisection example where the movable objects are assigned optimally to nine regions.

2.5. Overall global placement and macro placement

With these two components, quadratic placement and multisection, the global placement can be described. Each level begins with a quadratic placement. Before subdividing the array of regions further, we fix macro cells that are too large to be assigned completely to a subregion. Macro placement uses minimum cost flow, branch-and-bound, and greedy techniques. We briefly describe its main component.

Assume that we want to place rectangular macros numbered $c_1, \ldots, c_n$, with widths $w_1, \ldots, w_n$ and heights $h_1, \ldots, h_n$ within an area $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$. The objective function is weighted bounding box netlength. If we fix a relation $r_{ij} \in \{W, S, E, N\}$ for each $1 \leq i < j \leq n$, then this can be written as linear program:

$$\min \sum_{N \in \mathcal{N}} w(N) \left(\overline{x}_N - \underline{x}_N + \overline{y}_N - \underline{y}_N \right) \tag{1}$$

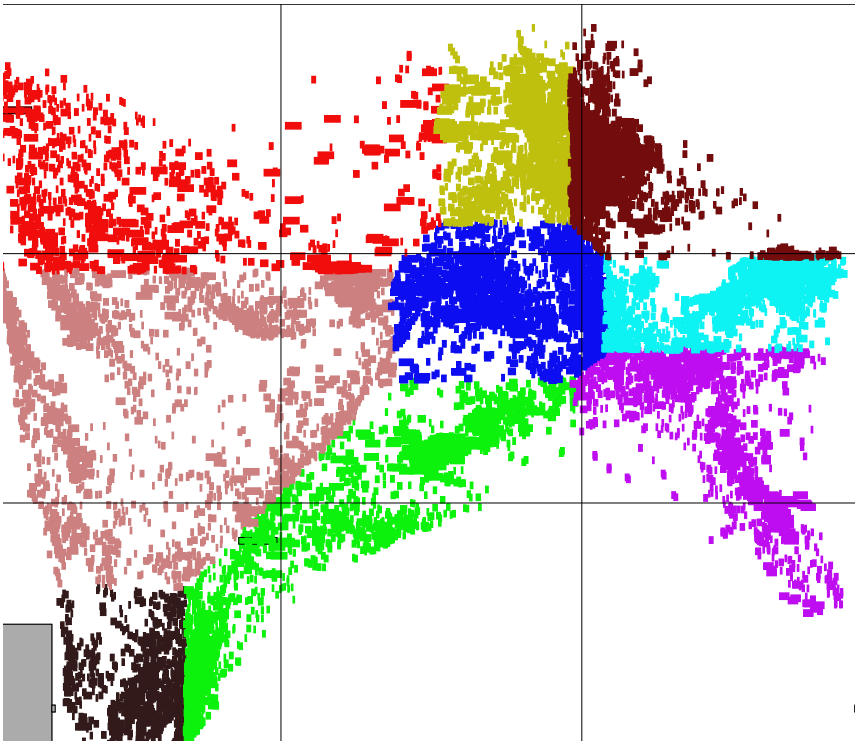

Figure 11. An example for multisection: objects are assigned to 3×3 subregions. The colors reflect the assignment: the red objects are assigned to the top left region, the yellow ones to the top middle region, and so on. This assignment is optimal with respect to total ℓ_1-distance.

subject to

$$
\begin{aligned}
x_i &\geq x_{\min} && \text{for } i = 1, \ldots, n \\
x_i + w_i &\leq x_{\max} && \text{for } i = 1, \ldots, n \\
y_i &\geq y_{\min} && \text{for } i = 1, \ldots, n \\
y_i + h_i &\leq y_{\max} && \text{for } i = 1, \ldots, n \\
x_i + w_i &\leq x_j && \text{for } 1 \leq i < j \leq n \text{ with } r_{ij} = W \\
x_j + w_j &\leq x_i && \text{for } 1 \leq i < j \leq n \text{ with } r_{ij} = E \\
y_i + h_i &\leq y_j && \text{for } 1 \leq i < j \leq n \text{ with } r_{ij} = S \\
y_j + h_j &\leq y_i && \text{for } 1 \leq i < j \leq n \text{ with } r_{ij} = N \\
x_i + x(p) &\geq \underline{x}_N && \text{for } i = 1, \ldots, n \text{ and } p \in P \text{ with } \gamma(p) = c_i \\
x(p) &\geq \underline{x}_N && \text{for } p \in P \text{ with } \gamma(p) = \square \\
x_i + x(p) &\leq \overline{x}_N && \text{for } i = 1, \ldots, n \text{ and } p \in P \text{ with } \gamma(p) = c_i \\
x(p) &\leq \overline{x}_N && \text{for } p \in P \text{ with } \gamma(p) = \square \\
y_i + y(p) &\geq \underline{y}_N && \text{for } i = 1, \ldots, n \text{ and } p \in P \text{ with } \gamma(p) = c_i \\
y(p) &\geq \underline{y}_N && \text{for } p \in P \text{ with } \gamma(p) = \square \\
y_i + y(p) &\leq \overline{y}_N && \text{for } i = 1, \ldots, n \text{ and } p \in P \text{ with } \gamma(p) = c_i \\
y(p) &\leq \overline{y}_N && \text{for } p \in P \text{ with } \gamma(p) = \square
\end{aligned}
\tag{2}
$$

This is the dual of an uncapacitated minimum cost flow problem. Hence we can find an optimum solution to (2) in $O((n+m)(p+n^2+m\log m)\log(n+m))$ time, where $n=|C|$, $m=|\mathcal{N}|$, and $p=|P|$. Instead of enumerating all $2^{n(n-1)}$ possibilities for r, it suffices to enumerate all pairs of permutations π, ρ on $\{1,\ldots,n\}$. For $1 \le i < j \le n$ we then define $r_{ij} := W$ if i precedes j in π and ρ, $r_{ij} := E$ if j precedes i in π and ρ, $r_{ij} := S$ if i precedes j in π and j precedes i in ρ, and $r_{ij} := N$ if j precedes i in π and i precedes j in ρ. One of the $(n!)^2$ choices will lead to an optimum placement. This sequence-pair representation is due to [43] and [66]. In practice, however, a branch-and-bound approach is faster; Hougardy [39] solves instances up to approximately 20 circuits optimally. This is also used as part of a post-optimization heuristic.

However, interaction of small and large blocks in placement is still not fully understood [17,86], and placing large macros in practice often requires a significant amount of manual interaction.

After partitioning the array of regions, the movable objects are assigned to the resulting subregions. Several strategies are applied (see [13] and [89] for details), but the core subroutine in each case is the multisection described above. An important further step is repartitioning, where 2×2 or even 3×3 sub-arrays of regions are considered and all their movable objects are reassigned to these regions, essentially by computing a local quadratic placement followed by multisection.

There are further components which reduce routing congestion [11], deal with timing and resistance constraints, and handle other constraints like user-defined bounds on coordinates or distances of some objects. Global placement ends when the rows correspond to standard cell heights. Typically there are fewer columns than rows as most movable objects are wider than high. Therefore 2×3 partitioning is often used in the late stages of global placement.

2.6. Detailed placement

Detailed placement, or legalization, considers standard cells only; all others are fixed beforehand. The task is to place the standard cells legally without changing the (illegal) input placement too much. Detailed placement does not only arise as a finishing step during the overall placement flow, but also in interaction with timing optimization and clock tree insertion. These steps add, remove or resize cells and thus require another legalization of the placement that has become illegal.

Due to technology constraints cells cannot be placed with arbitrary y-coordinates. Instead, they have to be arranged in cell rows. Cells that are higher than a cell row must be fixed before detailed placement. Thus we can assume unit height and have the following problem formulation:

PLACEMENT LEGALIZATION PROBLEM

Instance:

- A rectangular chip area $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$,
- a set of rectangular blockages,
- a set C of rectangular cells with unit height,
- An equidistant subdivision $y_0^R = y_{\min} \leq y_1^R \leq \cdots \leq y_{n_R}^R = y_{\max}$ of $[y_{\min}, y_{\max}]$ into *cell rows*
- a width $w(c)$ and a position $(x(c), y(c)) \in \mathbb{R}^2$ of each cell $c \in C$.

Task: Find new positions $(x'(c), y'(c)) \in \mathbb{Z}^2$ of the cells such that

- each cell is contained in the chip area,
- each cell snaps into a cell row (its y-coordinate is a cell row coordinate),
- no two cells overlap,
- no cell overlaps with any blockage,

and $\sum_{c \in C} ((x(c) - x'(c))^2 + (y(c) - y'(c))^2)$ is minimum.

It is quite natural to model the legalization problem as a minimum cost flow problem, where flow goes from supply regions with too many objects to demand regions with extra space [93]. Brenner and Vygen [16] refined this approach. We describe this enhanced legalization algorithm in the following.

It consists of three phases. A *zone* is defined as a maximal part of a cell row that is not blocked by any fixed objects, i.e. can be used for legalization.

The first phase guarantees that no zone contains more cells than fit into it. The second phase places the cells legally within each zone in the given order. When minimizing quadratic movement, this can be done optimally in linear time by Algorithm 1, as shown in [16] (see also [44], [14], and [90]). The algorithm gets as inputs a zone $[x_{\min}, x_{\max}]$, coordinates $x_1, \ldots, x_n \in \mathbb{R}$ and widths $w_1, \ldots, w_n > 0$ with $\sum_{i=1}^n w_i \leq x_{\max} - x_{\min}$, and legalizes the circuits within $[x_{\min}, x_{\max}]$ in the given order. As all circuits stay within one zone, we do not consider y-coordinates. It places the circuits from left to right, each optimally and as far to the left as possible. If a circuit cannot be placed optimally, it is merged with its predecessor.

Theorem 3. *Algorithm 1 runs in linear time and computes coordinates $x'_1, \ldots, x'_n$ with $x_{\min} \leq x'_1$, $x'_i + w_i \leq x'_{i+1}$ $(i = 1, \ldots, n-1)$, and $x_n + w_n \leq x_{\max}$, such that $\sum_{i=1}^n (x_i - x'_i)^2$ is minimum.*

Proof: Each iteration increases i by one or decreases $|\mathcal{L}|$ and i by one. As $1 \leq i \leq |\mathcal{L}| \leq n + 1$, the total number of iterations is at most $2n$. Each takes constant time.

To prove correctness, the main observation is that we can merge circuits without losing optimality. So if we merge circuits h and i, we write $\mathcal{L}' := \{j \in \mathcal{L} \mid j < i\}$ and claim that there exists an optimum solution $(x_j^*)_{j \in \mathcal{L}' \cup \{i\}}$ of the subproblem defined by $(f_j, W_j)_{j \in \mathcal{L}' \cup \{i\}}$ where $x_h^* + W_h = x_i^*$.

Let $(x_j^*)_{j \in \mathcal{L}' \cup \{i\}}$ be an optimum solution of this subproblem. If $x_i^* - W_h \leq \arg\min f_h$, then x_h^* can be set to $x_i^* - W_h$ without increasing $f_h(x_h^*)$.

So suppose that $x_i^* > x_h^* + W_h$ and $x_i^* > \arg\min f_h + W_h$. Then $x_i^* > \max\{x_{\min}, \arg\min f_h\} + W_h \geq x'_h + W_h > \min\{x_{\max} - W_i, \arg\min f_i\}$, a contradiction as decreasing x_i^* would reduce $f_i(x_i^*)$. □

Let $f_i : x \mapsto (x - x_i)^2$.
$x'_0 \leftarrow x_{\min}$, $W_0 \leftarrow 0$, $W_i \leftarrow w_i$ for $i = 1, \ldots, n$.
Let $\mathcal{L}$ be the list consisting of $0, 1, \ldots, n, n+1$.
$i \leftarrow 1$.
while $i < n + 1$ **do**
    Let h be the predecessor and j the successor of i in $\mathcal{L}$.
    if $h = 0$ or $x'_h + W_h \leq \min\{x_{\max} - W_i, \arg\min f_i\}$ **then**
        $x'_i \leftarrow \max\{x_{\min}, \min\{x_{\max} - W_i, \arg\min f_i\}\}$.
        $i \leftarrow j$.
    else
        Redefine f_h by $f_h : x \mapsto f_h(x) + f_i(x + W_h)$.
        $W_h \leftarrow W_h + W_i$.
        Remove i from $\mathcal{L}$.
        $i \leftarrow h$.
    end if
end while
for $i \in \{1, \ldots, n\} \setminus \mathcal{L}$ **do**
    $x'_i \leftarrow x'_h + \sum_{j=h}^{i-1} w_j$, where h is the maximum index in $\mathcal{L}$ that is smaller than i.
end for

Algorithm 1: Single Row Placement Algorithm

Finally, some post-optimization heuristics (like exchanging two cells, but also much more complicated operations) are applied.

The most difficult and important phase is the first one, which we describe here in detail. If the global placement is very dense in some areas, a significant number of cells has to be moved. As phase two works in each zone separately, phase one has to guarantee that no zone contains more objects than fit into it.

In order to prevent long-distance movements within the zones later in phase two, wide zones are partitioned into regions. Each movable object is assigned to a region. This means that the center of the movable object must be located in the assigned region. Parts of an object can overlap with neighboring regions.

Unless all movable objects that are assigned to a region R can be placed legally with their center in R, some of them have to be moved out of R. But this is not sufficient: in addition, it may be necessary to move some objects out of certain sequences of consecutive regions. More precisely, for a sequence of consecutive regions $R_1, \ldots, R_k$ within a zone, we define its supply by

$$supp(R_1, \ldots, R_k) := \max \left\{ 0, \sum_{i=1}^{k} (w(R_i) - a(R_i)) - \frac{1}{2}(w_l(R_1) + w_r(R_k)) - \sum_{1 \leq i < j \leq k, (i,j) \neq (1,k)} supp(R_i, \ldots, R_j) \right\},$$

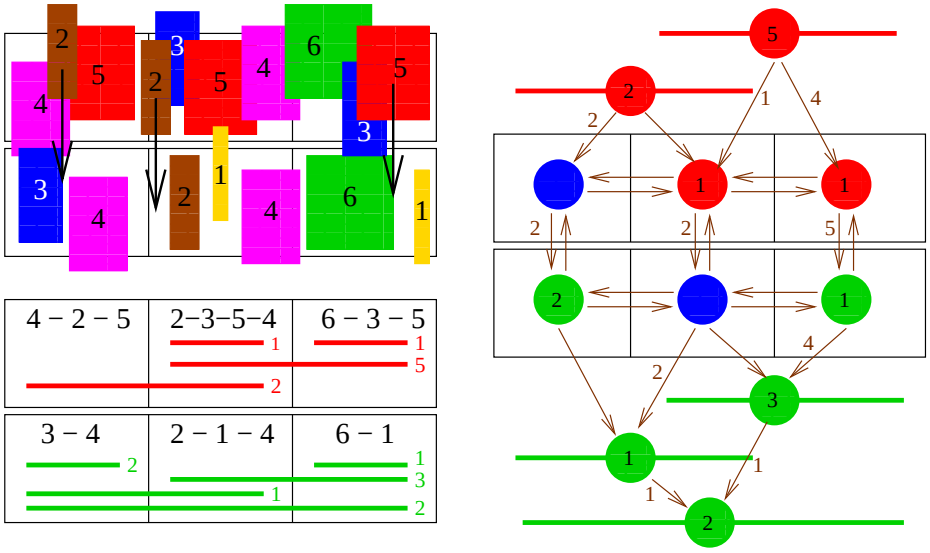

Figure 12. An example with two zones and six regions, each of width 10 (top left), the supply (red) and demand (green) regions and intervals with their supply and demand (bottom left), and the minimum cost flow instance (right) with a solution shown in brown numbers. To realize this flow, objects of size 2, 2, and 5, respectively, have to be moved from the top regions downwards.

where $a(R_i)$ is the width of region R_i, $w(R_i)$ is the total width of cells that are currently assigned to region R_i, and $w_l(R_i)$ and $w_r(R_i)$ are the widths of the leftmost and rightmost cell in R_i, respectively, or zero if R_i is the leftmost (rightmost) region within the zone.

If $supp(R_1, \ldots, R_k)$ is positive, $(R_1, \ldots, R_k)$ is called a supply interval. Similarly, we define the demand of each sequence of consecutive regions, and the demand intervals. We now define a directed network $G = (V, E, c)$ on regions, supply intervals, and demand intervals, in which we compute a minimum cost flow that cancels demands and partly cancels supplies. Let vertices and edges be defined by

$$\begin{aligned} V(G) &:= \{\text{regions, supply intervals, demand intervals}\} \\ E(G) &:= \{(A, A') \mid A, A' \text{ adjacent regions}\} \\ &\quad \cup \{(A, A') \mid A \text{ supply interval, } A' \text{ maximal proper subset of } A\} \\ &\quad \cup \{(A, A') \mid A' \text{ demand interval, } A \text{ maximal proper subset of } A'\}. \end{aligned}$$

Let the cost $c(A, A')$ between two adjacent regions A, A' be the expected cost of moving a cell of width 1 from A to A' and all other arcs costs be zero. The construction of this uncapacitated minimum cost flow instance is illustrated in Figure 12. We look for a minimum cost flow f which cancels all supplies:

$$f(\delta^+(v)) - f(\delta^-(v)) \geq \text{supp}(v) + \text{dem}(v) \quad \text{for all } v \in V(G).$$

This can be computed in $O(n^2 \log^2 n)$ time by Orlin's minimum cost flow algorithm [67]. Figure 13 shows part of a typical result on a real chip.

Finally the flow is realized by moving objects along flow arcs. This means to move cells of total size $f(A, A')$ from region A to region A' for each pair of neighbors (A, A'). An exact realization may not exists as small amounts of flow may not be realizable by wide objects. Therefore the realization is approximated. We scan the arcs carrying flow in topological order and solve a multi-knapsack problem by dynamic programming for selecting the best set of cells to be moved for realizing the flow on each arc [93,16].

Of course zones can remain overloaded after realization. In this case phase one is repeated with increased region widths and decreased demand values. Typically, after a few iterations of phase 1 no overloaded zones remain.

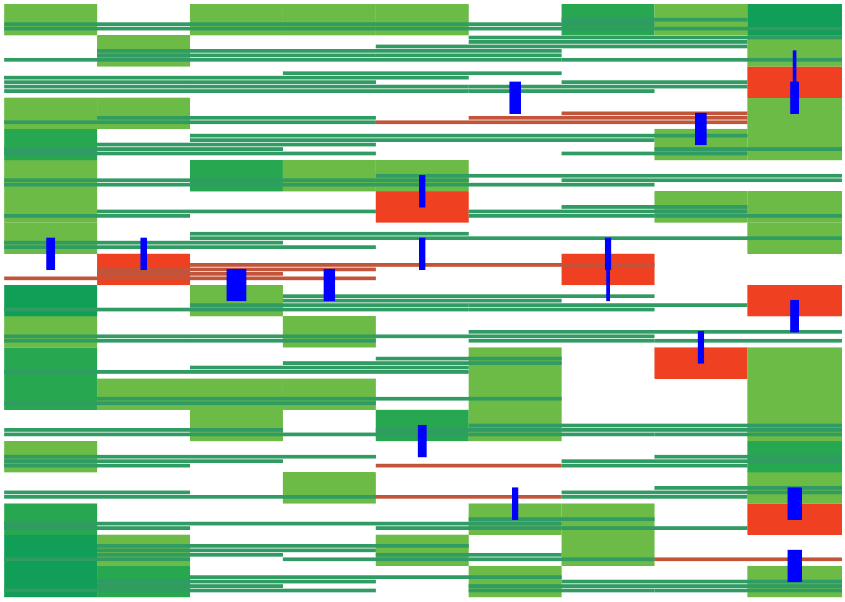

Figure 13. Small part of a real chip in legalization. Supply regions and intervals are shown in red, demand regions and intervals in green. The blue edges represent the minimum cost flow, and their width is proportional to the amount of flow.

The minimum cost flow formulation yields an optimum solution under some assumptions [16], and an excellent one in practice. Experimental results show that the gap between the computed solution and a theoretical lower bound is only approximately 10%, and neither routability nor timing is significantly affected [10].

3. Timing optimization

In this section we describe the main ingredients of timing optimization. These include algorithms for the construction of timing- and routing-aware fan-out trees (repeater trees), for the timing-oriented logic restructuring and optimization, and for the timing- and power-aware choice of different physical realizations of individual gates. Each is based on new mathematical theory.

Altogether, these routines combined with appropriate net weight generation and iterative placement runs form the so-called timing-driven placement. Using the new algorithms introduced in this section the overall turn-around time for timing closure, including full placement and timing optimization, could be reduced from more than a week to 26 hours on the largest designs.

3.1. Timing constraints

During optimization the signal propagation through a VLSI chip is estimated by a static timing analysis. We give a simplified description and refer the interested reader to [83] for further reading.

