Global Timing Optimization in Chip Design

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Vorgelegt von

Siad Daboul

aus

Lippstadt

Bonn, Januar 2021

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

Erstgutachter:	Herr Professor Dr. Stephan Held
Zweitgutachter:	Herr Professor Dr. Jens Vygen
Tag der Promotion:	19. März 2021
Erscheinungsjahr:	2021

Acknowledgements

Throughout the writing of this work, I have received a large amount of support and assistance by many people. First and foremost, I would like to thank my supervisors Professor Dr. Stephan Held and Professor Dr. Jens Vygen for their support, feedback, ideas and collaboration on many aspects of this thesis.

The working conditions at the Institute for Discrete Mathematics are excellent. I want to thank especially Prof. Dr. Bernhard Korte for creating this environment and for establishing the cooperation with IBM more than 30 years ago. This partnership has provided a lot of benefits for both sides and will hopefully do so for many years to come.

I am grateful to my colleagues Tilmann Bihler, Dr. Ulrich Brenner, Josefine Foos, Dr. Anna Hermann, Dr. Dirk Müller, Bento Natura, Stefan Rabenstein, Benjamin Rockel, Dr. Daniel Rotter, Pietro Saccardi, Dr. Rudolf Scheifele, Dr. Ulrike Schorr and Dr. Jannik Silvanus. Discussing some difficult problem or just meeting up for a coffee was always a pleasure. I want to thank Benjamin Rockel and Dr. Daniel Rotter in particular for proofreading a preliminary version of this work and for providing valuable comments and remarks.

Furthermore, I would like to express my gratitude to several current and former IBM employees for providing help and sharing their knowledge. In particular William Dougherty, Harald Folberth, Michael Kazda, Lakshmi Reddy, Gregory Schaeffer, Friedrich Schroeder, Alex Suess and Cindy Washburn.

Lastly, I thank my family for their encouragement and support in the time of working on this thesis.

Contents

Chapter 1

Introduction

During the design of a computer chip, numerous objectives compete against each other. First and foremost, we have the specified clock frequency. The chip only works correctly if implicit interconnect delay bounds are met. Other objectives include the minimization of wiring space usage, power consumption and placement space usage. Individual objectives are often contrary, for example fast realizations of combinatorial cells are available with the drawback of a substantial power consumption. Solving these tradeoff problems is the main topic of this thesis.

On a fundamental level, a computer chip consists of transistors. These tiny devices allow a current to flow depending on the voltage at their gate contact. By adjusting transistor materials in the manufacturing process, switching voltages (V_t) of combinatorial cells can be greatly improved at the cost of increased power consumption. The individual transistors only implement a simple Boolean function, but when connected to each other in the right way they realize a complex circuit. Modern chips have many wiring layers with various options to select wire widths and spacings. These choices heavily influence resource consumption and signal delays. Figure 1.1 shows the metal stack of a chip in the 10nm technology node. Higher layers have a significantly larger cross-section for individual wiring segments. This corresponds to a lower electrical resistance and, thus, reduced signal delays. However, due to large spacings only few wires can be packed on upper layers. Hence, individual nets compete for limited routing resources. The main goal of the design process is to reach the specified timing constraints while not overusing any of the available resources.

The availability of extremely thick wiring layers and fast transistors allows most signal constraints to be met at the cost of high resource usage. We can also see timing constraints as an available resource which is overused if the constraints are violated. Therefore, the design process is inherently a *resource-sharing* problem, in which the available resources are distributed to meet timing constraints. An essential objective of this problem is to find a