At every pin $v \in P$ the latest arrival time a_v of a possible signal occurrence is computed. Signals are propagated along the edges of the timing graph $G_{\mathcal{T}}$, which is a directed graph on the vertex set $V(G_{\mathcal{T}}) = P$ of pins in the design. $G_{\mathcal{T}}$ contains two type of edges. First "net" edges are inserted for each source-sink pin pair $(v, w) \in N$ of a net $N \in \mathcal{N}$, directed from the unique source of N to each sink. Second "circuit" edges $(v, w) \in E(G_{\mathcal{T}})$ are inserted for each input-output pin pair (v, w) of a cell, where a signal in v triggers a signal in w. For every edge $e \in E(G_{\mathcal{T}})$ a delay d_e is given. The delay depends on various parameters, e.g. circuit size, load capacitance (wire plus sink pin capacitances), net topologies, signal shapes.

The timing graph is acyclic as every cycle in the netlist is cut by a register, which does not have a direct input to output connection[2]. Arrival times are propagated in topological order. At each vertex $v \in V(G_{\mathcal{T}})$ with $\delta^-(v) = \emptyset$ a start time $AT(v)$ is given as design constraint. It initializes the arrival time in v by $a_v = AT(v)$. Then for each $v \in V(G_{\mathcal{T}})$ with $\delta^-(v) \neq \emptyset$, the arrival time a_v is the maximum over all incoming arrival times:

$$a_v := \max_{(u,v)\in\delta^-(v)} a_u + d_{(u,v)}. \tag{3}$$

At each endpoint pin $v \in V(G_{\mathcal{T}})$ with $\delta^+(v) = \emptyset$ of the combinational paths required arrival times $RAT(v)$ are given as design constraints. The signals arrive in time at v if $a_v \leq RAT(v)$. To measure timing feasibility on arbitrary pins, the maximum feasible required arrival time variable r_v is computed for each $v \in V(G_{\mathcal{T}})$. It is initialized by $r_v = RAT(v)$ for all $v \in V(G_{\mathcal{T}}), \delta^+(v) = \emptyset$. For the remaining vertices $v \in V(G_{\mathcal{T}})$ with $\delta^+(v) \neq \emptyset$ they are computed in reverse topological order by

$$r_v := \min_{(v,w)\in\delta^+(v)} a_w - d_{(v,w)}. \tag{4}$$

The difference $\sigma_v := r_v - a_v$ is called *slack*. If it is non-negative the timing constraints of all paths through v are satisfied. If $\sigma_v \geq 0$ for all endpoint pins $v \in V(G_{\mathcal{T}})$ with $\delta^+(v) = \emptyset$, all timing constraints are met, which implies $\sigma_v \geq 0$ for all nodes $v \in V(G_{\mathcal{T}})$. The slack can also be defined for an edge $(v, w) \in E(G_{\mathcal{T}})$ by $\sigma_{(v,w)} := r_w - d_{(v,w)} - a_v$ with following interpretation. When adding at most $\sigma_{(v,w)}$ to the delay $d_{(v,w)}$, all paths through (v, w) are fast enough. Note that $\sigma_{(v,w)}$ can also be negative; then delays must be reduced.

[2]We omit transparent latches here.

In some applications (e.g. Section 3.3) we are interested in a most critical path. Observe that there must be at least one path in $G_{\mathcal{T}}$ from a start pin $v \in V(G_{\mathcal{T}}), \delta^-(v) = \emptyset$ to an endpoint pin $v' \in V(G_{\mathcal{T}}), \delta^+(v') = \emptyset$, in which each vertex and edge has the overall worst slack $\min\{\sigma_v | v \in V(G_{\mathcal{T}})\}$. Such a path can be determined efficiently by backward search along a most critical incoming edge, starting from an endpoint pin $v' \in V(G_{\mathcal{T}}), \delta^+(v') = \emptyset$ with a smallest slack value $\sigma_{v'} = \min\{\sigma_v | v \in V(G_{\mathcal{T}})\}$.

Beside the introduced late mode constraints earliest arrival time or early mode constraints are given, too. Here signals must not arrive too early at the endpoints. Propagation is analog to (3) and (4) with $\min$ and $\max$ being swapped. Together with the arrival times also signal shapes — slews[3] in VLSI terminology — are propagated. These are needed for proper delay calculation. When incorporating slews the above propagation rules become incorrect, as an early signal with a very large slew can result in later arrival times in subsequent stages than a late signal with a tight slew. In [97] we describe how to overcome these slew related problems by a slight modification of the propagation rules.

3.2. Fan-out trees

On an abstract level the task of a fan-out tree is to carry a signal from one gate, the root r of the fan-out tree, to other gates, the sinks $s_1, \ldots, s_n$ of the fan-out tree, as specified by the netlist. If the involved gates are not too numerous and not too far apart, then this task can be fulfilled just by a metal connection of the involved pins, i.e. by a single net without any repeaters. But in general we need to insert repeaters (buffers or inverters). Inverters logically invert a signal while buffers implement the identity function. A *repeater tree* is a netlist in which all circuits are repeaters, r is the only input, and S is the set of outputs.

In fact, fan-out trees are a very good example for the observation mentioned in the introduction that the development of technology continually creates new complex design challenges that also require new mathematics for their solution. Whereas circuit delay traditionally dominated the interconnect delay and the construction of fan-out trees was of secondary importance for timing, the feature size shrinking is about to change this picture drastically.

Extending the current trends one can predict that in future technologies more than half of all circuits of a design will be needed just for bridging distances, i.e. in fan-out trees. The reason for this is that with decreasing feature sizes the wire resistances increase more than wire capacitances decrease. The delay over a pure metal wire is roughly proportional to the product of the total resistance and capacitance. It increases quadratically with its length, as both resistance and capacitance depend linearly on the length. But by inserting repeaters the growth rate can be kept linear. In current technologies buffers are realized by two subsequent inverters. Therefore, inverters are more flexible and typically faster than buffers.

Repeaters can be chosen from a finite library $\mathcal{L}$ of different sizes. Smaller sizes have a smaller drive strength. The smaller a repeater is, the higher is its delay sensitivity on the load capacitance $\frac{\partial delay}{\partial cap}$. On the other hand do smaller repeaters involve smaller input pin capacitances and therefore smaller load capacitances and delays for the predecessor. We now define the repeater tree problem.

[3] The slew of a signal is an estimate for how fast the voltage changes

REPEATER TREE PROBLEM

Instance:

- A root r and a set S of sinks.
- a start time $AT(r)$ at the root r and a required arrival time $RAT(s)$ at each sink s,
- a parity in $\{+,-\}$ for each sink indicating whether it requires the signal or its inversion,
- placement information for the root and the sinks $Pl(r), Pl(s_1), Pl(s_2), ..., Pl(s_n) \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$,
- physical information about the driver strength of r and the input capacitances of the sinks $s \in S$, and
- physical information about the wiring and the library $\mathcal{L}$ of available repeaters (inverters and buffers).

Task: Find a repeater tree T that connects r with all sinks in S such that

- the desired parities are realized (i.e. for each sink s the number of inverters on the r-s-path in T is even iff s has parity $+$),
- the delay from r to s is at most $RAT(s) - AT(r)$, for each $s \in S$,
- and the power consumption is minimum.

In another formulation, $AT(r)$ is not given but should be maximized. The procedure that we proposed for fan-out tree construction [5,6,7] works in two phases. The first phase generates a preliminary topology for the fan-out tree, which connects very critical sinks in such a way as to maximize the minimum slack, and which minimizes wiring for non-critical sinks. During the second phase the resulting topology is finalized and buffered in a bottom-up fashion using mainly inverters and respecting the parities of the sinks.

A *topology* for root r and set S of sinks is a pair (T, Pl) where T is an arborescence rooted at r in which the root has one child, the sinks have no children, and all other vertices have two children, and $Pl : V(T) \setminus (\{r\} \cup S) \rightarrow [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is an embedding of the internal vertices of T in the chip area. Let us denote by $T_{[r,s]}$ the unique path from r to s in T. A simplified delay model is used to model delay within (T, Pl). The delay on the path from the root r to a sink $s \in S$ is approximated by

$$c_{node} \cdot (|E(T_{[r,s]})| - 1) \; + \sum_{(u,v) \in E(T_{[r,s]})} c_{wire} \cdot ||Pl(u) - Pl(v)||_1 \qquad (5)$$

The second term in this formula accounts for the wire or distance delay of the r-s-path in T. This is a linear function in wire length as buffering from phase two is anticipated. The first term adds an additional delay of c_{node} for every bifurcation. This reflects the fact that a bifurcation adds additional capacitance. A constant adder is used as repeaters can be inserted later to shield large downstream capacitances in the branches.

The involved constants are derived in a pre-processing step. The accuracy of this very simple delay model is illustrated in Figure 14, which compares the estimated delay with the measured delay after buffering and sizing at the critical sinks.

Our topology generation algorithm inserts the sinks by a greedy strategy. First the sinks are sorted by their criticality. As criticality we use an upper bound for the slack at

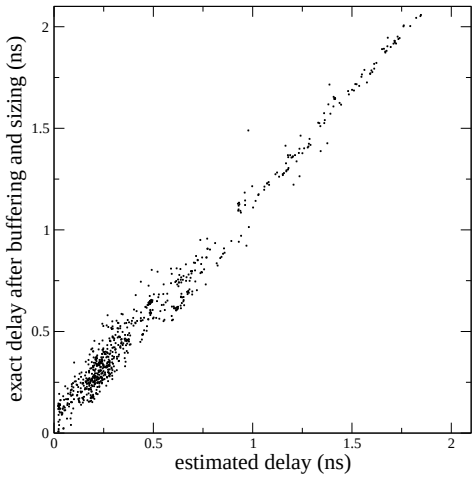

Figure 14. The simple timing model used for topology generation matches actual timing results after buffering well.

the sink, namely the slack that would result at $s \in S$ if we connected s to r by a shortest possible wire without any bifurcation:

$$\sigma_s := RAT(s) - AT(r) - c_{wire}||Pl(r) - Pl(s)||_1. \tag{6}$$

The individual sinks are now inserted one by one into the preliminary topology in order of non-increasing criticality, i.e. non-decreasing value of σ_s. When we insert a new sink s, we consider all arcs $e = (u, v) \in E(T)$ of the preliminary topology constructed so far and estimate the effect of subdividing e by a new internal node w and connecting s to w.

The sink s will be appended to a new vertex subdividing an arc e of T that maximizes $\xi\sigma_e - (100 - \xi)l_e$, where σ_e and l_e estimate the corresponding worst slack at r and the total length, respectively, when choosing e. The parameter $\xi \in [0, 100]$ allows us to favor slack maximization for timing critical instances or wiring minimization for non-critical instances. Figure 15 gives an example for a preliminary topology.

In most cases it is reasonable to choose values for ξ that are neither too small nor too large. Nevertheless, in order to mathematically validate our procedure we have proved optimality statements for the extreme values $\xi = 0$ and $\xi = 100$. If we ignore timing ($\xi = 0$) and choose an appropriate order of the sinks, the final length of the topology is at most $3/2$ times the minimum length of a rectilinear Steiner tree connecting the root and the sinks. If we ignore wiring ($\xi = 100$), the topology realizes the optimum slack with respect to our delay model (see below).

To measure the quality of the resulting topologies in practice, we can compute bounds for performance and power consumption to compare against. A minimum power topology obviously arises from a minimum Steiner tree on $S \cup \{r\}$. Some Steiner points may need to be replaced by two topology nodes with same coordinates. The following bound can be specified for the maximum achievable slack:

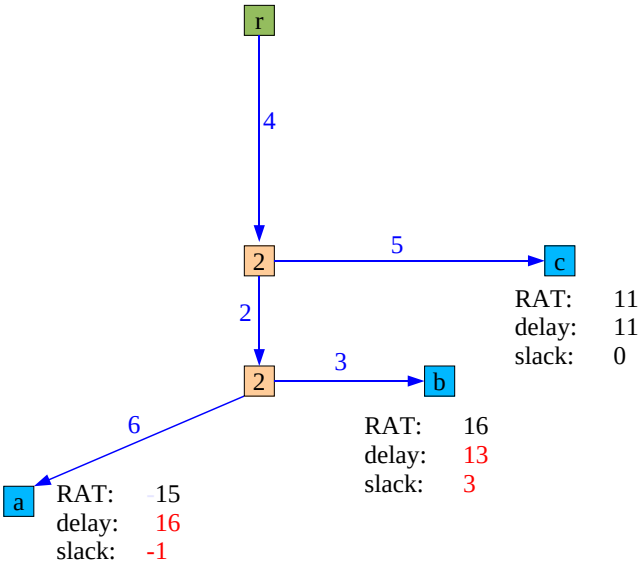

Figure 15. An example for topology generation with $AT(r) = 0$, $c_{wire} = 1$, $c_{node} = 2$, and three sinks a, b and c with displayed required arrival times. The criticalities are $\sigma_a = 15 - 0 - (4 + 2 + 6) = 3$, $\sigma_b = 16 - 0 - (4 + 2 + 3) = 7$, and $\sigma_c = 11 - 0 - (4 + 5) = 2$. Our algorithm first connects the most critical sink c to r. The next critical sink is a which is inserted into the only arc (r, c) creating an internal node w. For the insertion of the last sink b there are now three possible arcs (r, w), (w, a), and (w, c). Inserting b into (w, a) creates the displayed topology whose worst slack is -1, which is best possible here.

Theorem 4. *The maximum possible slack $\sigma_{\max}$ of a topology (T, Pl) with respect to our delay model is at most*

$$-c_{node} \cdot \log_2 \left(\sum_{s \in S} 2^{-\left(\frac{RAT(s) - AT(r) - c_{wire}||Pl(r) - Pl(s)||_1}{c_{node}} \right)} \right).$$

Proof: If $|S| = 1$, the statement is trivial. Let us assume $|S| > 1$. This means that we have at least one internal node in T. We can assume that all internal nodes are placed at $Pl(r)$. The slack of a such a topology T is at least $\sigma_{\max}$ if and only if

$$RAT(s) - AT(r) - c_{wire}||Pl(r) - Pl(s)||_1 - c_{node} \cdot (|E(T_{[r,s]})| - 1) \geq \sigma_{\max},$$

for all sinks s. Equivalently,

$$|E(T_{[r,s]})| - 1 \leq \frac{RAT(s) - AT(r) - c_{wire}||Pl(r) - Pl(s)||_1}{c_{node}} - \frac{\sigma_{\max}}{c_{node}}.$$

By Kraft's inequality [54] there exists a rooted binary tree with n leaves at depths $l_1, l_2, \ldots, l_n$ if and only if $\sum_{i=1}^{n} 2^{-l_i} \leq 1$. If we contract the arc incident to r in our topology we obtain a binary tree for which $(|E(T_{[r,s]})| - 1)$ is exactly the depth of sink s (remember |S| > 1). Now Kraft's inequality implies the theorem. □

It is possible to calculate a slightly better and numerically stable bound using Huffman coding [40]: if we set as in (6)

$$\sigma_s = RAT(s) - AT(r) - c_{wire}||Pl(r) - Pl(s)||_1$$

for all $s \in S$, order these values $\sigma_{s_1} \leq \sigma_{s_2} \leq \cdots \leq \sigma_{s_n}$, and iteratively replace the largest two $\sigma_{s_{n-1}}$ and σ_{s_n} by $-c_{node} + \min\{\sigma_{s_{n-1}}, \sigma_{s_n}\} = -c_{node} + \sigma_{s_{n-1}}$ until only one value σ^* is left, then the maximum possible slack with respect to our delay model is at most this σ^*. This bound is never worse than the closed formula of Theorem 4. In fact, it corresponds to shortest wires from each sink to the source and an optimum topology all internal nodes of which are at the position of the root. Such a topology would of cause waste too many wiring resources and lead to excessive power consumption. The topology generated by our algorithm is much better in these respects. Moreover we have the following guarantee.

Theorem 5. *For $c_{wire} = 0$, the topology constructed by the above procedure with $\xi = 100$ realizes the maximum possible slack with respect to our delay model.*

For $c_{node} = 1$ and integer values for $AT(r)$ and $RAT(s)$, $s \in S$, the theorem follows quite easily from Kraft's inequality, by induction on $|S|$ [5]. The general case is more complicated; see [7].

After inserting all sinks into the preliminary topology, the second phase begins, in which we insert the actual inverters [6]. For each sink s we create a cluster C containing only s. In general a cluster C is assigned a position $Pl(C)$, a set of sinks $S(C)$ all of the same parity, and an estimate $W(C)$ for the wiring capacitance of a net connecting a circuit at position $Pl(C)$ with the sinks in $S(C)$. The elements of $S(C)$ are either original sinks of the fan-out tree or inverters that have already been inserted.

There are three basic operations on clusters. Firstly, if $W(C)$ and the total input capacitance of the elements of $S(C)$ reach certain thresholds, we insert an inverter I at position $Pl(C)$ and connect it by wire to all elements of $S(C)$. We create a new cluster C' at position $Pl(C)$ with $S(C') = \{I\}$ and $W(C) = 0$. As long as the capacitance thresholds are not attained, we can move the cluster along arcs of the preliminary topology towards the root r. By this operation $W(C)$ increases while $S(C)$ remains unchanged. Finally, if two clusters happen to lie on a common position and their sinks are of the same parity, we can merge them, but we may also decide to add inverters for some of the involved sinks. This decision again depends on the capacitance thresholds and on the objectives timing and wire length.

During buffering, the root connects to the clusters via the preliminary topology and the clusters connect to the original sinks s_i via appropriately buffered nets. Once all clusters have been merged to one which arrives at the root r, the construction of the fan-out tree is completed.

The optimality statements which we proved within our delay model and the final experimental results show that the second phase nearly optimally buffers the desired connections. Our procedure is extremely fast. The topology generation solved 4.6 million instances with up to 10000 sinks from a current 90 nm design in less than 100 seconds on a 2.93 GHz Xeon machine [6], and the buffering is completed in less than 45 minutes. On average we deviated less than 1.5 % from the minimum length of a rectilinear Steiner tree

Figure 16. A sequence of circuits with Boolean functions $g_1, g_2, \ldots, g_n$ on a critical path P.

when minimizing wire length, and less than 3 ps from the theoretical upper slack bound when maximizing worst slack.

We are currently including enhanced buffering with respect to timing constraints, wire sizing, and plane assignment in our algorithm. We are also considering an improved topology generation, in particular when placement or routing resources are limited.

3.3. Fan-in trees

Whereas in the last section one signal had to be propagated to many destinations via a logically trivial structure, we now look at algorithmic tasks posed by the opposite situation in which several signals need to be combined to one signal as specified by some Boolean expression. The netlist itself implicitly defines such a Boolean expression for all relevant signals on a design. The decisions about these representations were taken at a very early stage in the design process, i.e. in logic synthesis, in which physical effects could only be crudely estimated. At a relatively late stage of the physical layout process much more accurate estimates are available. If most aspects of the layout have already been optimized but we still see negative slack at some cells, changing the logic that feeds the cell producing the late signal is among the last possibilities for eliminating the timing problem. Traditionally, late changes in the logic are a delicate matter and only very local modifications replacing some few cells have been considered, also due to the lack of global algorithms.

To overcome the limitations of purely local and conservative changes, we have developed a totally novel approach that allows for the redesign of the logic on an entire critical path taking all timing and placement information into account [79]. Keep in mind that static timing analysis computes slack values for all pins of a design and that it reports timing problems for instance as lists of critical paths. Whereas most procedures for Boolean optimization of combinational logic are either purely heuristic or rely on exhaustive enumeration and are thus very time consuming, our approach is much more effective.

We consider a critical path P which combines a number of signals $x_1, x_2, \ldots, x_n$ arising at certain times $AT(x_i)$ and locations $Pl(x_i)$ by a sequence $g_1, g_2, \ldots, g_{n-1}$ of 2-input gates as in Figure 16. Then we try to re-synthesize P in a best possible way.

CRITICAL PATH LOGIC RESYNTHESIS PROBLEM

Instance:

- A set X of sources and a sink y,
- a start time $AT(x)$ (and possibly a slew) at each source $x \in X$ and a required arrival time $RAT(y)$ at the sink,
- placement information for the sources and the sink $Pl(x_1)$, $Pl(x_2)$, ..., $Pl(x_n)$, $Pl(y) \in [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$,
- a Boolean expression of the form

$$y = f(x_1, x_2, ..., x_n) = g_{n-1}(...g_3(g_2(g_1(x_1, x_2), x_3), x_4)..., x_n)$$

  where the g_i are elementary Boolean functions,
- physical information about the driver strength of the sources and the input capacitances of the sink, and
- physical information about the wiring and the library L of available elementary logical circuits (`and`, `or`, `nand`, `nor`, `invert`,...).

Task: Find a circuit representation of y as a function of the $x \in X$

- using elementary Boolean circuits,
- together with placement and sizing information for the circuits such that
- the computation of y completes before $RAT(y)$, or as early as possible.

Our algorithm first generates a standard format. It decomposes complex circuits on P into elementary `and`- and `or`-circuits with fan-in two plus inversions. Applying the de Morgan rules we eliminate all inversions except for those on input signals of P. We arrive at a situation in which P is essentially represented by a sequence of `and`- and `or`-circuits. Equivalently, we could describe the procedure using `nand`-circuits only, and we will indeed use `nand`s for the final realization. However, for the sake of a simpler description of our algorithm, `and`- and `or`-circuits are more suitable.

We now design an alternative, logically equivalent representation of the signal produced by g_m as a function of the x_i in such a way that late input signals do not pass through too many logic stages of this alternative representation. This is easy if this sequence consists either just of `and`-circuits or just of `or`-circuits. In this situation every binary tree with n leaves leads to a representation of the Boolean function by identifying its leaves with the x_i and its internal nodes with `and`-circuits or `or`-circuits. If we consider only the arrival times $AT(x_i)$ of the signals which might be justified because all locations are close together, we can easily construct and implement an optimal representation using Huffman coding. If we consider both $AT(x_i)$ and $Pl(x_i)$, then inverting time the problem is actually equivalent to the construction of a fan-out tree which we have described in Section 3.2.

The most difficult case occurs if the `and`- and `or`-circuits alternate, i.e. the function calculated by P is of the form

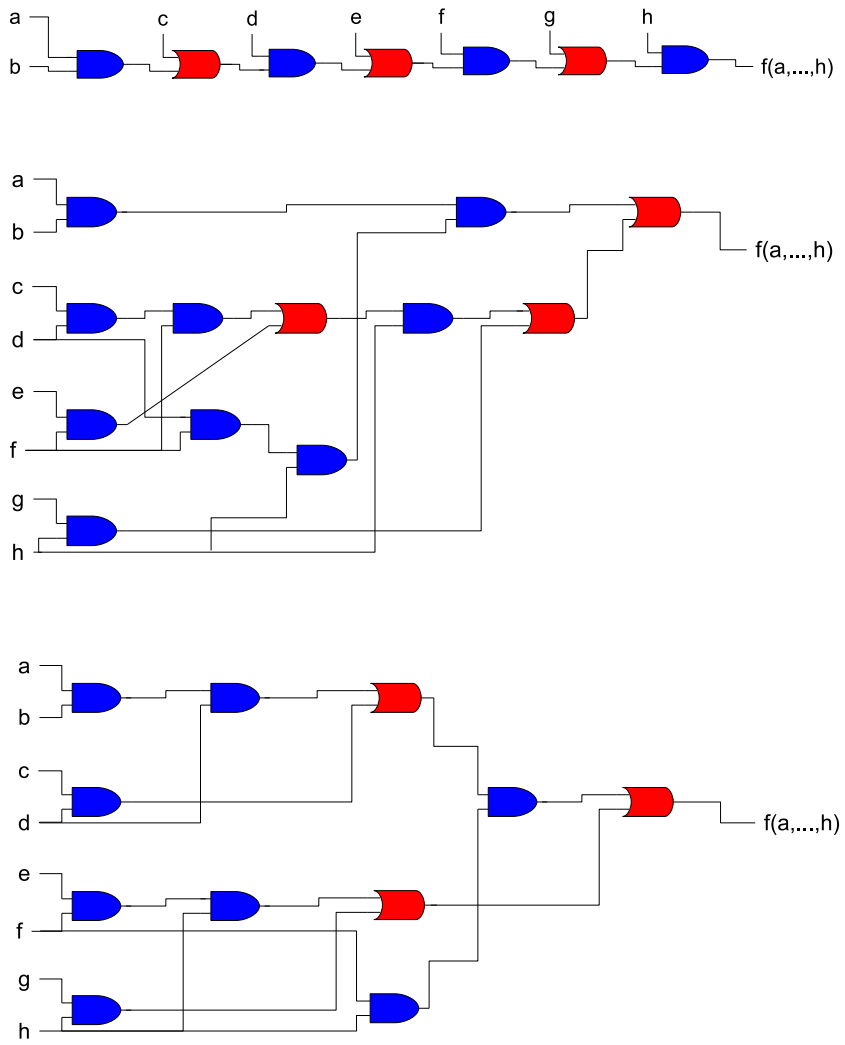

Figure 17. Three logically equivalent circuits for the function $f(a, b, ..., h)$ that correspond to the formulas $f(a, ..., h) = ((((((a \wedge b) \vee c) \wedge d) \vee e) \wedge f) \vee g) \wedge h$, $f(a, ..., h) = ((a \wedge b) \wedge ((d \wedge f) \wedge h)) \vee (((((c \wedge d) \wedge f) \vee (e \wedge f)) \wedge h) \vee (g \wedge h))$, and $f(a, ..., h) = ((((a \wedge b) \wedge d) \vee (c \wedge d)) \wedge (f \wedge h)) \vee (((e \wedge f) \wedge h) \vee (g \wedge h))$. The first path is a typical input of our procedure and the two alternative netlists have been obtained by the dynamic programming procedure based on the identity (7). Ignoring wiring and assuming unit delays for the circuits, the second netlist would for instance be optimal for $AT(a) = AT(b) = AT(g) = AT(h) = 3$, $AT(e) = AT(f) = 1$, and $AT(c) = AT(d) = 0$, leading to an arrival time of 6 for $f(a, ..., h)$ instead of 10 in the input path.

$$f(x_1, x_1', x_2, x_2', \ldots, x_n, x_n') := ((\cdots(((x_1 \wedge x_1') \vee x_2) \wedge x_2') \cdots) \vee x_n) \wedge x_n')$$
$$= \bigvee_{i=1}^{n} \left(x_i \wedge \left(\bigwedge_{j=i}^{n} x_j' \right) \right).$$

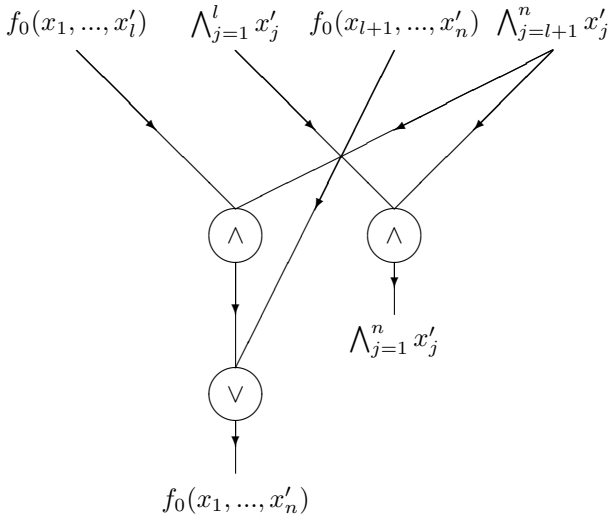

Figure 18. The logic circuit corresponding to equation (7)

See Figure 17 for an example. In this case we apply dynamic programming based on identities like the following:

$$f(x_1, \ldots, x'_n) \;=\; \left(f(x_1, \ldots, x'_l) \wedge \left(\bigwedge_{j=l+1}^{n} x'_j \right) \right) \vee\; f(x_{l+1}, \ldots, x'_n). \qquad (7)$$

Note that equation (7) corresponds to a circuit structure as shown in Figure 18.

Our dynamic programming procedure maintains sets of useful sub-functions such as $f(x_i, \ldots, x'_j)$ and $\bigwedge_{k=i}^{j} x'_k$ together with estimated timing and placement information. In order to produce the desired final signal, these sets of sub-functions are combined using small sets of circuits as shown for instance in Figure 18, and the timing and placement information is updated. We maintain only those representations that are promising. The final result of our algorithm is found by backtracking through the data accumulated by the dynamic programming algorithm. After having produced a faster logical representation, we apply de Morgan rules once more and collapse several consecutive elementary circuits to more complex ones if this improves the timing behavior. In many cases this results in structures mainly consisting of `nand`-circuits and inverters. Certainly, the number of circuits used in the new representation is typically larger than in the old representation but the increase is at most linear.

Our procedure is very flexible and contains the purely local changes as a special case. Whereas the dynamic programming procedure is quite practical and easily allows us to incorporate physical insight as well as technical constraints, we can validate its quality theoretically by proving interesting optimality statements. In order to give an example for such a statement, let us neglect placement information, assume non-negative integer arrival times and further assume a unit delay for `and`- and `or`-circuits. We proceed in two steps. First, we derive a lower bound on the arrival time of desired signal and then estimate the arrival time within our new logical representation.

Theorem 6. *If C is a circuit of fan-in 2 for some Boolean function f depending on the inputs $x_1, x_2, ..., x_n$ with arrival times $AT_1, AT_2, ..., AT_n \in \mathbb{N}_0$, then*

$$AT(f, C) \geq \left\lceil \log_2 \left(\sum_{i=1}^{n} 2^{AT_i} \right) \right\rceil,$$

where $AT(f, C)$ denotes the arrival time of the value of f as computed by C assuming a unit delay for every circuit.

Proof: The existence of a circuit C of fan-in 2 that calculates the value of f by the time T implies the existence of a rooted binary tree with n leaves of depths $(T - AT_1), (T - AT_2), ..., (T - AT_n) \in \mathbb{N}_0$. By Kraft's inequality, such a tree exists if and only if $\sum_{i=1}^{n} 2^{-(T-AT_i)} \leq 1$ or, equivalently, $T \geq \log_2 \left(\sum_{i=1}^{n} 2^{AT_i} \right)$, and the proof is complete. □

In order to estimate the arrival time within the new logical representation we have to analyze the growth behavior of recursions based on the decomposition identities used during the dynamic programming. If we just use (7) for instance, then we have to estimate the growth of the following recursion which reflects the additional delays incured by the three circuits in Figure 18: For $n \geq 2$ and non-negative integers $a, a_1, \ldots, a_n \in \mathbb{N}_0$ let

$$AT(a) = a$$
$$AT(a_1, ..., a_n) = \min_{1 \leq l \leq n-1} \max\{AT(a_1, ..., a_l) + 2, AT(a_{l+1}, ..., a_n) + 1\}.$$

We have demonstrated how to analyze such recursions in [79,78,81] and how to obtain results like the following.

Theorem 7. *If $a_1, a_2, ..., a_n \in \mathbb{N}_0$, then*

$$AT(a_1, a_2, ..., a_n) \leq 1.44 \log_2 \left(\sum_{i=1}^{n} 2^{a_i} \right) + 2.$$

Comparing the bounds in Theorems 6 and 7 implies that just using (7) during the dynamic programming would lead to an algorithm with asymptotic approximation ratio 1.44 [79]. Using further decomposition identities this can be reduced to $(1 + \epsilon)$ for arbitrary $\epsilon > 0$ [78]. Further details about practical application and computational results can be found in [82].

So far we have described an algorithm for the redesign of a critical path which is in fact a main issue during timing closure. This algorithm was "blind" for the actual function that was involved and hence applicable to almost every critical path. We have also devised procedures for the timing-aware design of more complex functions. As an example consider the following so-called prefix problem which is essential for the construction of fast binary adders.

PREFIX PROBLEM	
Instance:	An associative operation $\circ : D^2 \to D$ and inputs $x_1, x_2, ..., x_n \in D$.
Task:	Find a circuit computing $x_1 \circ x_2 \circ ... \circ x_i$ for all $1 \leq i \leq n$.

Applying the above algorithm to the n desired output functions would lead to a circuit with good delay properties but with a quadratic number of circuits. Similar constructions with close-to-optimal delay but quadratic size were described also by Liu et al. in [56]. In [80] we constructed circuits solving the prefix problem with close-to-optimal delay and much smaller sizes of $O(n \log(\log(n)))$.

For the addition of two n-bit binary numbers whose $2n$ bits arrive at times $t_1, t_2, ..., t_{2n} \in \mathbb{N}_0$ this leads to circuits over the basis $\{\vee, \wedge, \neg\}$ of fan-in 2 for $\vee$- or $\wedge$-circuits and fan-in 1 for $\neg$-circuits calculating the sum of the two numbers with size $O(n \log(\log(n)))$ and delay

$$2 \log_2 \left(\sum_{i=1}^{2n} \left(2^{t_i} \right) \right) + 6 \log_2 (\log_2 (n)) + O(1).$$

In view of Theorem 6, the delay bound is close-to-optimal and the bound on the size is optimal up to a factor of $O(\log(\log(n)))$. The best known adders are of depth $\log_2(n) + O\left(\sqrt{\log(n)}\right)$ and size $O(n \log(n))$ [19] or size $O(n)$ [46], respectively. The adder developed in [99], which takes arrival times into account, has size $O(n \log(n))$, but no delay bound has been proved.

3.4. Gate sizing and V_t-assignment

The two problems considered in this section consist of making individual choices from some discrete sets of possible physical realizations for each circuit of the netlist such that some global objective function is optimized.

For gate sizing one has to determine the size of the individual circuits measured for instance by their area or power consumption. This size affects the input capacitance and driver strength of the circuit and therefore has an impact on timing. A larger circuit typically decreases downstream delay and increases upstream delay. Circuits with a single output pin are called (logic) gates. Assuming gates instead of general multi-output circuits simplifies mathematical problem formulations. This is the reason why the problem is called rather gate sizing than circuit sizing.

Whereas the theoretically most well-founded approaches for the gate sizing problem rely on convex/geometric programming formulations [8,22,29], these approaches typically suffer from their algorithmic complexity and restricted timing models. In many situations, approaches that choose circuit sizes heuristically can produce competitive results because it is much easier to incorporate local physical insight into heuristic selection rules than into a sophisticated convex program. Furthermore, the convex programming formulations often assume continuous circuit size while standard cell libraries typically only offer a discrete set of different sizes. In BonnOpt we use both, a global formulation and convex programming for the general problem as well as heuristics for special purposes.

For the simplest form of the global formulation we consider a directed graph G which encodes the netlist of the design. G can be considered as the timing graph $G_{\mathcal{T}}$ from

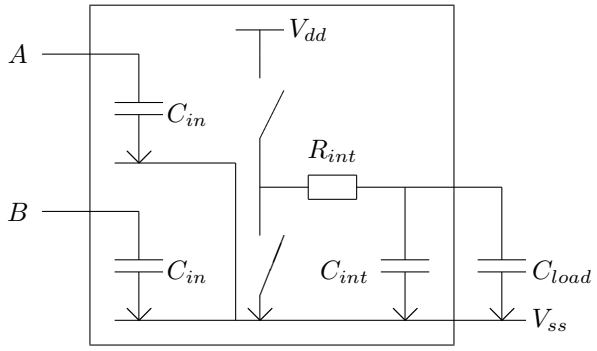

Figure 19. A simple electrical model of a circuit. The input capacitance C_{in} and the internal capacitance C_{int} are proportional to the scaling factor of the circuit while the internal resistance R_{int} is antiproportional.

Section 3.1 after contracting all input pin vertices. For a set V_0 of nodes v — e.g., start and end nodes of maximal paths — we are given signal arrival times a_v and we must choose circuit sizes $x = (x_v)_{v \in V(G)} \in [l, u] \subseteq \mathbb{R}^{V(G)}$ and arrival times for nodes not in V_0 minimizing

$$\sum_{v \in V(G)} x_v$$

subject to the timing constraints

$$a_v + d_{(v,w)}(x) \leq a_w$$

for all arcs $(v, w) \in E(G)$. The circuit sizes x_v are scaling factors for the internal structures of the circuit v (see Figure 19).

For simplicity it is assumed that the input and internal capacitances of circuit v are proportional to x_v while the internal resistance is antiproportional to x_v. Using the Elmore delay model [27], the delay through circuit v is of the form $R_{int}(C_{int} + C_{load})$ where C_{load} is the sum of the wire capacitance and the input capacitances of the structures that are charged over the circuit v. Since C_{load} depends on the circuit sizes of the corresponding circuits in the same way, the delay $d_{(v,w)}(x)$ of some arc (v, w) of G is modeled by a linear function with positive coefficients depending on terms of the form x_v, $\frac{1}{x_v}$ and $\frac{x_w}{x_v}$. Dualizing the timing constraints via Lagrange multipliers $\lambda_{(u,v)} \geq 0$ leads to the following Lagrange function.

$$
\begin{aligned}
& L(x, a, \lambda) \\
&= \sum_{u \in V(G)} x_u + \sum_{(u,v) \in E(G)} \lambda_{(u,v)} \left(a_u + d_{(u,v)}(x) - a_v \right) \\
&= \sum_{u \in V(G)} x_u + \sum_{(u,v) \in E(G)} \lambda_{(u,v)} d_{(u,v)}(x) + \sum_{(u,v) \in E(G)} \lambda_{(u,v)} (a_u - a_v) \\
&= \sum_{u \in V(G)} x_u + \sum_{(u,v) \in E(G)} \lambda_{(u,v)} d_{(u,v)}(x) \\
&\qquad + \sum_{u \in V(G)} a_u \left(\sum_{v:(u,v) \in E(G)} \lambda_{(u,v)} - \sum_{v:(v,u) \in E(G)} \lambda_{(v,u)} \right).
\end{aligned}
$$

Since after the dualization all arrival times $(a_v)_{v \in V(G)}$ (except those that are constant) are free variables, every optimal solution of the dual maximization problem has the property that

$$
\sum_{u \in V(G)} a_u \left(\sum_{v:(u,v) \in E(G)} \lambda_{(u,v)} - \sum_{v:(v,u) \in E(G)} \lambda_{(v,u)} \right) = 0
$$

for all choices of a_u, i.e. the Lagrange multipliers $\lambda_{(u,v)} \geq 0$ constitute a non-negative flow on the timing graph [22].

Therefore, for given Lagrange multipliers the problem reduces to minimizing a weighted sum of the circuit sizes x and delays $d_{(u,v)}(x)$ subject to $x \in [l, u]$. This step is typically called local refinement. Generalizing results from [22,24,25,55] we proved that it can be solved by a very straightforward cyclic relaxation method with linear convergence rate in [77]. The overall algorithm is the classical constrained subgradient projection method (cf. [60]). The known convergence guarantees for this algorithm require an exact projection, which means that we have to determine the above-mentioned non-negative flow on G that is closest to some given vector $(\lambda_e)_{e \in E}$.

Since this exact projection is actually the most time-consuming part, practical implementations use crude heuristics having unclear impact on convergence and quality. To overcome this limitation, we proved in [76] that the convergence of the algorithm is not affected by executing the projection in an approximate and much faster way. This is done by combining the subgradient projection method [70,71] in a careful way with the method of alternating projections [23] and results in a stable, fast, and theoretically well-founded implementation of the subgradient projection procedure for circuit sizing.

Nevertheless, in practice there exist advanced heuristics that are also good and much faster than the subgradient method. Such approaches improve all circuits iteratively based on the "dual" slack values. The improvement does not follow a strict mathematical formula but makes reasonable choices heuristically. Furthermore, time-consuming slack updates are not done after every single cell change but only once per iteration. Additional side constraints like load and slew limits, or placement density can be incorporated easily.

In [37], we developed a heuristic that yields competitive results compared to continuous mathematical optimization models, which lack accuracy due to simplified delay models and rounding errors. The worst path delays are within 6% of a lower delay bound on average.

Global circuit sizing approaches are followed by local search on the critical paths. Here delay effects due to layout changes are computed accurately and slacks are updated after every circuit change. As accurate computations are extremely time consuming, only a few circuits (approximately 1%) are considered by this local search. Afterwards the worst path delays are within 2% of a lower delay bound on average.

The second optimization problem that we consider in this section is V_t-assignment. A physical consequence of feature size shrinking is that leakage power consumption represents a growing part of the overall power consumption of a chip. Increasing the threshold voltage of a circuit reduces its leakage but increases its delay. Modern libraries offer circuits with different threshold voltages. The optimization problem that we face is to choose the right threshold voltages for all circuits, which minimize the overall (leakage) power consumption while respecting timing restrictions.

As proposed in [84,36], we first consider a netlist in which every circuit is realized in its slowest and least-leaky version. We define an appropriate graph G whose arcs are assigned delays, and some of whose arcs correspond to circuits for which we could choose a faster yet more leaky realization. For each such arc e we can estimate the power cost c_e per unit delay reduction. We add a source node s joined to all primary inputs and to all output nodes of memory elements and a sink node t joined to all primary outputs and to all input nodes of memory elements. Then we perform a static timing analysis on this graph and determine the set of arcs E' that lie on critical paths.

The general step now consists in finding a cheapest s-t-cut $(S, \bar{S})$ in $G' = (V(G), E')$ by a max-flow calculation in an auxiliary network. Arcs leaving S that can be made faster contribute c_e to the cost of the cut, and arcs entering S that can be made slower contribute $-c_e$ to the cost of the cut. Furthermore, arcs leaving S that cannot be made faster contribute ∞ to the cost of the cut, and arcs entering S that cannot be made slower contribute 0 to the cost of the cut.