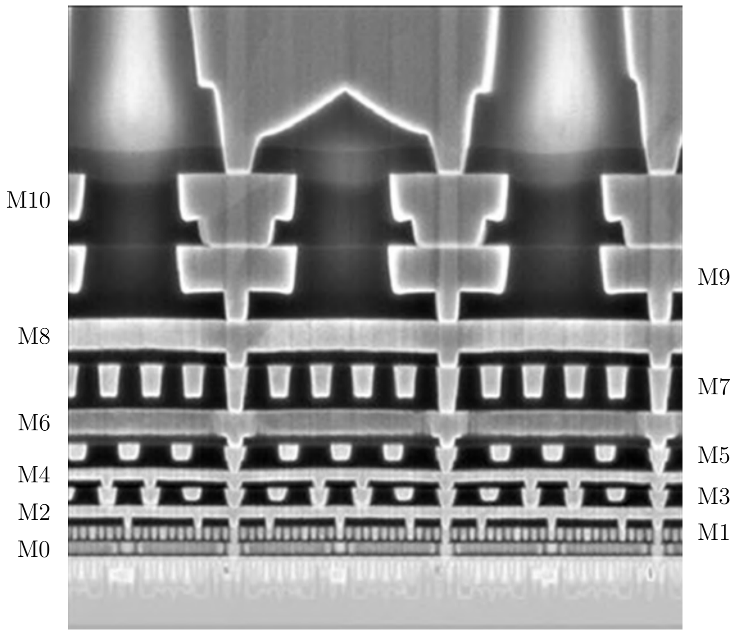


Figure 1.1: A cross section of the metal stack in the Intel 10nm technology node. Annotations on the left and right side name the routing layers. Layers ending in an odd digit are vertical, i.e., the routing preference direction is orthogonal to the cross section. Layers ending in even digits are horizontal. One can see that the individual layers have highly asymmetric routing space consumption characteristics. Image from: Intel, [Aut+17].

solution that does not overuse any resource. If such a solution does not exist the maximum overuse should be minimized. In contemporary industrial design flows, this problem has often been solved heuristically, for example by starting with a solution of low resource usage and to rely on postoptimization to resolve timing problems [Li+12].

For the case of interconnect optimization, an example of this approach is depicted in Figure 1.2. Here, we show the resource usage, as indicated by the wiring length (WL) and the routing space consumption (visualized by the edge colors), compared to the achieved timing. As timing constraints usually cannot be satisfied in early design phases, the infeasibility is given by the worst slack (WS). The worst slack measures the maximum time by which a signal misses its required time of arrival. As it depends only on a single failing signal path it can be easily affected by perturbations of the netlist. Therefore, we also compute the total negative slack (TNS), which does not only measure the worst path but sums the slacks of a larger set of timing endpoints. Negative values of WS and TNS indicate that signals arrive too late. The timing-unaware solution, which only aims to minimize the routing resource usage, obtains a good wiring length but significantly violates the timing constraints. Therefore, heuristic postoptimizations lead to a suboptimal