If we have found such a cut of finite cost, we can improve the timing at the lowest possible power cost per time unit by speeding up the arcs from S to $\bar{S}$ and slowing down (if possible) the arcs from $\bar{S}$ to S. The acceleration is performed until non-accelerated paths become critical and the next iteration is performed on a growing auxiliary network. Figure 20 illustrates the algorithm. The optimality statement is proved in [69] subject to the simplifying assumptions that the delay/power dependence is linear and that we can realize arbitrary V_t-values within a given interval, which today's libraries typically do not allow. Nevertheless, the linearity of the delay/power dependence approximately holds locally and the discrete selectable values are close enough. There are strong connections to the so-called discrete time-cost tradeoff problem as studied for instance in [88].

We point out that the described approach is not limited to V_t-assignment. It can be applied whenever we consider roughly independent and local changes and want to find an optimal set of operations that corrects timing violations at minimum cost. This has been part of BonnTools for some time [28], but previously without using the possibility of slowing arcs from $\bar{S}$ to S, and thus without optimality properties.

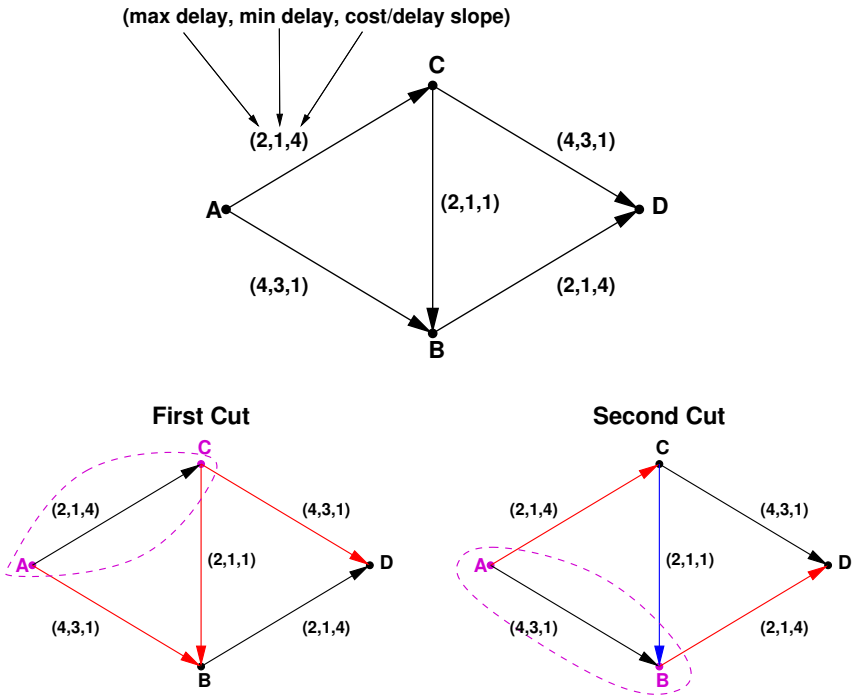

Figure 20. A time-cost-tradeoff instance (top). For each arc the maximum delay, the minimum delay, and the cost increase per unit delay decrease are specified. Initially every arc delay is chosen slowest possible. All three paths have the same delay of 6 time units. The minimum A-D-cut is $(\{A, C\}, \{B, D\})$ with value $3 = 1 + 1 + 1$ (bottom left). After accelerating the involved arcs (A, B), (C, B) and (C, D) all paths have delay 5. Now the minimum cut is $(\{A, B\}, \{C, D\})$ with value $7 = 4 + 4 - 1$ (bottom right). Note that (A, C) and (B, D) are leaving the cut and therefore accelerated, but (C, B) is entering the cut and decelerated. All arcs except (C, B) reached their minimum delay therefore have infinity capacitance. Now the minimum cut has weight infinity and the algorithm stops. The critical paths $A \to B \to D$ and $A \to C \to D$ cannot be accelerated further.

4. Clock scheduling and clock tree construction

Most computations on chips are synchronized. They are performed in multiple cycles. The result of a cycle is stored in special memory elements (registers, flip-flops, latches) until it is used as input for the next cycle. Each memory element receives a periodic clock signal, controlling the times when the bit at the data input is to be stored and transferred to further computations in the next cycle. Today it is well-known that striving for simultaneous clock signals (zero skew), as most chip designers did for a long time, is not optimal. By clock skew scheduling, i.e. by choosing individual clock signal arrival times for the memory elements, one can improve the performance. However, this also makes clock tree synthesis more complicated. For nonzero skew designs it is very useful if clock tree synthesis does not have to meet specified points in time, but rather time intervals. We proposed this methodology together with new algorithms in [3], [4], and [35]. Since then it has been successfully applied on many industrial high performance ASICs.

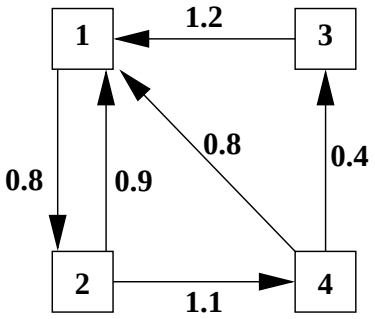

Figure 21. A latch graph with 4 latches (numbered boxes). The arc numbers specify the longest path delays. With a zero skew tree — when all latches switch simultaneously — the slowest path ($3 \to 1$) determines the cycle time $T_{zs} = 1.2$. With an optimum scheduled tree the slowest average delay cycle ($1 \to 2 \to 4 \to 1$) determines the cycle time $T_{opt} = 0.9$.

4.1. Clock skew scheduling

Let us define the latch graph as the digraph whose vertex set is the set of all memory elements and which contains an arc (v, w) if the netlist contains a path from the output of v to the input of w. Let $d_{(v,w)}$ denote the maximum delay of a path from v to w. If all memory elements receive a periodic clock signal of the same frequency $\frac{1}{T}$ (i.e. their cycle time is T), then a zero skew solution is feasible only if all delays are at most T. Figure 21 shows a latch graph with four latches. In this example the minimum cycle time with a zero skew tree would be 1.2, bounded by the path $3 \to 1$. With clock skew scheduling one can relax this condition. Let latch 3 in the above example switch earlier by 0.2 time units. Now signals on path $3 \to 1$ could spent 0.2 more time units per cycle. In turn the maximum allowed delay for signals on path $4 \to 3$ would decrease by 0.2 time units, which would not harm the overall cycle time as the path delay of 0.4 is very fast. Now path $2 \to 4$ determines a limit for the minimum cycle time of 1.1.

Motivated by this observation, the question arises how much can we improve the performance for given delays? We ask for arrival times a_v of clock signals at all memory elements x such that

$$a_v + d_{(v,w)} \leq a_w + T \tag{8}$$

holds for each arc (v, w) of the latch graph. We call such arrival times *feasible*. Now the best achievable cycle time T due to clock scheduling is given by following theorem.

Theorem 8. *Given a latch graph G with arcs delays d, the minimum cycle time T for which feasible arrival times $a : V(G) \to \mathbb{R}$ exist equals the maximum mean delay $\frac{d_{E(C)}}{|E(C)|}$ of a directed cycle in C in G.*

Proof: T is feasible if and only if there are arrival times a such that:

$$a_c + d_{(v,w)} \leq a_w + T \quad \forall (v, w) \in E(G).$$

From shortest path theory it is known that such arrival times (node potentials) exist if and only if (G, c) does not contain any cycle with negative total cost, where $c(e) := T - d_e$. Equivalently,

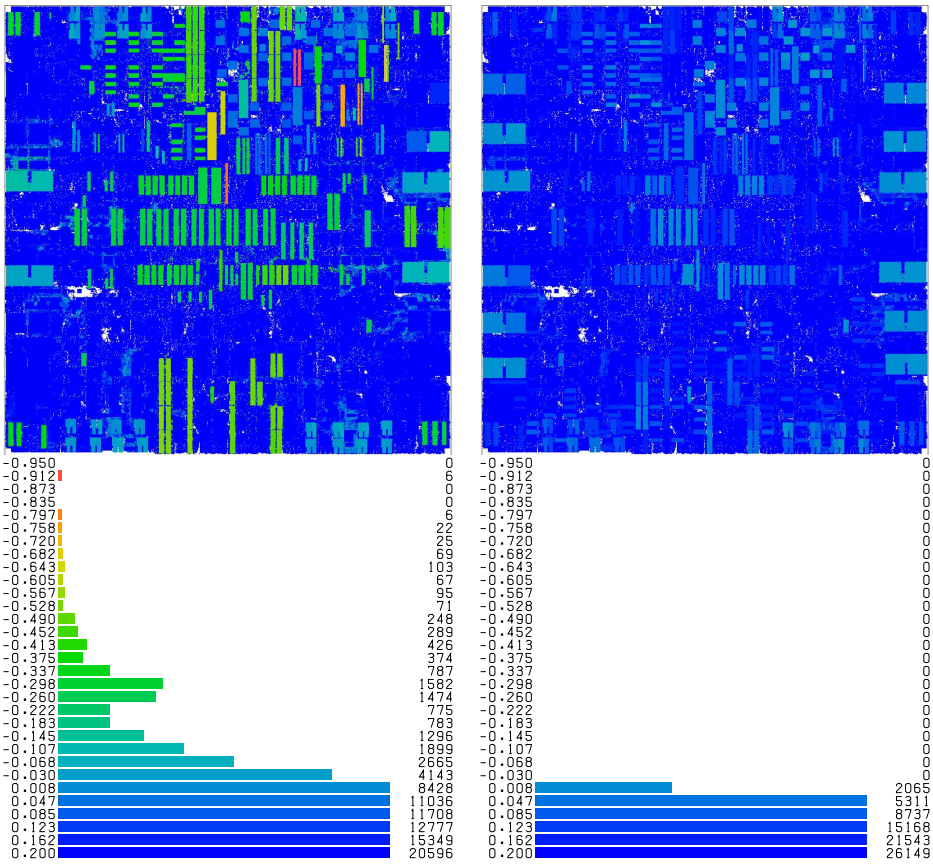

Figure 22. Slack histograms showing the improvement due to clock skew scheduling and appropriate clock tree synthesis; left: zero skew, right: with BonnClock trees. Each histogram row represents a slack interval (in ns) and shows the number of circuits with their worst slacks in this range. The placements on top are also colored according to the worst circuit slacks.

$$T \geq \frac{d_{E(C)}}{|E(C)|} \quad \text{for all cycles } C \text{ in } G.$$

This shows that the minimum possible T equals the longest average delay of a cycle. □

In the example of Figure 21 this gives an optimum cycle time of 0.9, which can be computed easily by enumerating all three cycles. In general, the optimal feasible cycle time T and feasible clock signal arrival times $a(T)$ can be computed by minimum mean cycle algorithms, e.g. those of Karp [45] or Young, Tarjan, and Orlin [100].

However, this simple situation is unrealistic. Today systems on a chip have multiple frequencies and often several hundred different clock domains. The situation is further complicated by transparent latches, user-defined timing tests, and various advanced design methodologies.

Moreover, it is not sufficient to maximize the frequency only. The delays that are input to clock skew scheduling are necessarily estimates: detailed routing will be done later and will lead to different delays. Thus one would like to have as large a safety margin — positive slack — as possible. In analogy to the slack definition from Section 3.1 the arc

slack $\sigma_{(v,w)}$ for given T and a is defined as $\sigma_{(v,w)} := a_w - a_v - d_{(v,w)} + T$ for every arc $(v, w) \in E(G)$. Note that $a_w + T$ defines a required arrival time for signals entering latch w. In fact, maximizing the worst slack by clock scheduling is equivalent to minimizing the cycle time:

Proposition 9. *Let G be a latch graph with arc delays d. Let T' be the minimum possible cycle time for (G, d), $T > 0$, and*

$$\sigma'_T = \max_a \min_{(v,w)\in E(G)} \big(a_w - a_v - d_{(v,w)} + T\big)$$

be the maximum achievable worst slack for cycle time T. Then

$$T' = T - \sigma'_T.$$

Proof: This follows from the observation that $\sigma'_{T'} = 0$ and the difference $T - \sigma'_T$ is invariant in T. □

In practice the cycle time is a fixed input parameter and clock scheduling is used to achieve this cycle time and optimize the overall slack distribution.

Next, signals can also be too fast. This means that we are also given minimum delays $\delta_{(v,w)}$ of each path $(v, w) \in E(G)$, and a signal must not arrive in the previous cycle:

$$a_v + \delta_{(v,w)} \geq a_w, \quad \forall (v, w) \in E(G).$$

Although such early-mode violations can be repaired by delay insertion via buffering, this can be very expensive in terms of placement and wiring resources as well as power consumption. Clock skew scheduling can remove most early-mode violations at almost no cost.

Finally, it is very hard to realize arbitrary individual arrival times exactly; moreover this would lead to high power consumption in clock trees. Computing time intervals rather than points in time is much better. Without making critical paths any worse, the power consumption (and use of space and wiring resources) by clock trees can be reduced drastically. Intervals for the clock arrival times can be introduced by splitting every $v \in V(G)$ into two nodes v_l, v_u and adding the constraints $a_{v_l} \leq a_{v_u}$. Delay constraints have to be reconnected to the corresponding split nodes, i.e. $a_v + d_{(v,w)} \leq a_w + T$ is replaced by $a_{v_u} + d_{(v,w)} \leq a_{w_l} + T$. Now $[a_{v_l}, a_{v_u}]$ defines an admissible arrival time interval for v.

A three-stage clock skew scheduling approach was proposed by Albrecht et al. [4]. Firstly, only late-mode slacks are considered. Then early-mode violations are reduced (more precisely, slacks that can be increased by inserting extra delays), without decreasing any negative or small positive late-mode slacks. Thirdly, a time interval for each memory element is computed such that whenever each clock signal arrives within the specified time interval, no negative or small positive slack will decrease. In the next section we discuss how to balance a certain set of slacks while not decreasing others.

4.2. Slack balancing

In [4,34,36], generalizing the early work of Schneider and Schneider [85] and Young, Tarjan and Orlin [100], we have developed slack balancing algorithms for very general situations. The most general problem can be formulated as follows.

THE SLACK BALANCING PROBLEM

Instance: A directed graph G (the timing graph), $c : E(G) \to \mathbb{R}$ (delays), a set $F_0 \subseteq E(G)$ (arcs where we are not interested in positive slack) and a partition $\mathcal{F}$ of $E(G) \setminus F_0$ (groups of arcs in which we are interested in the worst slack only), and weights $w : E(G) \setminus F_0 \to \mathbb{R}_{>0}$ (sensitivity of slacks), such that there are no positive delay cycles in (V, F_0).

Task: Find arrival times $\pi : V(G) \to \mathbb{R}$ with

$$\pi(v) + c(e) \leq \pi(w) \text{ for } e = (v, w) \in F_0 \tag{9}$$

such that the vector of relevant slacks

$$\left(\min \left\{ \frac{\pi(w) - \pi(v) - c(e)}{w(e)} \,\middle|\, e = (v, w) \in F \right\} \right)_{F \in \mathcal{F}} \tag{10}$$

(after sorting entries in non-decreasing order) is lexicographically maximal.

Note that the delays c include cycle adjusts and thus can be negative (for the latch graph example above $c(e)$ is the propagation delay minus the cycle time T). The conditions for $e \in F_0$ correspond to edges on which slack must be non-negative, but the actual amount is not of interest. In the following we denote $\sigma_\pi(e) := \pi(w) - \pi(v) - c(e)$ as the slack of $e \in E(G)$ with respect to the node potential π.

An alternative formulation of the SLACK BALANCING PROBLEM is given by the following theorem.

Theorem 10. *Let $(G, c, w, \mathcal{F})$ be an instance of the* SLACK BALANCING PROBLEM. *Let $\pi : V(G) \to \mathbb{R}$ with $\sigma_\pi(e) \geq 0$ for $e \in F_0$. For $F \in \mathcal{F}$ define*

$$F_\pi := \left\{ e \in F \,\middle|\, \frac{\sigma_\pi(e)}{w(e)} \textit{ minimal in } F \right\}.$$

and $E_\pi = \bigcup_{F \in \mathcal{F}} F_\pi$. Then π is an optimum solution if and only if there are no $F \in \mathcal{F}$ and $X_f \subset V(G)$ for $f \in F_\pi$ such that

$$\begin{aligned} & f \in \delta^-(X_f) && \textit{for } f \in F_\pi, \\ & \sigma_\pi(e) > 0 && \textit{for } f \in F_\pi,\ e \in \delta^+(X_f) \cap F_0, \textit{ and} \\ & \frac{\sigma_\pi(e)}{w(e)} > \frac{\sigma_\pi(f)}{w(f)} && \textit{for } f \in F_\pi,\ e \in \delta^+(X_f) \cap E_\pi. \end{aligned} \tag{11}$$

Proof: If there is an $F \in \mathcal{F}$ and $X_f \subset V(G)$ for $f \in F_\pi$ with (11), then π is not optimum, because setting $\pi'(v) := \pi(v) - \epsilon \cdot |\{f \in F_\pi : v \in X_f\}|$ for $v \in V(G)$ for a

sufficiently small $\epsilon > 0$ increases the sorted vector (10) lexicographically (disproving optimality).

Let now $\pi, \pi' : V(G) \to \mathbb{R}$ be two vectors with (9) and (11), and suppose there exists an $f = (p, q) \in E_\pi \cup E_{\pi'}$ with $\sigma_\pi(f) \neq \sigma_{\pi'}(f)$ and choose f such that $\frac{\min\{\sigma_\pi(f), \sigma_{\pi'}(f)\}}{w(f)}$ is minimum. Without loss of generality $\sigma_\pi(f) < \sigma_{\pi'}(f)$. Let $F \in \mathcal{F}$ be the set containing f. Then $f \in F_\pi$: The contrary assumption would imply that there exists $f' \in F_\pi$ with $\frac{\sigma_\pi(f')}{w(f')} < \frac{\sigma_\pi(f)}{w(f)}$. Then $\frac{\sigma_{\pi'}(f')}{w(f')} = \frac{\sigma_\pi(f')}{w(f')} < \frac{\sigma_\pi(f)}{w(f)} < \frac{\sigma_{\pi'}(f)}{w(f)}$ and thus $f \notin F_{\pi'}$, a contradiction.

For each $f' = (p, q) \in F_\pi$ let $X_{f'}$ be the set of all vertices reachable from q via arcs $e \in F_0$ with $\sigma_\pi(e) = 0$ or arcs $e \in E_\pi$ with $\frac{\sigma_\pi(e)}{w(e)} \leq \frac{\sigma_\pi(f)}{w(f)}$.

Then there exists an $f' = (p, q) \in F$ with $p \in X_{f'}$; for otherwise $(X_{f'})_{f' \in F}$ would satisfy (11). Hence there is a q-p-path P that consists only of arcs e with $\sigma_\pi(e) \leq \sigma_{\pi'}(e)$. Summation yields $\sigma_\pi(f) \geq \sigma_{\pi'}(f)$, a contradiction. □

This proof is essentially due to [97]. The special case $w \equiv 1, |F| = 1\ \forall F \in \mathcal{F}$, was considered by Albrecht [2], and the *minimum balance problem* ($w \equiv 1, F_0 = \emptyset, \mathcal{F} = \{\{e\} | e \in E(G)\}$) by Schneider and Schneider [85].

Now we show how to solve the SLACK BALANCING PROBLEM.

Theorem 11. *The* SLACK BALANCING PROBLEM *can be solved in strongly polynomial time* $O(I \cdot (n^3 \log n + \min\{nm, n^3\} \cdot \log^2 n \log\log n + nm \log m))$, *where* $I := n + |\{F \in \mathcal{F} \ ; \ |F| > 1\}|$, *or in pseudo-polynomial time* $O(w_{\max}(nm + n^2 \log n) + I(n \log n + m))$ *for integral weights* $w : E(G) \to \mathbb{N}$ *with* $w_{\max} := \max\{w(e) | e \in E \setminus F_0\}$.

Sketch of proof: We may assume that $(V(G), F_0)$ contains no circuit of positive total weight. Set $w(e) := 0$ for $e \in F_0$ Analogously to the proof of Theorem 8, it is easy to see that the maximum worst weighted slack is given by the negative maximum weighted delay $-\frac{c(E(C))}{w(E(C))}$ of a cycle C in G with $w(E(C)) > 0$ and that arrival times exist that achieve this value.

Thus, an optimum solution for the SLACK BALANCING PROBLEM can be obtained by iteratively identifying the maximum weighted delay cycle C, resolving all intersecting partitions, and and just preserving their respective minimum weighted slacks $-\frac{c(E(C))}{w(E(C))}$ in subsequent iterations. Algorithm 2 describes the overall procedure.

In each iteration, the edges $e \in F \cap E(C)$ determine the final worst weighted slack $\min\{\sigma_\pi(f)/w(f) \ : \ f \in F\}$ for their sets F. In lines 4–10 we fix the slack on all edges in sets $F \in \mathcal{F}$ that intersect C. Note that the delay and weighting modifications just preserve $\sigma_\pi(e)/w(e) \geq \lambda^\star$ in future iterations, but prevent the slacks of these edges from growing at the cost of less critical partitions.

If $|V(C)| > 1$, the critical cycle is contracted requiring adaption of incoming and outgoing edge costs in lines 12–17. Contraction may leave loops $e = (v, v)$ that can be removed unless $e \in F \in \mathcal{F}$ with $|F| > 1$. In this case it is not clear whether e or another edge from F will be the most critical edge in F. Thus, e must be kept, and may later lead to critical cycles C that are loops.

In each iteration either a cycle C with $|E(C)| > 1$ is contracted or, if C is a loop, a set F with $|F| > 1$ is removed from $\mathcal{F}$. Thus there are at most $I := n + |\{F \in \mathcal{F} \ ; \ |F| > 1\}|$ iterations of the while loop.

The dominating factor in each iteration is the computation of the maximum weighted delay cycle. This can be done in strongly polynomial time $O(\min\{n^3 \log^2 n + n^2 m \log m,$

```
1:  while (F ≠ ∅) do
2:      Compute the maximum weighted delay cycle C;
3:      λ* ← c(C)/w(C);
        /* Fixing (slack-preserving) delays in critical partitions */
4:      for F ∈ F with F ∩ E(C) ≠ ∅ do
5:          for f ∈ F do
6:              c(f) ← c(f) + λ*w(f);
7:              w(f) ← 0;
8:          end for
9:          F ← F \ {F};
10:     end for
        /* Critical cycle contraction*/
11:     if |V(C)| > 1 then
12:         for (x, y) ∈ δ⁻(V(C)) do
13:             c(x, y) ← c(x, y) − π(y);
14:         end for
15:         for (x, y) ∈ δ⁺(V(C)) do
16:             c(x, y) ← c(x, y) + π(x);
17:         end for
18:         Contract C;
19:         Remove irrelevant loops from G;
20:     end if
21: end while
```

Algorithm 2: Slack Balancing Algorithm

$n^3 \log n + n^2 m \log^2 n \log\log n\}$) by an adaption [34] of Megiddo's [59] minimum ratio cycle algorithm for non-simple graphs.

Alternatively, adopting the minimum balance algorithm of Young, Orlin and Tarjan [100] for our purpose, the cumulative running time for all cycle computations is bounded by the bound for a single maximum weighted delay cycle computation, which is $O(w_{\max}(mn + n^2 \log n)\})$. This is the fastest algorithm for small $w_{\max}$ (especially if $w : \mathbb{R} \to \{0, 1\}$). Detailed proofs can be found in [36], and in [4] for unit weights. □

In practice, the SLACK BALANCING PROBLEM can be solved much faster if we replace $\frac{\pi(w)-\pi(v)-c(e)}{w(e)}$ by $\min\{\Theta, \pi(w) - \pi(v) - c(e)\}$ in (10), i.e. ignore slacks beyond a certain threshold Θ, which we typically set to small positive values. In our three stage clock scheduling approach optimizing only slacks below some threshold is necessary to enable optimization in the next stage. Otherwise all arrival time relations would be fixed already.

Positive slacks which have been obtained in a previous stage should not be decreased in later stages. This can be modeled by increasing the corresponding delays and setting their weights to 0, analogously to Algorithm 2. Time intervals for clock signal arrival times also correspond to positive slack on arcs described in the last section. Time intervals are maximized by the same algorithm.

By working on the timing graph — a direct representation of the timing analysis constraints — rather than on the latch graph, we can consider all complicated timing

constraints, different frequencies, etc. directly. Furthermore its size is linear in the size of the netlist, while the latch graph can have a quadratic number of arcs and be much bigger.

On the other hand, in the timing graph model the vector of endpoints is optimized instead the vector of longest paths as in the latch graph model. In an optimum solution all paths entering the most critical cycle will obtain the most critical slack and give an overall worse timing result. In our experiments it turned out to be most efficient to use a combination of the latch graph — on most critical parts only — and the timing graph, incorporating the advantages of both models.

Figure 22 shows a typical result on a leading-edge ASIC. The left-hand side shows the slacks after timing-driven placement, but without clock skew scheduling, assuming zero skew and estimating the on-chip variation on clock tree paths with 300 ps. The right-hand side shows exactly the same netlist after clock skew scheduling and clock tree synthesis. The slacks have been obtained with a full timing analysis as used for sign-off, also taking on-chip variation into account. All negative slacks have disappeared. In this case we improved the frequency of the most critical clock domain by 27%. The corresponding clock tree is shown in Figure 25. It runs at 1.033 gigahertz [35], which is a very high frequency for an ASIC design even today, several years later. Next we explain how to construct such a clock tree, using the input of clock skew scheduling.

4.3. Clock tree synthesis

The input to clock tree construction is a set of sinks, a time interval for each sink, a set of possible sources, a logically correct clock tree serving these sinks, a library of inverters and other books that can be used in the clock tree, and a few parameters, most importantly a slew target. The goal is to replace the initial tree by a logically equivalent tree which ensures that all clock signals arrive within the specified time intervals.

Current ASIC chips contain several hundred clock tree instances with up to a million sinks. For gigahertz frequencies manufacturing process variations already dissipate 20–30% of the cycle time. Therefore clock trees have to be constructed very carefully, especially when realizing the delay targets induced by the arrival time windows.

Traditionally the individual delay constraints were met by balancing wires (cf. Chao et al. [21]). Theoretically very exact delay targets can be met by tuning wire delay. The drawback of this approach is that it often requires a lot of wiring resources, the prescribed wiring layout is hard to achieve in detailed routing (see Section 5.4), and due to increasing wiring resistances in new technologies delays increase significantly.

We proposed a different approach [35], which assumes all inserted wires to be routed shortest possible. Delays are balanced by the constructed tree topology and accurate gate sizing.

First, the input tree is condensed to a minimal tree by identifying equivalent books and removing buffers and inverter pairs. For simplicity we will assume here that the tree contains no special logic and can be constructed with inverters only.

Next we do some preprocessing to determine the approximate distance to a source from every point on the chip, taking into account that some macros can prevent us from going straight towards a source.

The construction then proceeds in a bottom-up fashion (cf. Figure 23). Consider a sink s whose earliest feasible arrival time is latest, and consider all sinks whose arrival time intervals contain this point in time. Then we want to find a set of inverters that drives

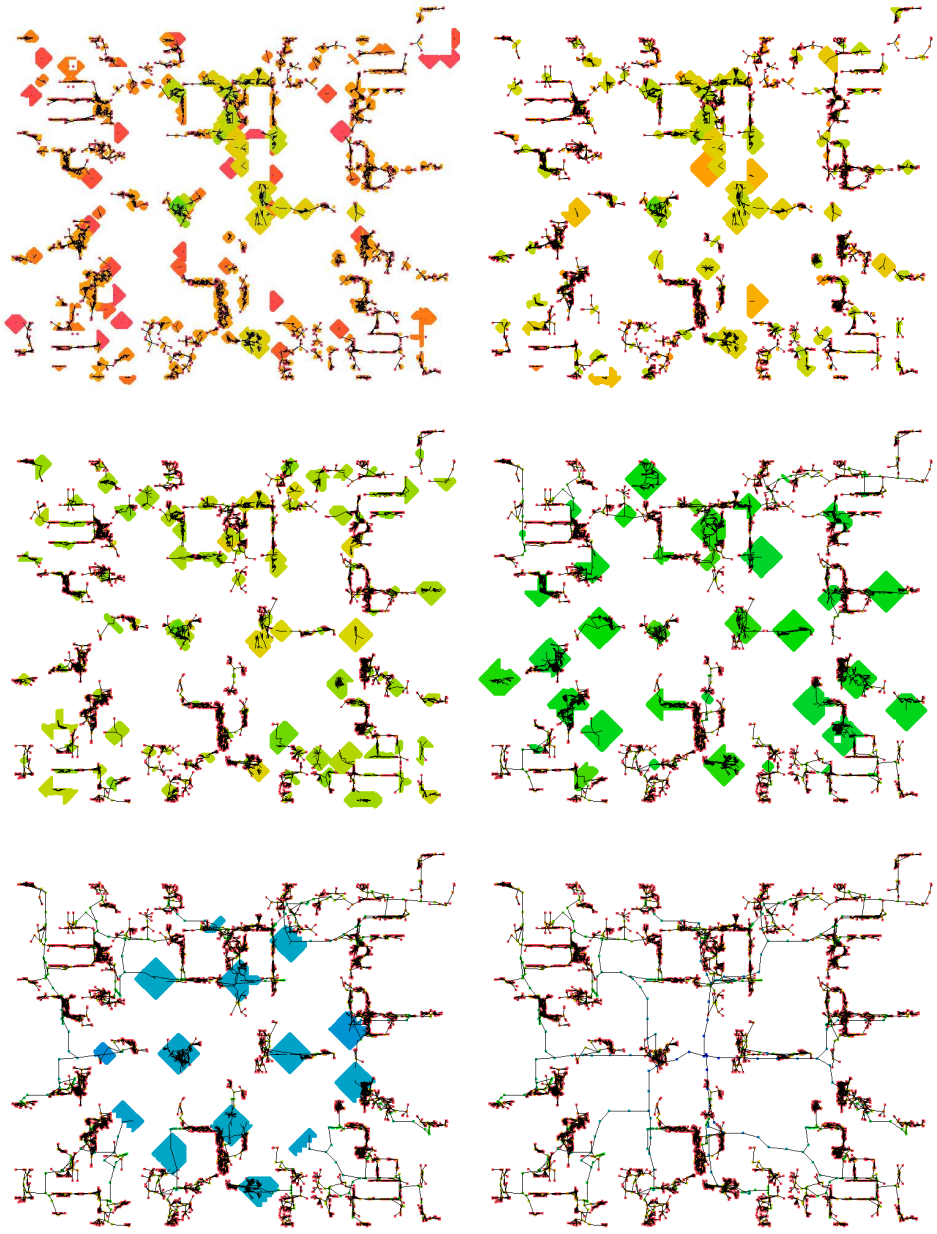

Figure 23. Different stages of a clock tree construction using BonnClock. The colored octagons indicate areas in which inverters (current sinks) can be placed. The colors correspond to arrival times within the clock tree: blue for signals close to the source, and green, yellow, and red for later arrival times. During the bottom-up construction the octagons slowly converge to the source, here located approximately at the center of the chip.

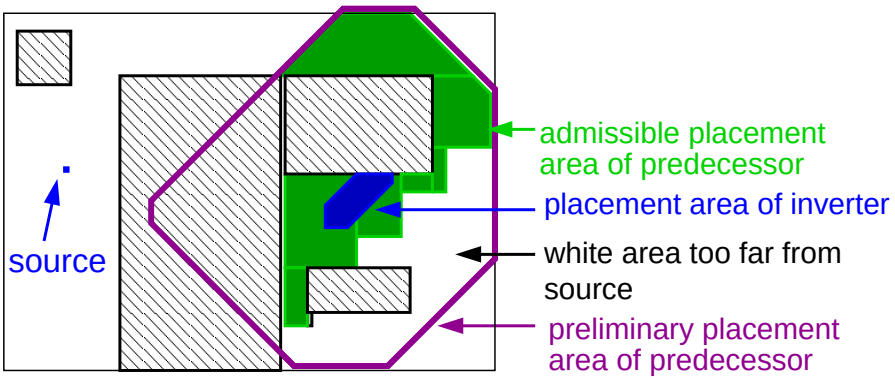

Figure 24. Computation of the feasible area for a predecessor of an inverter. From all points that are not too far away from the placement area of the inverter (blue) we subtract unusable areas (e.g., those blocked by macros) and points that are too far away from the source. The result (green) can again be represented as a union of octagons.

at least s but maybe also some of the other sinks. For each inverter we have a maximum capacitance which it can drive, and the goal is to minimize power consumption.

The input pins of the newly inserted inverters become new sinks, while the sinks driven by them are removed from the current set of sinks. When we insert an inverter, we fix neither its position nor its size. Rather we compute a set of octagons as feasible positions by taking all points with a certain maximal distance from the intersection of the sets of positions of its successors, and subtracting blocked areas and all points that are too far away from a source (cf. Figure 24). This can be computed efficiently [33].

The inverter sizes are determined only at the very end after constructing the complete tree. During the construction we work with solution candidates. A solution candidate is associated with an inverter size, an input slew, a feasible arrival time interval for the input, and a solution candidate for each successor. We prune dominated candidates, i.e. those for which another candidate with the same input slew exists whose time interval contains the time interval of the former. Thus the time intervals imply a natural order of the solution candidates with a given input slew.

Given the set of solution candidates for each successor, we compute a set of solution candidates for a newly inserted inverter as follows. For each input slew at the successors we simultaneously scan the corresponding candidate lists in the natural order and choose maximal intersections of these time intervals. For such a non-dominated candidate set we try all inverter sizes and a discrete set of input slews and check whether they generate the required input slews at the successors. If so, a new candidate is generated.

After an inverter is inserted but before its solution candidates are generated, the successors are placed at a final legal position. It may be necessary to move other objects, but with BonnPlace legalization (cf. Section 2.6) we can usually avoid moves with a large impact on timing. There are some other features which pull sinks towards sources, and which cause sinks that are ends of critical paths to be joined early in order to bound negative timing effects due to on-chip variation.

The inverter sizes are selected at the very end by choosing a solution candidate at the root. The best candidate (i.e. the best overall solution) with respect to timing (interval

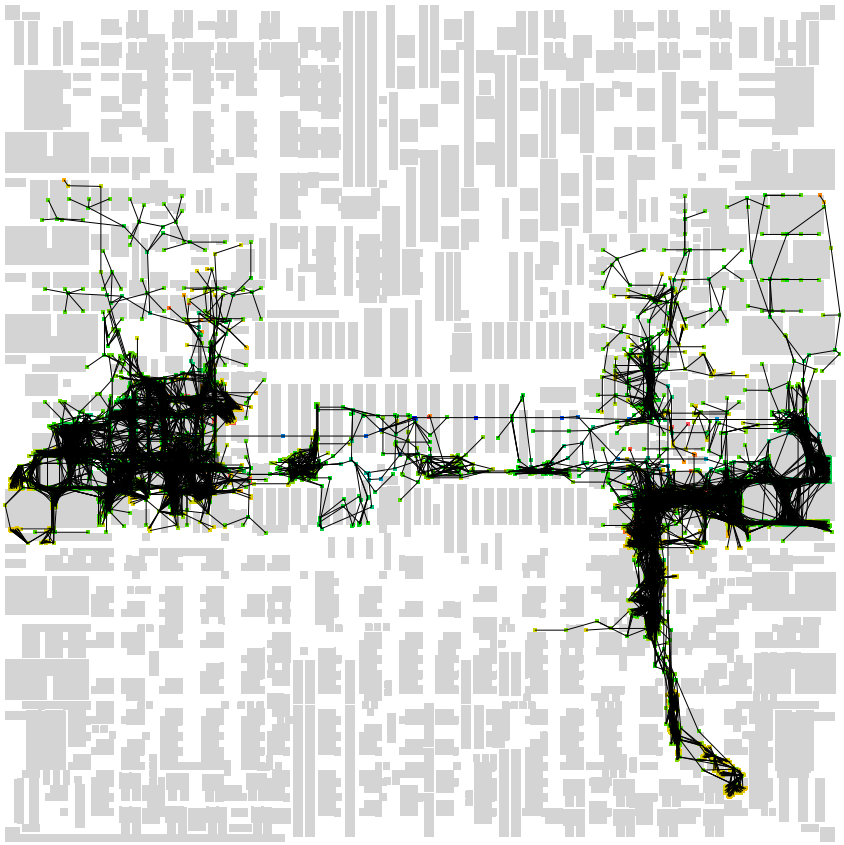

Figure 25. Gigahertz clock tree built by BonnClock based on the result of BonnCycleOpt shown in Figure 22. Colors indicate different arrival times as in Figure 23. Each net is represented by a star connecting the source to all sinks.

matching and tree latency) and power consumption is chosen. Due to discretizing slews, assuming bounded RC delays, and legalization, the timing targets may be missed by a small amount, in the order of 20 ps. But this impacts the overall timing result only if the deviation occurs in opposite directions at the ends of a critical path.

4.4. Sink clustering

The overall power consumption of the clock trees is dominated by the bottom stage, where 80–90% of the power is consumed. Therefore this stage is very important.

The basic mathematical problem that we face here can be formulated as following kind of *facility location problem*:

SINK CLUSTERING PROBLEM

Instance: A metric space (V, c),
a finite set $\mathcal{D} \subseteq V$ (terminals/clients),
demands $d : \mathcal{D} \to \mathbb{R}_+$ (input pin capacitances),
facility opening cost $f \in \mathbb{R}_+$ (cost for inserting a driver circuit),
capacity $u \in \mathbb{R}_+$ (capacity limit for a facility).

Task: Find a partition $\mathcal{D} = D_1 \dot\cup \cdots \dot\cup D_k$ and
Steiner trees T_i for D_i $(i = 1, \ldots, k)$ with

$$c(E(T_i)) + d(D_i) \leq u \quad \text{for } i = 1, \ldots, k \tag{12}$$

such that $\sum_{i=1}^{k} c(E(T_i)) + kf$ is minimum.

The term $c(E(T_i)) + d(D_i)$ in (12) is the total *load* capacitance (wire plus input pin capacitances) that must be served/driven by a facility/cell. The objective function models power consumption. In our case, V is the plane and c is the ℓ_1-metric.

The sink clustering problem is closely related to the soft-capacitated facility location problem. It contains the bin packing problem and the Steiner tree problem. The problem can therefore not be approximated arbitrary well [57]:

Theorem 12. *The* SINK CLUSTERING PROBLEM *has no* $(2-\epsilon)$*-approximation algorithm for any* $\epsilon > 0$ *for any class of metrics where the Steiner tree problem cannot be solved exactly in polynomial time.*

Proof: Assume that we have a $(2 - \epsilon)$-approximation algorithm for some $\epsilon > 0$, and let $S = \{s_1, \ldots, s_n\}$, $k \in \mathbb{R}_+$ be an instance of the decision problem "`Is there a Steiner tree for` S `with length` $\leq k$`?`". We construct an instance of the SINK CLUSTERING PROBLEM by taking S as the set of terminals, setting $d(s) = 0 \; \forall s \in S, u = k$ and $f = \frac{2k}{\epsilon}$. Then the $(2 - \epsilon)$-approximation algorithm computes a solution consisting of one facility if and only if there is a Steiner tree of length $\leq k$. This implies that the above decision problem, and hence the Steiner tree problem, can be solved in polynomial time. □

The first constant-factor approximation algorithms for this problem were given by Maßberg and Vygen [57]. One of them has a very fast running time of $O(n \log n)$ and is described now.

Let F_1 be a minimum spanning tree for $(\mathcal{D}, c)$ and $e_1, \ldots, e_{n-1}$ be the edges of F_1 in sorted order such that $c(e_1) \geq \ldots \geq c(e_{n-1})$. Let us further define a sequence of forests by $F_k := F_{k-1} \setminus \{e_{k-1}\}$ for $k = 2, \ldots, n$. Exploiting the matroid property of forests it is easy to see that each F_k, $k = 1, \ldots, n$ is a minimum weight spanning forest with exactly k connected components. By a k-Steiner forest we mean a forest F with exactly k connected components and $\mathcal{D} \subseteq V(F)$. By extending the Steiner ratio [4] from minimum spanning trees to minimum spanning forests we get:

Lemma 13. $\frac{1}{\alpha} c(F_k)$ *is a lower bound for the cost of a minimum weight* k*-Steiner forest, where* α *is the Steiner ratio.*

[4] The Steiner ratio of a metric space (V, c) is the worst-case ratio, over all terminal sets T, of the lengths of a minimum spanning tree for (T, c) and a shortest Steiner tree for T in (V, c).

We now compute a lower bound on the cost of an optimum solution. A *feasible* k-Steiner forest is a k-Steiner forest where inequality (12) holds for each of the connected components $T_1, \ldots, T_k$.

Let t' be the smallest integer such that $\frac{1}{\alpha}c(F_{t'}) + d(\mathcal{D}) \leq t' \cdot u$. By inequality (12) and Lemma 13 this is a lower bound for the number of facilities:

Lemma 14. *t' is a lower bound for the number of facilities in any feasible solution.*

Let further t'' be an integer in $\{t', \ldots, n\}$ minimizing $\frac{1}{\alpha}c(F_{t''}) + t'' \cdot f$.

Theorem 15. *$\frac{1}{\alpha}c(F_{t''}) + t'' \cdot f$ is a lower bound for the cost of an optimal solution.*

Denote $L_r := \frac{1}{\alpha}c(F_{t''})$, and $L_f := t'' \cdot f$. Then $L_r + L_f$ is a lower bound on the cost of an optimum solution, and

$$L_r + d(D) \leq L_f \frac{u}{f}. \tag{13}$$

Based on these lower bound considerations the algorithm proceeds as follows. First it computes a minimum spanning tree on (D, c). Second t'' and $F_{t''}$ are computed according to Theorem 15. If a component T of $F_{t''}$ violates (12) it must be decomposed into smaller components.