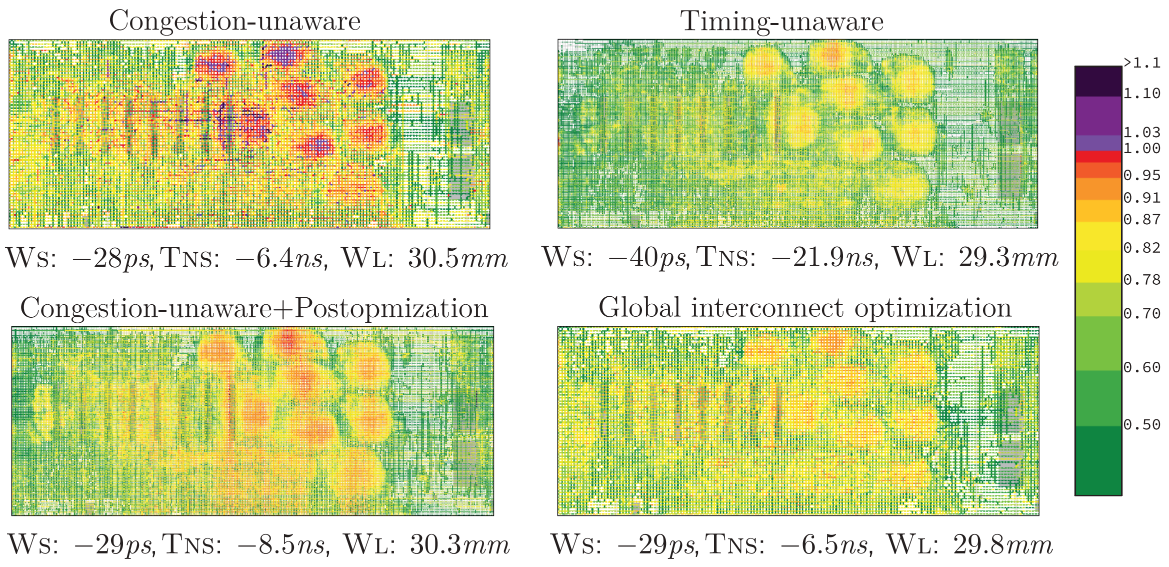


Figure 1.2: An example of four different design flows. The respective images highlight the routing resource usage of the computed interconnect. A red or purple color indicates infeasible resource usage. If resource capacity constraints are ignored, the computed result is not feasible, as can be seen in the top left picture. If timing constraints are ignored, the chip has a poor design frequency as indicated in the top right. A classic approach postoptimizes this solution, resulting in the solution shown in the bottom left. A new approach, combining timing constraints and resource usage minimization is shown in the bottom right.

solution at significantly increased resource usage and a large wiring length.

In this thesis, we aim at solving the interconnect optimization problem comprehensively. By balancing global timing, routing, placement, and power constraints in a global model, we obtain solutions which outperform the classic approach in both resource usage and timing quality.

An important special case of the *resource sharing* problem arises when we only have two resources: timing and cost. Consider the problem of selecting a solution in a discrete set of delay/cost alternatives for the vertices of an acyclic directed graph. If both delay and cost functions are separable, this is the well-known discrete *time-cost tradeoff* problem. Due to its numerous applications, it was already considered in the context of planning and scheduling more than 60 years ago [KW59]. However, on the positive side not much is known. In practice this problem is mostly solved by heuristics.

Our main contributions are as follows. For the time-cost tradeoff problem in chip design, we present a new implementation of a primal-dual V_t optimization algorithm. Instead of requiring separable delay constraints as in previous approaches, it only has mild assumptions on the delay model. The new assumptions mostly hold in practice, even when computing non-linear delays with an industrial sign-off timer. This approach allows us to achieve leakage reductions of up to 8% on netlists that were pre-optimized by one of the most successful algorithms for gate sizing and V_t assignment [RSR16a]. Our algorithm simultaneously computes an a posteriori lower bound which shows that we solve some of the instances almost optimally. After global routing the reduction grows up to 34% without changing any footprints.

The V_t assignment problem with separable delays directly corresponds to the discrete time-cost tradeoff problem in directed graphs. If d is the maximum number of vertices in any path, our practical algorithm yields a d approximation. Previously, Svensson [Sve12] showed that for general instances with unbounded values of d no constant factor approximation exists if we assume $\mathrm{P} \neq \mathrm{NP}$ and the Unique Games Conjecture holds. The latter conjecture was proposed by Khot in 2002 [Kho02]. Since then, many strong inapproximability results have been found under this conjecture. For example, it implies that vertex cover is NP hard to approximate up to a factor $2 - \epsilon$ [KR08]. Therefore, the simple greedy algorithm is essentially best possible.

For the discrete time-cost tradeoff problem, we devise an improved algorithm with a guarantee of $\frac{d}{2}$. We achieve this by reformulating the problem to a vertex cover problem in d-partite hypergraphs. For this more general problem, the approximation ratio of our new algorithm is slightly better than $\frac{d}{2}$, which is asymptotically best possible under the Unique Games Conjecture and $\mathrm{P} \neq \mathrm{NP}$. We also study the inapproximability of the time-cost tradeoff problem and show that no better approximation ratio than $\frac{d+2}{4}$ is possible, again assuming the Unique Games Conjecture and $\mathrm{P} \neq \mathrm{NP}$. Therefore, we settle the approximability of this problem up to a factor of less than 2.

We then focus on the gate sizing problem. It is similar to the time-cost tradeoff problem but optimizes a more involved timing function, which is not linear but only posynomial. Schorr [Sch15] presented a resource-sharing formulation for the gate sizing problem in her dissertation. We give a new runtime analysis for the resource sharing algorithm applied to gatesizing, resolving small inaccuracies in the previous proof. Furthermore, Schorr [Sch15] compared the subgradient method to the resource sharing algorithm. We extend her analysis for the case where an additional power constraint is added to the problem. Our findings indicate that a power constraint significantly improves the subgradient method. However, even with the new model the resource sharing algorithm converges faster.