Thus overloaded components (with $c(E(T)) + d(D_i) > u$) are split. We do this in such a way that at least $\frac{u}{2}$ of the load will be removed whenever we introduce a new component. This can be done by considering a minimal overloaded subtree and applying the next-fit algorithm for bin packing. Splitting continues until no overloaded component exists. The number of new components is at most $\frac{2}{u}$ times the load of T.

Thus the total cost of the solution that we obtain is at most $c(F_{t''}) + t''f + \frac{2}{u}(c(F_{t''}) + d(D))f = \alpha L_r + L_f + \frac{2f}{u}(\alpha L_r + d(D))$. As $\frac{f}{u}L_r \leq L_f$ by (13), we get [57]:

Corollary 16. *The above algorithm computes a solution of cost at most $(2\alpha + 1)$ times the optimum in $O(n \log n)$ time, where α is the Steiner ratio.*

[57] also proved better approximation guarantees for other metric spaces, but for the rectilinear plane the above performance ratio of 4 is still the best known. However, Maßberg [58] proved stronger lower bounds. In practice, the ratio of the cost of the computed solution over a tight lower bound is typically less than 1.1. Furthermore, an exchange and merge heuristic is used to improve the clustering further as a post-optimization step. The above approximation algorithm also proves extremely fast in practice; we used it on instances with up to one million sinks.

In general, non-overlapping time windows might restrict the clustering. In addition to being computed by BonnCycleOpt, time windows occur naturally in upper levels of the clock tree even with uniform windows at the leaves. An extension of the approximation algorithm for the problem with time windows is given in [58]. By exploiting the time intervals, which are single points only for the few most critical memory elements, and by using an algorithm with provable performance guarantee the clock tree power consumption could be reduced substantially.

5. Routing

Due to the enormous instance sizes, most routers comprise at least two major parts, global and detailed routing. Global routing defines an area for each net to which the search for actual wires in detailed routing is restricted. As global routing works on a much smaller graph, we can globally optimize the most important design objectives. Moreover, global routing has another important function: decide for each placement whether a feasible routing exists and if not, give a certificate of infeasibility.

5.1. The global routing graph

The global router works on a three-dimensional grid graph which is obtained by partitioning the chip area into regions. For classical Manhattan routing this can be done by an axis-parallel grid. In any case, these regions are the vertices of the global routing graph. Adjacent regions are joined by an edge, with a capacity value indicating how many wires of unit width can join the two regions. Each routing plane has a preference direction (horizontal or vertical), and we remove edges orthogonal to this direction in the global routing graph.

For each net we consider the regions that contain at least one of its pins. These vertices of the global routing graph have to be connected by a Steiner tree. If a pin consists of shapes in more than one region, we may assign it to one of them, say the one which is closest to the center of gravity of the whole net, or by solving a group Steiner tree problem.

The quality of the global routing depends heavily on the capacities of the global routing edges. A rough estimate has to consider blockages and certain resources for nets whose pins lie in one region only. These nets are not considered in global routing. However, they may use global routing capacity. Therefore we route very short nets, which lie in one region or in two adjacent regions, first in the routing flow, i.e. before global routing. They are then viewed as blockages in global routing. Yet these nets may be rerouted later in local routing if necessary.

Routing short nets before global routing makes better capacity estimates possible, but this also requires more sophisticated algorithms than are usually used for this task. We consider a vertex-disjoint paths problem for every set of four adjacent global routing regions, illustrated in Figure 26. There is a commodity for each wiring plane, and we try to find as many paths for each commodity as possible. Each path may use the plane of its commodity in preference direction and adjacent planes in the orthogonal direction.

An upper bound on the total number of such paths can be obtained by considering each commodity independently and solving a maximum flow problem. However, this is too optimistic and too slow. Instead we compute a set of vertex-disjoint paths (i.e., a lower bound) by a very fast multicommodity flow heuristic [62]. It is essentially an augmenting path algorithm but exploits the special structure of a grid graph. For each augmenting path it requires only $O(k)$ constant-time bit pattern operations, where k is the number of edges orthogonal to the preferred wiring direction in the respective layer. In practice, k is less than three for most paths.

This very fast heuristic finds a number of vertex-disjoint paths in the region of 90% of the (weak) max-flow upper bound. For a complete chip with about one billion paths it needs 5 minutes of computing time whereas a complete max-flow computation with our implementation of the Goldberg-Tarjan algorithm would need more than a week.

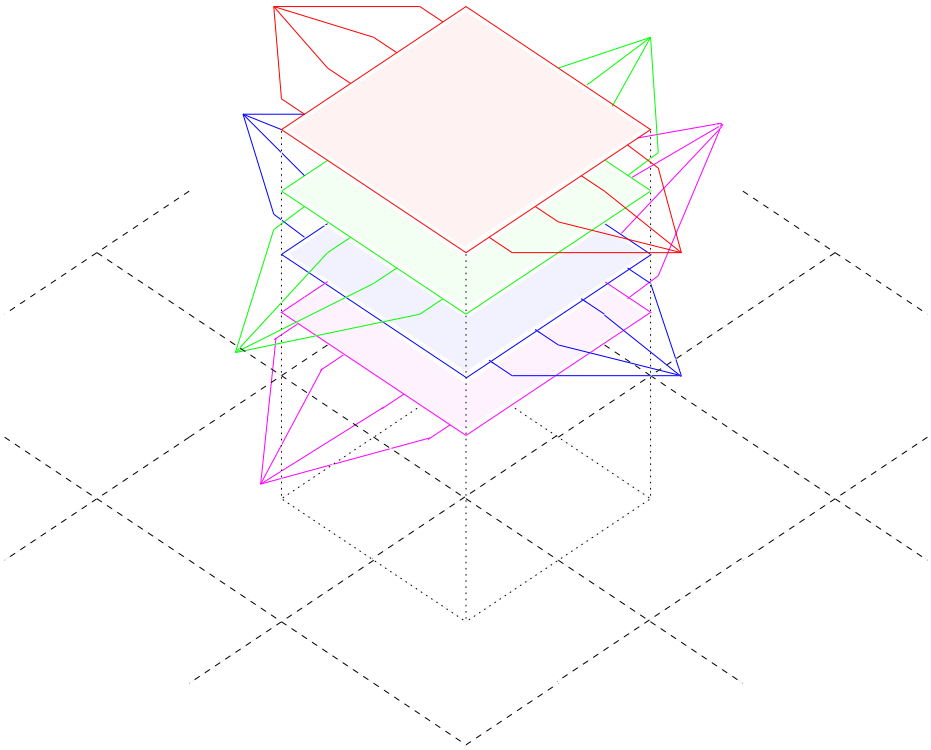

Figure 26. An instance of the vertex-disjoint paths problem for estimating global routing capacities. Dashed lines bound global routing regions. Here we show four wiring planes, each with a commodity (shown in different colors), in alternating preference directions.

Please note that this algorithm is used only for a better capacity estimation, i.e. for generating accurate input to the main global routing algorithm. However, this better capacity estimate yields much better global routing solutions and allows the detailed router to realize these solutions.

5.2. Classical global routing

In its simplest version, the global routing problem amounts to packing Steiner trees in a graph with edge capacities. A fractional relaxation of this problem can be efficiently solved by an extension of methods for the multicommodity flow problem. However, the approach does not consider today's main design objectives which are timing, signal integrity, power consumption, and manufacturing yield. Minimizing the total length of all Steiner trees is no longer important. Instead, minimizing a weighted sum of the capacitances of all Steiner trees, which is equivalent to minimizing power consumption, is an important objective. Delays on critical paths also depend on the capacitances of their nets. Wire capacitances can no longer be assumed to be proportional to the length, since coupling between neighboring wires plays an increasingly important role. Small detours of nets are often better than the densest possible packing. Spreading wires can also improve the yield.

Our global router is the first algorithm with a provable performance guarantee which takes timing, coupling, yield, and power consumption into account directly. Our algorithm extends earlier work on multicommodity flows, fractional global routing, the min-max resource sharing problem, and randomized rounding.

Let G be the global routing graph, with edge capacities $u : E(G) \to \mathbb{R}_+$ and lengths $l : E(G) \to \mathbb{R}_+$. Let $\mathcal{N}$ be the set of nets. For each $N \in \mathcal{N}$ we have a set $\mathcal{Y}_N$ of feasible Steiner trees. The set $\mathcal{Y}_N$ may contain all delay-optimal Steiner trees of N or, in many cases, it may simply contain all possible Steiner trees for N in G. Actually, we do not need to know the set $\mathcal{Y}_N$ explicitly. The only assumption which we make is that for each $N \in \mathcal{N}$ and any $\psi : E(G) \to \mathbb{R}_+$ we can find a Steiner tree $Y \in \mathcal{Y}_N$ with $\sum_{e \in E(Y)} \psi(e)$ (almost) minimum sufficiently fast. This assumption is justified since in practical instances almost all nets have less than, say, 10 pins. We can use a dynamic programming algorithm for finding an optimum Steiner tree for small nets and a fast approximation algorithm for others. With $w(N, e) \in \mathbb{R}_+$ we denote the width of net N at edge e. A straightforward integer programming formulation of the classical global routing problem is:

$$
\begin{aligned}
\min \quad & \sum_{N \in \mathcal{N}} \sum_{e \in E(G)} l(e) \sum_{Y \in \mathcal{Y}_N : e \in E(Y)} x_{N,Y} \\
\text{s.t.} \quad & \sum_{N \in \mathcal{N}} \sum_{Y \in \mathcal{Y}_N : e \in E(Y)} w(N,e) x_{N,Y} \le u(e) \qquad (e \in E(G)) \\
& \sum_{Y \in \mathcal{Y}_N} x_{N,Y} = 1 \qquad (N \in \mathcal{N}) \\
& x_{N,Y} \in \{0,1\} \qquad (N \in \mathcal{N}, Y \in \mathcal{Y}_N)
\end{aligned}
$$

Here the decision variable $x_{N,Y}$ is 1 iff the Steiner tree Y is chosen for net N. The decision whether this integer programming problem has a feasible solution is already *NP*-complete. Thus, we relax the problem by allowing $x_{N,Y} \in [0, 1]$. Raghavan and Thompson [73,74] proposed solving the LP relaxation first, and then using randomized rounding to obtain an integral solution whose maximum capacity violation can be bounded. Although the LP relaxation has exponentially many variables, it can be solved in practice for moderate instance sizes since it has only $|E(G)| + |\mathcal{N}|$ many constraints. Therefore all but $|E(G)| + |\mathcal{N}|$ variables are zero in an optimum basic solution. However, for current complex chips with millions of nets and edges, all exact algorithms for solving the LP relaxation are far too slow.

Fortunately, there exist combinatorial fully polynomial approximation schemes, i.e. algorithms that compute a feasible solution of the LP relaxation which is within a factor of $1 + \epsilon$ of the optimum, and whose running time is bounded by a polynomial in $|V(G)|$ and $\frac{1}{\epsilon}$, for any accuracy $\epsilon > 0$. If each net has exactly two pins, $\mathcal{Y}_N$ contains all possible paths connecting N, and $w \equiv 1$, the global routing problem reduces to the edge-disjoint paths problem whose fractional relaxation is the multicommodity flow problem. Shahrokhi and Matula [87] developed the first fully polynomial approximation scheme for multicommodity flows. Carden, Li and Cheng [20] first applied this approach to global routing, while Albrecht [1] applied a modification of the approximation algorithm by Garg

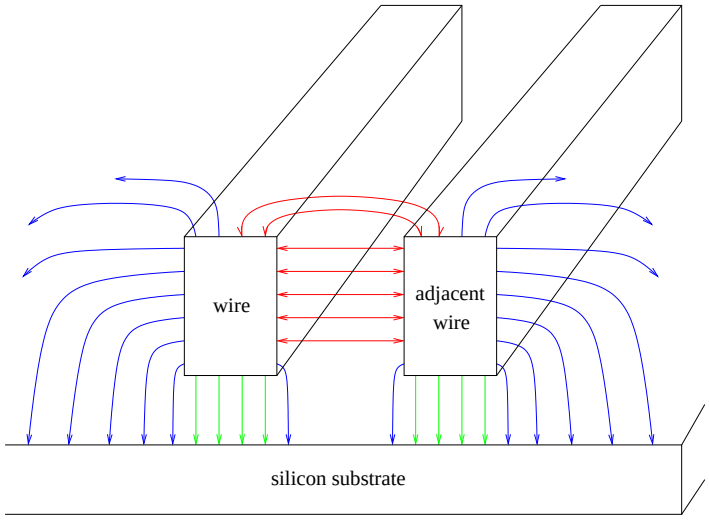

Figure 27. The capacitance of a net consists of area capacitance (green) between the undersurface of the wire and the substrate, proportional to length times width, fringing capacitance (blue) between side face of the wire and substrate, proportional to length, and coupling capacitance (red), proportional to length if adjacent wires exist. Note that the wire height is fixed within each plane, while the width varies.

and Könemann [32]. However, these approaches did not consider the above-mentioned design objectives, like timing, power, and yield.

5.3. Advanced global routing

The power consumption of a chip induced by its wires is proportional to the weighted sum of all capacitances (see Figure 27), weighted by switching activities. The coupling capacitance depends on the distance between adjacent wires. In older technologies coupling capacitances were quite small and therefore could be ignored. In deep submicron technologies coupling matters a lot.

To account for this in global routing, we assign a certain space to each edge e and each net N using this edge. We write $s(e, N) \geq 0$ for the extra space that we assign in addition to the width $w(e, N)$. The contribution of edge e to the total capacitance of N is then a convex function of $s(e, N)$.

Similarly to minimizing power consumption based on the above capacitance model, we can optimize yield by replacing capacitance by "critical area", i.e. the sensitivity of a layout to random defects [63]. Such random defects are caused by small particles that contaminate the chip during lithography. They can either disconnect a wire or connect two wires to a short.

Moreover, we can also consider timing restrictions. This can be done by excluding from the set $\mathcal{Y}_N$ all Steiner trees with large detours, or by imposing upper bounds on the weighted sums of capacitances of nets that belong to critical paths. For this purpose, we first do a static timing analysis under the assumption that every net has some expected capacitance. The set $\mathcal{Y}_N$ will contain only Steiner trees with capacitance below this expected value. We enumerate all paths which have negative slacks under this assumption. We compute the sensitivity of the nets of negative slack paths to capacitance changes,

and use these values to translate the delay bound to appropriate bounds on the weighted sum of capacitances for each path. To compute reasonable expected capacitances we can apply weighted slack balancing (cf. Section 4.2) using delay sensitivity and congestion information.

Altogether we get a family $\mathcal{M}$ of subsets of $\mathcal{N}$ with $\mathcal{N} \in \mathcal{M}$, bounds $U : \mathcal{M} \to \mathbb{R}_+$ and convex functions $g(e, N, M) : \mathbb{R}_+ \to \mathbb{R}_+$ for $N \in M \in \mathcal{M}$ and $e \in E(G)$. We also treat the objective function as a constraint (we can apply binary search to compute the optimum value approximately, but in practice we can guess an excellent bound).

With these additional assumptions and this notation we can generalize the original integer programming formulation of the global routing problem to:

GENERAL GLOBAL ROUTING PROBLEM

Instance:
- An undirected graph G with edge capacities $u : E(G) \to \mathbb{R}_+$,
- a set $\mathcal{N}$ of nets and a set $\mathcal{Y}_N$ of feasible Steiner trees for each net N,
- wire widths $w : E(G) \times \mathcal{N} \to \mathbb{R}_+$,
- A family $\mathcal{M}$ of subsets of $\mathcal{N}$ with bounds $U : \mathcal{M} \to \mathbb{R}_+$ and convex functions $g(e, N, M) : \mathbb{R}_+ \to \mathbb{R}_+$ for $N \in M \in \mathcal{M}$ and $e \in E(G)$.

Task: Find a Steiner tree $Y_N \in \mathcal{Y}_N$ and numbers $s(e, N) \geq 0$ for each $N \in \mathcal{N}$ and $e \in E(Y_N)$, such that

- $\sum_{N \in \mathcal{N}: e \in E(Y_N)} (w(e, N) + s(e, N)) \leq u(e)$ for each edge $e \in E(G)$,
- $\sum_{N \in M} \sum_{e \in E(Y_N)} g(e, N, M)(s(e, N)) \leq U(M)$ for each $M \in \mathcal{M}$.

This general formulation was proposed by [95]. Reformulating it, we look for a feasible solution (λ, x, s) to the following nonlinear optimization problem, where x is integral and $\lambda = 1$. As this is hard, we first solve the following fractional relaxation approximately and then apply randomized rounding to obtain an integral solution.

$$
\begin{aligned}
&\min \lambda \quad \text{s.t.} \\
&\qquad \sum_{Y \in \mathcal{Y}_N} x_{N,Y} = 1 && (N \in \mathcal{N}) \\
&\sum_{N \in M} \left(\sum_{Y \in \mathcal{Y}_N} x_{N,Y} \sum_{e \in E(Y)} g(e, N, M)(s(e, N)) \right) \leq \lambda\, U(M) && (M \in \mathcal{M}) \\
&\sum_{N \in \mathcal{N}} \left(\sum_{Y \in \mathcal{Y}_N : e \in E(Y)} x_{N,Y}(w(e, N) + s(e, N)) \right) \leq \lambda\, u(e) && (e \in E(G)) \\
&\qquad s(e, N) \geq 0 && (e \in E(G), N \in \mathcal{N}) \\
&\qquad x_{N,Y} \geq 0 && (N \in \mathcal{N}, Y \in \mathcal{Y}_N)
\end{aligned}
\tag{14}
$$

This can be transformed to an instance of the MIN-MAX RESOURCE SHARING PROBLEM, defined as follows. Given finite sets $\mathcal{R}$ of resources and $\mathcal{N}$ of customers, an implicitly given convex set $\mathcal{B}_N$, called block, and a convex resource consumption function $g_N : \mathcal{B}_N \to \mathbb{R}_+^{\mathcal{R}}$ for every $N \in \mathcal{N}$, the task is to find $b_N \in \mathcal{B}_N$ $(N \in \mathcal{N})$ approximately attaining $\lambda^* := \inf\{\max_{r \in \mathcal{R}} \sum_{N \in \mathcal{N}} (g_N(b_N))_r \mid b_N \in \mathcal{B}_N (N \in \mathcal{N})\}$. In the general problem formulation we have access to the sets $\mathcal{B}_N$ only via oracle functions

$f_N : \mathbb{R}_+^{\mathcal{R}} \to \mathcal{B}_N$, called block solvers, which for $N \in \mathcal{N}$ and $y \in \mathbb{R}_+^{\mathcal{R}}$ return an element $b_N \in \mathcal{B}_N$ with $y^\top g_N(b_N) \leq \sigma \inf_{b \in \mathcal{B}_N} y^\top g_N(b)$. Here $\sigma \geq 1$ is a given constant.

In our application the customers are the nets, and the resources are the elements of $E(G) \cup \mathcal{M}$. We can define

$$\mathcal{B}_N := \text{conv}(\{(\chi(Y), s) \mid Y \in \mathcal{Y}_N,\ s \in \mathbb{R}_+^{E(G)},\ s_e = 0 \text{ for } e \notin E(Y)\}),$$

where $\chi(Y) \in \{0,1\}^{E(G)}$ denote the edge-incidence vector of a Steiner tree Y. The functions g_N are then given by

$$\begin{array}{rcll} (g_N(x,s))_e & := & (x_e w(e,N) + s_e)/u(e) & (e \in E(G)) \\ (g_N(x,s))_M & := & \left(\sum_{e \in E(G): x_e > 0} x_e g(e,N,M)(s_e/x_e)\right)/U(M) & (M \in \mathcal{M}) \end{array} \tag{15}$$

for each $N \in \mathcal{N}$ and $(x,s) \in \mathcal{B}_N$.

We showed in [65] that the block solvers can be implemented by an approximation algorithm for the Steiner tree problem in weighted graphs. Then they always return an extreme point $b \in \mathcal{B}_N$, corresponding to a single Steiner tree, which we denote by Y_b.

Algorithm 3 solves the MIN-MAX RESOURCE SHARING PROBLEM, and hence (14), approximately. It is a primal-dual algorithm which takes two parameters $0 < \epsilon < 1$ and $t \in \mathbb{N}$. They control the approximation guarantee and running time.

The algorithm proceeds in t iterations where it calls the block solver for every net based on current resource prices. After each individual choice the prices are updated.

/* Initialization */
$y_r \leftarrow 1$ for $r \in \mathcal{R}$
$x_{N,b} \leftarrow 0$ for $N \in \mathcal{N}$, $b \in \mathcal{B}_N$
$X_N \leftarrow 0$ for $N \in \mathcal{N}$
/* Main Loop */
for $p := 1$ **to** t **do**:
  for $N \in \mathcal{N}$ **do**:
    while $X_N < p$ **do**:
      /* Call block solver */
      $b \leftarrow f_N(y)$.
      $a \leftarrow g_N(b)$.
      /* Update variables */
      $\xi \leftarrow \min\{p - X_N, 1/\max\{a_r \mid r \in \mathcal{R}\}\}$.
      $x_{N,b} \leftarrow x_{N,b} + \xi$ and $X_N \leftarrow X_N + \xi$.
      /* Update prices */
      for $r \in \mathcal{R}$ **do**:
        $y_r \leftarrow y_r e^{\epsilon \xi a_r}$.
/* Take Average */
$x_{N,b} \leftarrow \frac{1}{t} x_{N,b}$ for $N \in \mathcal{N}$ and $b \in \mathcal{B}_N$.

Algorithm 3: Resource Sharing Algorithm

Let $opt_N(y) := \inf_{b \in \mathcal{B}_N} y^\top g_N(b)$. The analysis of the algorithm relies on weak duality: any set of prices yields a lower bound on the optimum:

Lemma 17. *Let $y \in \mathbb{R}_+^{\mathcal{R}}$ be some cost vector with $\mathbb{1}^\top y \neq 0$. Then*

$$\frac{\sum_{N\in\mathcal{N}} opt_N(y)}{\mathbb{1}^\top y} \;\leq\; \lambda^*.$$

Proof: Let $\delta > 0$ and $(b_N \in \mathcal{B}_N)_{N\in\mathcal{N}}$ a solution with $\max_{r\in\mathcal{R}} \sum_{N\in\mathcal{N}} (g_N(b_N))_r < (1+\delta)\lambda^*$. Then

$$\frac{\sum_{N\in\mathcal{N}} opt_N(y)}{\mathbb{1}^\top y} \;\leq\; \frac{\sum_{N\in\mathcal{N}} y^\top g_N(b_N)}{\mathbb{1}^\top y} \;<\; \frac{(1+\delta)\lambda^* \mathbb{1}^\top y}{\mathbb{1}^\top y} \;=\; (1+\delta)\lambda^*.$$

□

The RESOURCE SHARING ALGORITHM yields $x_{N,b} \geq 0$ for all $b \in \mathcal{B}_N$ with $\sum_{b\in\mathcal{B}_N} x_{N,b} = 1$. Hence we have a convex combination of vectors in $\mathcal{B}_N$ for each $N \in \mathcal{N}$. To estimate the quality of the solution we prove two lemmas. Let $y^{(p,i)}$ denote y at the end of the i-th innermost iteration and k_p the total number of innermost iterations within the p-th outer iteration. We call the outer iterations phases. Let $y^{(p)}$ denote y at the end of phase p. Similar for the other variables in the algorithm.

Lemma 18. *Let (x, y) be the output of the* RESOURCE SHARING ALGORITHM. *Then*

$$\max_{r\in\mathcal{R}} \sum_{N\in\mathcal{N}} \Big(g_N\Big(\sum_{b\in\mathcal{B}_N} x_{N,b} b\Big)\Big)_r \;\leq\; \max_{r\in\mathcal{R}} \sum_{N\in\mathcal{N}} \sum_{b\in\mathcal{B}_N} x_{N,b}(g_N(b))_r \;\leq\; \frac{1}{\epsilon t}\ln\big(\mathbb{1}^\top y\big).$$

Proof: The first inequality follows from the convexity of the functions g_N. For the second inequality, note that for $r \in \mathcal{R}$:

$$\sum_{N\in\mathcal{N}} \sum_{b\in\mathcal{B}_N} x_{N,b}(g_N(b))_r \;=\; \frac{1}{t}\sum_{p=1}^{t}\sum_{i=1}^{k_p} \xi^{(p,i)}(a^{(p,i)})_r \;=\; \frac{1}{\epsilon t}\ln y_r^{(t)} \;\leq\; \frac{1}{\epsilon t}\ln\left(\mathbb{1}^\top y^{(t)}\right).$$

□

Lemma 19. *Let $\sigma \geq 1$ such that $y^\top g_N(f_N(y)) \leq \sigma opt_N(y)$ for all y. Let $\epsilon > 0$ and $\epsilon' := (e^\epsilon - 1)\sigma$. If $\epsilon'\lambda^* < 1$, then*

$$\mathbb{1}^\top y^{(t)} \;\leq\; |\mathcal{R}| e^{t\epsilon'\lambda^*/(1-\epsilon'\lambda^*)}.$$

Proof: We will consider the term $\mathbb{1}^\top y^{(p)}$ for all phases p. Initially we have $\mathbb{1}^\top y^{(0)} = |\mathcal{R}|$. We can estimate the increase of the resource prices as follows:

$$\begin{aligned}\sum_{r\in\mathcal{R}} y_r^{(p,i)} &= \sum_{r\in\mathcal{R}} y_r^{(p,i-1)} e^{\epsilon\xi^{(p,i)}(a^{(p,i)})_r} \\ &\leq \sum_{r\in\mathcal{R}} y_r^{(p,i-1)} + (e^\epsilon - 1)\sum_{r\in\mathcal{R}} y_r^{(p,i-1)}\xi^{(p,i)}(a^{(p,i)})_r,\end{aligned} \tag{16}$$

because $\xi^{(p,i)}(a^{(p,i)})_r \leq 1$ for $r \in \mathcal{R}$, and $e^x \leq 1 + \frac{e^\epsilon - 1}{\epsilon}x$ for $0 \leq x \leq \epsilon$.

Moreover,

$$\sum_{r\in\mathcal{R}} y_r^{(p,i-1)}(a^{(p,i)})_r \;\le\; \sigma\, opt_{N^{(p,i)}}(y^{(p,i-1)}). \tag{17}$$

Using (16), (17), the monotonicity of y, the fact $\sum_{i:N^{(p,i)}=N} \xi^{(p,i)} = 1$ for all N, and Lemma 17 we get

$$\begin{aligned}
\mathbb{1}^\top y^{(p)} &\le \mathbb{1}^\top y^{(p-1)} + (e^\epsilon - 1)\sigma \sum_{i=1}^{k_p} \xi^{(p,i)} opt_{N^{(p,i)}}(y^{(p,i-1)}) \\
&\le \mathbb{1}^\top y^{(p-1)} + \epsilon' \sum_{N\in\mathcal{N}} opt_N(y^{(p)}) \\
&\le \mathbb{1}^\top y^{(p-1)} + \epsilon'\lambda^* \mathbb{1}^\top y^{(p)}
\end{aligned}$$

and hence

$$\mathbb{1}^\top y^{(p)} \;\le\; \frac{\mathbb{1}^\top y^{(p-1)}}{1-\epsilon'\lambda^*}.$$

Combining this with $\mathbb{1}^\top y^{(0)} = |\mathcal{R}|$ and $1 + x \le e^x$ for $x \ge 0$ we get, if $\epsilon'\lambda^* < 1$:

$$\mathbb{1}^\top y^{(t)} \;\le\; \frac{|\mathcal{R}|}{(1-\epsilon'\lambda^*)^t} \;=\; |\mathcal{R}| \left(1 + \frac{\epsilon'\lambda^*}{1-\epsilon'\lambda^*}\right)^t \;\le\; |\mathcal{R}| e^{t\epsilon'\lambda^*/(1-\epsilon'\lambda^*)}.$$

□

Combining Lemmas 18 and 19 we get:

Theorem 20. *Let λ^* be the optimum LP value, $\lambda^* \ge \frac{1}{2}$, $\sigma\lambda^* \le \frac{5}{2}$, and $0 < \epsilon \le \frac{1}{3}$, and $t\lambda^* > \log|\mathcal{R}|$. Then the algorithm computes a feasible solution whose value differs from the optimum by at most a factor*

$$\frac{2\ln|\mathcal{R}|}{\epsilon t} + \sigma\frac{(e^\epsilon - 1)}{\epsilon(1 - \frac{5}{2}(e^\epsilon - 1))}.$$

By choosing ϵ and t appropriately, we get a $(\sigma+\omega)$-optimal solution in $O(\omega^{-2}\ln|\mathcal{R}|)$ iterations, for any $\omega > 0$. □

Although these assumptions on λ^* and σ are realistic in practice, one can also get rid of them and obtain a $(\sigma+\omega)$-optimal solution with $O(\log|\mathcal{R}|((|\mathcal{N}|+|\mathcal{R}|)\log\log|\mathcal{R}| + (|\mathcal{N}|+|\mathcal{R}|)\omega^{-2}))$ oracle calls in general [65].

Moreover, we proposed several speedup techniques and an extremely efficient parallel implementation [64,65]. This makes the approach applicable even on the largest VLSI instances. One can obtain a solution which is provably within a few percent of the optimum for an instance with millions of nets and constraints in a few hours of computing time.

The algorithm always gives a dual solution and can therefore, by Lemma 17, give a certificate of infeasibility if a given placement is not routable. We also showed how to make randomized rounding work [95,65].

This approach is quite general. It allows us to add further constraints. Here we have modeled timing, yield, and power consumption, but we may think of other constraints if further technological or design restrictions come up.

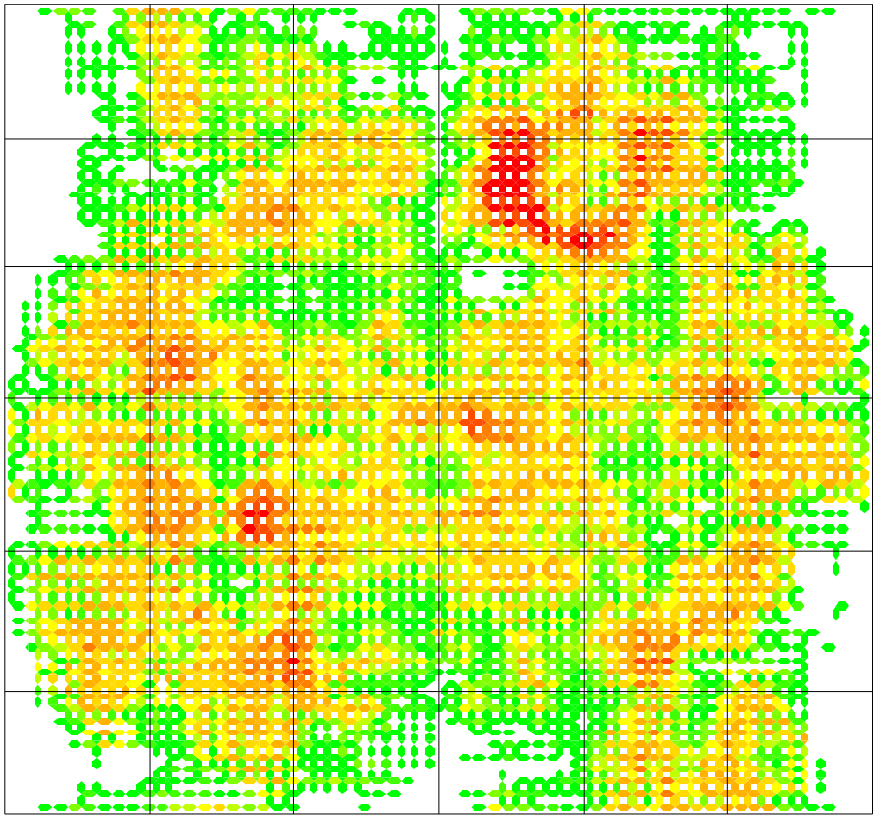

Figure 28. A typical global routing congestion map. Each edge corresponds to approximately 10x10 global routing edges (and to approximately 1 000 detailed routing channels). Red, orange, yellow, green, and white edges correspond to an average load of approximately 90–100%, 70–90%, 60–70%, 40–60%, and less than 40%.

Figure 28 shows a typical result of global routing. In the dense (red and orange) areas the main challenge is to find a feasible solution, while in other areas there is room for optimizing objectives like power or yield. Experimental results show a significant improvement over previous approaches which optimized net length and number of vias, both in terms of power consumption and expected manufacturing yield [63,64].

We conclude this section by pointing out that this problem is not restricted to VLSI design. It is in fact equivalent to routing traffic flow, with hard capacity bounds on edges (streets), without capacity bounds on vertices, with flows statically repeated over time, with bounds on weighted sums of travel times. Algorithm 3 can then be interpreted as selfish routing with taxes that depend exponentially on congestion.

5.4. Detailed routing

The task of detailed routing is to determine the exact layout of the metal realizations of the nets. Efficient data structures are used to store all metal shapes and allow fast queries. Grid-based routers define routing tracks (and minimum distances) and work with a detailed routing graph G which is an incomplete three-dimensional grid graph, i.e. $V(G) \subseteq \{x_{\min}, \ldots, x_{\max}\} \times \{y_{\min}, \ldots, y_{\max}\} \times \{1, \ldots, z_{\max}\}$ and $((x, y, z), (x', y', z')) \in E(G)$ only if $|x - x'| + |y - y'| + |z - z'| = 1$.

The z-coordinate models the different routing layers of the chip and $z_{\max}$ is typically around 10–12. We can assume without loss of generality that the x- and y-coordinates correspond to the routing tracks; typically the number of routing tracks in each plane, and hence $x_{\max} - x_{\min}$ and $y_{\max} - y_{\min}$, is in the order of magnitude of 10^5, resulting in a graph with more than 10^{11} vertices. The graph is incomplete because some parts are reserved for internal circuit structures or power supply, and some nets may have been routed earlier.

To find millions of vertex-disjoint Steiner trees in such a huge graph is very challenging. Thus we decompose this task, route the nets and even the two-point connections making up the Steiner tree for each net individually. Then the elementary algorithmic task is to determine shortest paths within the detailed routing graph (or within a part of it, as specified by global routing).

Whereas the computation of shortest paths is probably the most basic and well-studied algorithmic problem of discrete mathematics [52], the size of G and the number of shortest paths that have to be found concurrently makes the use of textbook versions of shortest path algorithms impossible. The basic algorithm for finding a shortest path connecting two given vertices in a digraph with nonnegative arc weights is Dijkstra's algorithm. Its theoretically fastest implementation, with Fibonacci heaps, runs in $O(m + n \log n)$ time, where n and m denote the number of vertices and edges, respectively [30]. For our purposes this is much too slow. Various strategies are applied to speed up Dijkstra's algorithm.

Since we are not just looking for one path but have to embed millions of disjoint trees, the information provided by global routing is most important. For each two-point connection global routing determines a corridor essentially consisting of the global routing tiles to which this net was assigned in global routing. If we find a shortest path for the two-point connection within this corridor, the capacity estimates used during global routing approximately guarantee that all desired paths can be realized disjointly. Furthermore, we get a dramatic speedup by restricting the path search to this corridor, which usually represents a very small fraction of the entire routing graph.

The second important factor speeding up our shortest path algorithm is the way in which distance information is stored. Whereas Dijkstra's algorithm labels individual vertices, we consider intervals of consecutive vertices that are similar with respect to their usability and their distance properties. Since the layers are assigned preferred routing directions, the intervals are chosen parallel to these. By the similarity of the vertices in one interval we mean that their distance properties can be encoded more efficiently than by storing numbers for each individual vertex. If e.g. the distance increases by one unit from vertex to vertex we just need to store the distance information for one vertex and the increment direction. Hetzel's version of Dijkstra's algorithm [38], generalized by [68] and [41], labels intervals instead of vertices, and its time complexity therefore depends on the

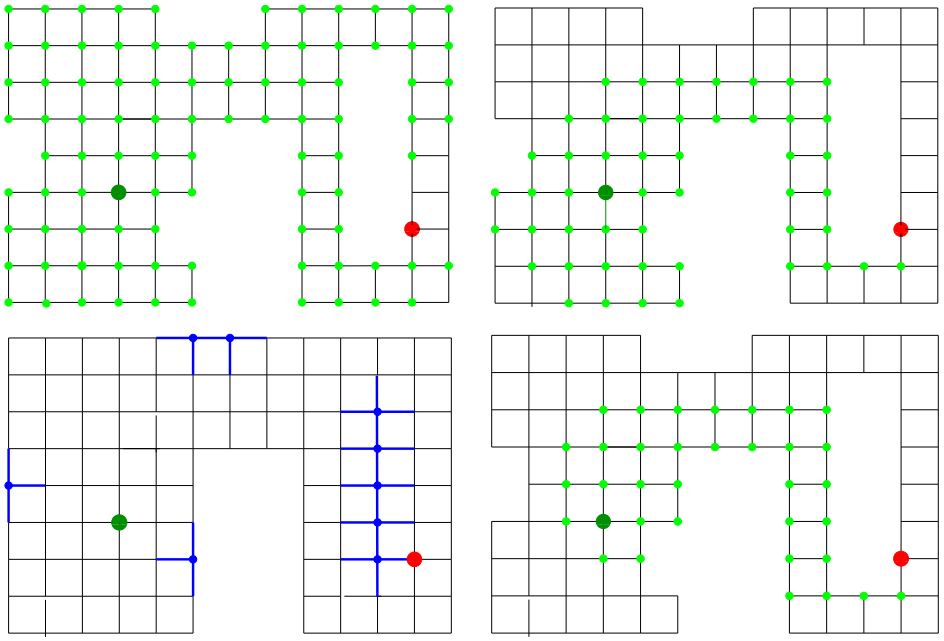

Figure 29. An instance of the shortest paths problem with unit edge weights. We look for a shortest path from the big dark green vertex in the left to the big red vertex in the right part. The figures show Dijkstra's algorithm without future costs (top left), with ℓ_1-distances as future costs (top right), and with improved future costs (bottom right). The improved future costs are based on distances in a grid graph arising by filling small holes (blue vertices and edges in the bottom left). Points labeled by Dijkstra's algorithm are marked light green. The running time is roughly proportional to the number of labeled points (93 versus 51 versus 36).

number of intervals, which is typically about 50 times smaller than the number of vertices. A sophisticated data structure for storing the intervals and answering queries very fast is the basis of this algorithm and also of its efficient shared-memory parallelization.

The last factor speeding up the path search is the use of a future cost estimate, which is a lower bound on the distance of vertices to a given target set of vertices. This is a well-known technique. Suppose we are looking for a path from s to t in G with respect to edge weights $c : E(G) \to \mathbb{R}_+$, which reflect higher costs for vias and wires orthogonal to the preferred direction and can also be used to find optimal rip-up sets. Let $l(x)$ be a lower bound on the distance from x to t (the future cost) for any vertex $x \in V$. Then we may apply Dijkstra's algorithm to the costs $c'(x, y) := c(\{x, y\}) - l(x) + l(y)$. For any s-t-path P we have $c'(P) = c(P) - l(s) + l(t)$, and hence shortest paths with respect to c' are also shortest paths with respect to c. If l is a good lower bound, i.e. close to the exact distance, and satisfies the natural condition $l(x) \leq c(\{x, y\}) + l(y)$ for all $\{x, y\} \in E(G)$, then this results in a significant speedup.

If the future cost estimate is exact, our procedure will only label intervals that contain vertices lying on shortest paths.

Clearly, improving the accuracy of the future cost estimate improves the running time of the path search and there is a tradeoff between the time needed to improve the future cost and the time saved during path search. Hetzel [38] used ℓ_1-distances as future cost estimates. In [68] we showed how to obtain and use much better estimates efficiently by

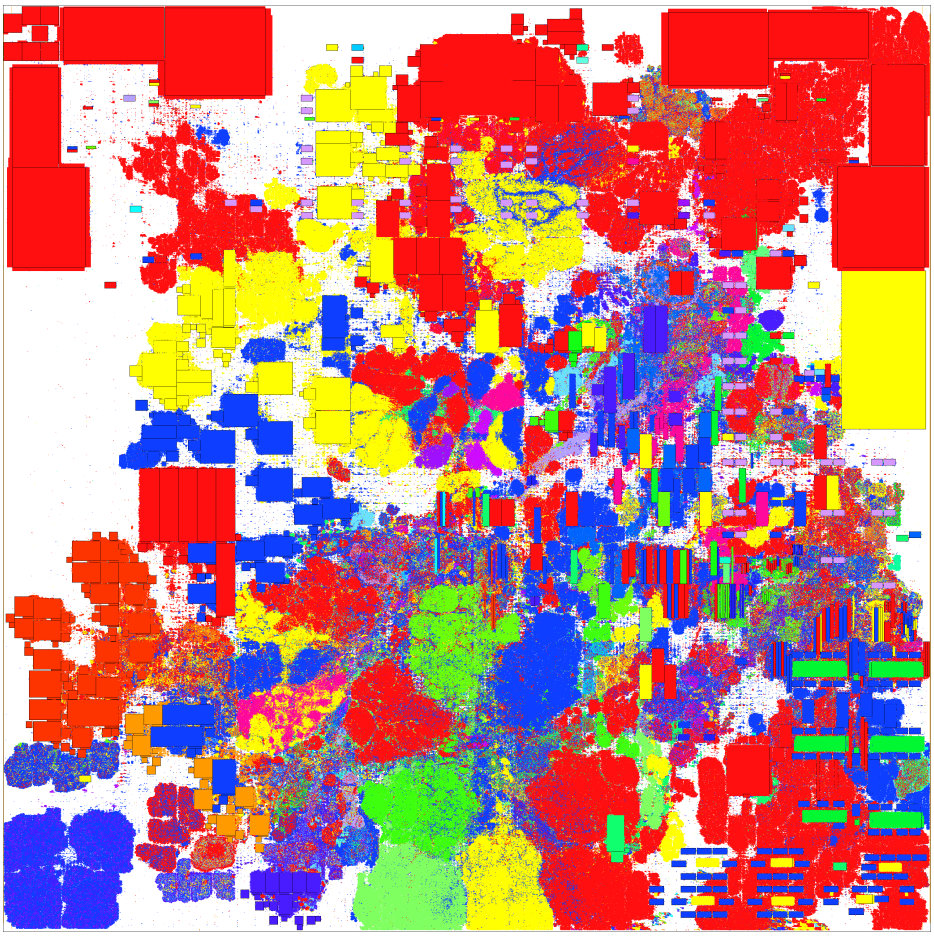

Figure 30. A system on a chip designed in 2006 with BonnTools. This 90nm design for a forthcoming IBM server has more than 5 million circuits and runs with frequencies up to 1.5 GHz. Colors reflect the structure of the underlying logic blocks.

computing distances in a condensed graph whose vertices correspond to rectangles. This leads to significant reductions of the running time as illustrated by Figure 29.

6. Conclusion

We have demonstrated that mathematics can yield better solutions for leading-edge chips. Several complete microprocessor series (cf., e.g., [28,48]) and many leading-edge ASICs (cf., e.g., [49,35]) have been designed with BonnTools. Many additional ones are in the design centers at the time of writing. Figures 30 and 31 show examples of chips that have been and are currently designed by IBM with BonnTools.

Chip design is inspiring a great deal of interesting work in mathematics. Indeed, most classical problems in combinatorial optimization, and many new ones, have been applied

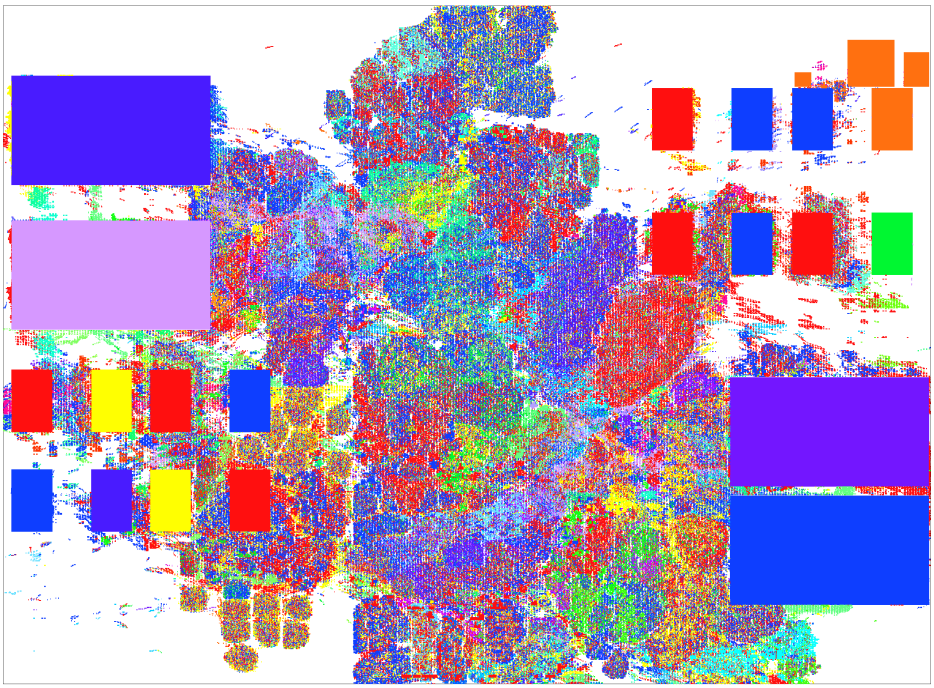

Figure 31. A vector processing unit currently designed with BonnTools. This 22nm prototype runs with a frequency of 4.7 GHz.

to chip design. Some algorithms originally developed for VLSI design automation are applied also in other contexts.

However, there remains a lot of work to do. Exponentially increasing instance sizes continue to pose challenges. Even some classical problems (e.g., logic synthesis) have no satisfactory solution yet, and future technologies continuously bring new problems. Yet we strongly believe that mathematics will continue to play a vital role in facing these challenges.

Acknowledgments

We thank all current and former members of our team in Bonn. Moreover, we thank our cooperation partners, in particular at IBM. Last but not least we thank Vašek Chvátal for inviting us to give lectures on the topics discussed in this paper at the 45th Session of the Séminaire de Mathématiques Supérieures at Montréal.

References

[1] Albrecht, C.: Global routing by new approximation algorithms for multicommodity flow. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 20 (2001), 622–632

[2] Albrecht, C.: Zwei kombinatorische Optimierungsprobleme im VLSI-Design. Ph.D. thesis, University of Bonn, 2001

[3] Albrecht, C., Korte, B., Schietke, J., and Vygen, J.: Cycle time and slack optimization for VLSI-chips. Proceedings of the IEEE International Conference on Computer-Aided Design (1999), 232–238

[4] Albrecht, C., Korte, B., Schietke, J., and Vygen, J.: Maximum mean weight cycle in a digraph and minimizing cycle time of a logic chip. Discrete Applied Mathematics 123 (2002), 103–127

[5] Bartoschek, C., Held, S., Rautenbach, D., and Vygen, J.: Efficient generation of short and fast repeater tree topologies. Proceedings of the International Symposium on Physical Design (2006), 120–127

[6] Bartoschek, C., Held, S., Rautenbach, D., and Vygen, J.: Fast buffering for optimizing worst slack and resource consumption in repeater trees. Proceedings of the International Symposium on Physical Design (2009), 43–50

[7] Bartoschek, C., Held, S., Maßberg, J., Rautenbach, D., and Vygen, J.: The repeater tree construction problem. Technical Report No. 09998, Research Institute for Discrete Mathematics, University of Bonn, 2009

[8] Boyd, S., Kim, S.-J., Patil, D., and Horowitz, M.: Digital circuit optimization via geometric programming. Operations Research 53 (6), 2005, 899–932

[9] Brenner, U.: A faster polynomial algorithm for the unbalanced Hitchcock transportation problem. Operations Research Letters 36 (2008), 408–413

[10] Brenner, U., Pauli, A., and Vygen, J.: Almost optimal placement legalization by minimum cost flow and dynamic programming. Proceedings of the International Symposium on Physical Design (2004), 2–9

[11] Brenner, U., and Rohe, A.: An effective congestion driven placement framework. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 22 (2003), 387–394

[12] Brenner, U., and Struzyna, M.: Faster and better global placement by a new transportation algorithm Proceedings of the 42nd IEEE/ACM Design Automation Conference (2005), 591–596

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

[14] Brenner, U., and Vygen, J.: Faster optimal single-row placement with fixed ordering. Design, Automation and Test in Europe, Proceedings, IEEE 2000, 117–121

[15] Brenner, U., and Vygen, J.: Worst-case ratios of networks in the rectilinear plane. Networks 38 (2001), 126–139

[16] Brenner, U., and Vygen, J.: Legalizing a placement with minimum total movement. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 23 (2004), 1597–1613

[17] Brenner, U.: Theory and Practice of VLSI Placement. Dissertation, UniversitÃd't Bonn, 2005

[18] Brenner, U., and Vygen, J.: Analytical methods in VLSI placement. In: Handbook of Algorithms for VLSI Physical Design Automation (Alpert, C.J., Mehta, D.P., Sapatnekar, S.S., Eds.), Taylor and Francis, Boca Raton 2009, 327–346

[19] Brent, R.: On the addition of binary numbers. IEEE Transactions on Computers 19 (1970), 758–759

[20] Carden IV, R.C., Li, J., and Cheng, C.-K.: A global router with a theoretical bound on the optimum solution. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15 (1996), 208–216

[21] Chao, T.-H., Hsu, Y.-C., Ho, J.M.: Zero skew clock net routing. Proceedings of the 29th ACM/IEEE Design Automation Conference (1992), 518–523

[22] Chen, C.-P., Chu, C.C.N., and Wong, D.F.: Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 18 (1999), 1014–1025

[23] Cheney, W., and Goldstein, A.: Proximity maps for convex sets. Proceedings of the AMS 10 (1959), 448–450

[24] Chu, C.C.N., and Wong, D.F: Greedy wire-sizing is linear time. Proceedings of the International Symposium on Physical Design (1998), 39–44

[25] Cong, J., and He, L.: Local-refinement-based optimization with application to device and interconnect sizing. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 18 (1999), 406–420.

[26] Cook, W.J., Lovász, L., and Vygen, J. (Eds.): Research Trends in Combinatorial Optimization. Springer, Berlin 2009

[27] Elmore, W.C.: The transient response of damped linear networks with particular regard to wide-band amplifiers. Journal of Applied Physics 19(1) (1948), 55–63.

[28] Fassnacht, U. and Schietke, J.: Timing analysis and optimization of a high-performance CMOS processor chipset. Design, Automation and Test in Europe, Proceedings, IEEE 1998, 325–331

[29] Fishburn, J., and Dunlop, A.: TILOS: A posynomial programming approach to transistor sizing. IEEE International Conference on Computer-Aided Design (1985). Digest of Technical Papers, 326–328.

[30] Fredman, M.L., and Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization problems. Journal of the ACM 34 (1987), 596–615

[31] Garey, M.R., and Johnson, D.S.: The rectilinear Steiner tree problem is *NP*-complete. SIAM Journal on Applied Mathematics 32 (1977), 826–834

[32] Garg, N., and Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (1998), 300–309

[33] Gester, M.: Voronoi diagrams of octagons in the maximum metric [in German]. Diploma thesis, University of Bonn, 2009.

[34] Held, S.: Algorithms for potential balancing problems and applications in VLSI design [in German]. Diploma thesis, Research Institute for Discrete Mathematics, University of Bonn, 2001

[35] Held, S., Korte, B., Maßberg, J., Ringe, M., and Vygen, J.: Clock scheduling and clocktree construction for high performance ASICs. Proceedings of the IEEE International Conference on Computer-Aided Design (2003), 232–239

[36] Held, S.: Timing closure in chip design. Ph.D. thesis, Research Institute for Discrete Mathematics, University of Bonn, 2008

[37] Held, S.: Gate sizing for large cell-based designs. Design, Automation and Test in Europe, Proceedings, IEEE 2009, 827–832

[38] Hetzel, A.: A sequential detailed router for huge grid graphs. Design, Automation and Test in Europe, Proceedings, IEEE 1998, 332–338

[39] Hougardy, S.: Personal communication, 2010.

[40] Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proceedings of the Institute of Radio Engineers 40 (1952), 1098–1102

[41] Humpola, J.: Schneller Algorithmus für kürzeste Wege in irregulären Gittergraphen. Diploma thesis, Research Institute for Discrete Mathematics, University of Bonn, 2009.

[42] Hwang, F.K.: On Steiner minimal trees with rectilinear distance. SIAM Journal on Applied Mathematics 30 (1976), 104–114

[43] Jerrum, M.: Complementary partial orders and rectangle packing. Technical Report, Deptartment of Computer Science, University of Edinburgh, 1985

[44] Kahng, A.B., Tucker, P., and Zelikovsky, A.: Optimization of linear placements for wire length minimization with free sites. Proceedings of the Asia and South Pacific Design Automation Conference, 1999, 241–244

[45] Karp, R.M.: A characterization of the minimum mean cycle in a digraph. Discrete Mathematics 23 (1978), 309–311

[46] Khrapchenko, V.M.: Asymptotic estimation of addition time of a parallel adder. Systems Theory Research 19 (1970), 105–122

[47] Kleinhans, J.M., Sigl, G., Johannes, F.M., and Antreich, K.J.: GORDIAN: VLSI placement by quadratic programming and slicing optimization. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 10 (1991), 356–365

[48] Koehl, J., Baur, U., Ludwig, T., Kick, B., and Pflueger, T.: A flat, timing-driven design system for a high-performance CMOS processor chipset. Design, Automation, and Test in Europe, Proceedings, IEEE 1998, 312–320

[49] Koehl, J., Lackey, D.E., and Doerre, G.W.: IBM's 50 million gate ASICs. Proceedings of the Asia and South Pacific Design Automation Conference, IEEE 2003, 628–634

[50] Korte, B., Lovász, L., Prömel, H.J., and Schrijver, A. (Eds.): Paths, Flows, and VLSI-Layout. Springer, Berlin 1990

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

[52] Korte, B., and Vygen, J.: Combinatorial Optimization: Theory and Algorithms. Fourth edition. Springer, Berlin 2008
[53] Korte, B., and Vygen, J.: Combinatorial problems in chip design. In: Building Bridges Between Mathematics and Computer Science (M. Grötschel, G.O.H. Katona, eds.), Springer, Berlin 2008, 333–368
[54] Kraft, L.G.: "A device for quantizing grouping and coding amplitude modulated pulses", Master thesis, EE Dept., MIT, Cambridge 1949
[55] Langkau, K.: Gate-Sizing im VLSI-Design [in German]. Diploma thesis, Research Institute for Discrete Mathematics, University of Bonn, 2000
[56] Liu, J., Zhou, S., Zhou, H., and Cheng, C.-K.: An algorithmic approach for generic parallel adders. Proceedings of the International Conference on Computer Aided Design (2003), 734–740
[57] Maßberg, J., and Vygen, J.: Approximation algorithms for a facility location problem with service capacities. ACM Transactions on Algorithms 4 (2008), Article 50
[58] Maßberg, J.: Facility location and clock tree synthesis. Ph.D. thesis, Research Institute for Discrete Mathematics, University of Bonn, 2009
[59] Megiddo, N.: Applying parallel computation algorithms in the design of serial algorithms: Journal of the ACM 30(4) (1983), 852–865
[60] Minoux, M.: Mathematical Programming: Theory and Algorithms. Wiley, Chichester 1986
[61] Moore, G.E.: Cramming more components onto integrated circuits. Electronics 38 (8) (1965), 114–117
[62] Müller, D.: Determining routing capacities in global routing of VLSI chips [in German]. Diploma thesis, Research Institute for Discrete Mathematics, University of Bonn, 2002
[63] Müller, D.: Optimizing yield in global routing. Proceedings of the IEEE International Conference on Computer-Aided Design (2006), 480–486
[64] Müller, D.: Fast resource sharing in VLSI design. Ph.D. thesis, Research Institute for Discrete Mathematics, University of Bonn, 2009
[65] Müller, D., Radke, K., and Vygen, J.: Faster min-max resource sharing in theory and practice. Technical Report No. 101013, Research Institute for Discrete Mathematics, University of Bonn, 2010
[66] Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y.: VLSI module placement based on rectangle-packing by the sequence-pair. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 15 (1996), 1518–1524
[67] Orlin, J.B.: A faster strongly polynomial minimum cost flow algorithm. Operations Research 41 (1993), 338–350
[68] Peyer, S., Rautenbach, D., and Vygen, J.: A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing. Journal of Discrete Algorithms 7 (2009), 377–390
[69] Philips, S., and Dessouky, M.: Solving the project time/cost tradeoff problem using the minimal cut concept. Management Science 24 (1977), 393–400
[70] Polyak, B.T.: A general method of solving extremum problems/ Doklady Akademii Nauk SSSR 174/1 (1967), 33-36 (in Russian). Translated in Soviet Mathematics Doklady 8/3 (1967), 593–597.
[71] Polyak, B.T.: Minimization of unsmooth functionals. USSR Computational Mathematics and Mathematical Physics 9 (1969), 14–29.
[72] Queyranne, M.: Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems. Operations Research Letters 4 (1986), 231–234
[73] Raghavan, P., and Thompson, C.D.: Randomized rounding: a technique for provably good algorithms and algorithmic proofs. Combinatorica 7 (1987), 365–374
[74] Raghavan, P., and Thompson, C.D.: Multiterminal global routing: a deterministic approximation. Algorithmica 6 (1991), 73–82
[75] Rautenbach, D.: Rectilinear spanning trees versus bounding boxes. The Electronic Journal of Combinatorics 11 (2004), #N12
[76] Rautenbach, D., and Szegedy, C.: A subgradient method using alternating projections. Technical Report No. 04940, Research Institute for Discrete Mathematics, University of Bonn, 2004
[77] Rautenbach, D. and Szegedy, C.: A class of problems for which cyclic relaxation converges linearly. Computational Optimization and Applications 41 (2008), 53–60
[78] Rautenbach, D., Szegedy, C., and Werber, J.: Asymptotically optimal Boolean circuits for functions of the form $g_{n-1}(g_{n-2}(...g_3(g_2(\ g_1(x_1,x_2),x_3),x_4)...,x_{n-1}),x_n)$. Technical Report No. 03931, Research Institute for Discrete Mathematics, University of Bonn, 2003
[79] Rautenbach, D., Szegedy, C., and Werber, J.: Delay optimization of linear depth Boolean circuits with prescribed input arrival times. Journal of Discrete Algorithms 4 (2006), 526–537

[80] Rautenbach, D., Szegedy, C., and Werber, J.: The delay of circuits whose inputs have specified arrival times. Discrete Applied Mathematics 155 (2007), 1233–1243.

[81] Rautenbach, D., Szegedy, C., and Werber, J.: On the cost of optimal alphabetic code trees with unequal letter costs. European Journal of Combinatorics 29 (2008), 386–394

[82] Rautenbach, D., Szegedy, C., and Werber, J.: Timing optimization by restructuring long combinatorial paths. Proceedings of the IEEE International Conference on Computer-Aided Design (2007), to appear.

[83] Sapatnekar, S.: Timing. Kluwer Academic Publishers, Boston, MA, 2004

[84] Schmedding, R.: Time-Cost-Tradeoff-Probleme und eine Anwendung in der Timing-Optimierung im VLSI-Design. Diploma thesis, Research Institute for Discrete Mathematics, University of Bonn, 2007

[85] Schneider, H., and Schneider, M.H.: Max-balancing weighted directed graphs and matrix scaling. Mathematics of Operations Research 16 (1991), 208–222

[86] Schneider, J.: Macro placement in VLSI design. Diploma thesis, Research Institute for Discrete Mathematics, University of Bonn, 2009

[87] Shahrokhi, F., and Matula, D.W.: The maximum concurrent flow problem. Journal of the ACM 37 (1990), 318–334

[88] Skutella, M.: Approximation algorithms for the discrete time-cost tradeoff problem. Mathematics of Operations Research 23 (1998), 909–929.

[89] Struzyna, M.: Flow-based partitioning and fast global placement in chip design. Ph.D. thesis, Research Institute for Discrete Mathematics, University of Bonn, 2010

[90] Suhl, U.: Row placement in VLSI design: the clumping algorithm and a generalization. Diploma thesis, Research Institute for Discrete Mathematics, University of Bonn, 2010

[91] Tennakoon, H., Sechen, C.: Gate sizing using Lagrangian relaxation combined with a fast gradient-based pre-processing step. Proceedings of the IEEE International Conference on Computer-Aided Design (2002), 395–402

[92] Vygen, J.: Algorithms for large-scale flat placement. Proceedings of the 34th IEEE/ACM Design Automation Conference (1997), 746–751

[93] Vygen, J.: Algorithms for detailed placement of standard cells. Design, Automation and Test in Europe, Proceedings, IEEE 1998, 321–324

[94] Vygen, J.: On dual minimum cost flow algorithms. Mathematical Methods of Operations Research 56 (2002), 101–126

[95] Vygen, J.: Near-optimum global routing with coupling, delay bounds, and power consumption. In: Integer Programming and Combinatorial Optimization; Proceedings of the 10th International IPCO Conference; LNCS 3064 (G. Nemhauser, D. Bienstock, eds.), Springer, Berlin 2004, 308–324

[96] Vygen, J.: Geometric quadrisection in linear time, with application to VLSI placement. Discrete Optimization 2 (2005), 362–390

[97] Vygen, J.: Slack in static timing analysis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25 (2006), 1876–1885

[98] Vygen, J.: New theoretical results on quadratic placement. Integration, the VLSI Journal 40 (2007), 305–314

[99] Yeh, W.-C., and Jen, C.-W.: Generalized earliest-first fast addition algorithm. IEEE Transactions on Computers 52 (2003), 1233–1242.

[100] Young, N.E., Tarjan, R.E., and Orlin, J.B.: Faster parametric shortest path and minimum balance algorithms. Networks 21 (1991), 205–221