Besides, we present a practical implementation of the resource sharing algorithm for gate sizing with heuristic modifications. We compare our new implementation with the state-of-the-art algorithm of Reimann et al. [RSR16a]. On all designs our algorithm obtains similar or better power savings while drastically reducing runtime. On the larger testcases the speedup is between a factor of 6 and 10.

Finally, we consider the buffering problem. In this problem the interconnect for all nets should be computed. Simultaneously, repeating gates need to be inserted to strengthen electrical signals. We build on the resource sharing formulation for this problem given by Rotter [Rot17]. On the theoretic side, we modify the model by using a path based formulation for timing proposed by Hähnle [Häh15]. This simplifies the implementation and does not lead to worse results as our experiments show. We also point out that the problem formulation of Rotter relied on a rough subdivision of the chip to select buffer positions. In practice this could lead to high movements in the subsequent legalization step. We change the problem formulation to account for this and present a practical algorithm to obtain solutions which can be easily legalized.

Rotter gave a first implementation of his algorithm, but acknowledged that it still had several shortcomings which prevented its application in practice [Rot17]. The new implementation is now robust enough to be used in an industrial environment. It resolves all major issues of the previous implementation and subsequently outperforms a state-of-the-art design flow in almost all metrics, including netlength, power, congestion and timing. We also implement speedups that reduce the runtime by up to 70%.

Our ambition is to solve the resource sharing problems globally instead of relying on local optimization. All three described algorithms are now integrated in the design flow of our industrial partner IBM where they replaced their previous counterparts. Therefore, the new optimization flow now uses a uniform objective which encompasses all important design aspects.

This thesis is structured as follows. In Chapter 2 we give an overview of the chip design process and describe how the timing of a chip is modeled. The design process step in which Steiner trees are packed in a grid graph to find an interconnect solution obeying routing capacities is called *Global Routing*. As this step is of major importance for obtaining an algorithm that optimizes both timing and resource usage, we dedicate a separate chapter to it: Chapter 3.

In Chapter 4 and 5 we show how to solve the time-cost tradeoff problem in theory and practice. The following Chapter 6 discusses the gate sizing problem. The final Chapter 7 is dedicated to BonnRouteBuffer.

Chapter 2

Preliminaries

The mathematical problems discussed in this thesis arise in the context of chip design. We will use this chapter to familiarize the reader with fundamental concepts of the aforementioned design process. To this end, we assume fundamental graph theoretic notions as introduced in the book of Korte and Vygen [KV11]. For a comprehensive overview on the topic, we also recommend the habilitation thesis by Vygen [Vyg01] or the dissertation of Held [Hel08].

In a computer chip, information is transmitted by electrical signals. These can be separated into data and clock signals, of which this thesis only considers data signals. Unlike clock signals, which are cyclic signals used to control the chip, data signals are generally the result of some Boolean computation. A data signal starts in either a primary input or a register, traverses combinatorial gates and ends in a primary output or a register. This gives rise to the fine *timing graph*, a non-empty directed acyclic graph $D = (V, E)$. The vertices correspond to *pins*. More formally, we have $V(D) = P_g \dot\cup P_{inp} \dot\cup P_{out}$. The sets P_{inp} and P_{out} correspond to starting points and endpoints of signals respectively. As described earlier, these pins are precisely given by register inputs or outputs and primary inputs or outputs. The remaining pins P_g are given by pins of combinatorial gates. The timing graph of a circuit is sketched in Figure 2.2. An edge in the timing graph is present if and only if a signal passes along the corresponding pins.

At every point in time, the electrical signal can be measured and corresponds to a certain voltage between 0 and V_{dd}. Here, V_{dd} is the operating voltage of the chip. Technically, it is possible that a chip has multiple voltage domains with different values of V_{dd}.

When the signal passes through the chip, the exact characteristics of the signal propagation are given by solutions to differential equations. Unfortunately, for the general case no closed form of these solutions is known. As solving differential equations is a costly operation, approximations are used. If the signal at a pin changes, its voltage either transitions from 0 to V_{dd} or vice versa. Therefore, we distinguish between rising and falling signals. A sample

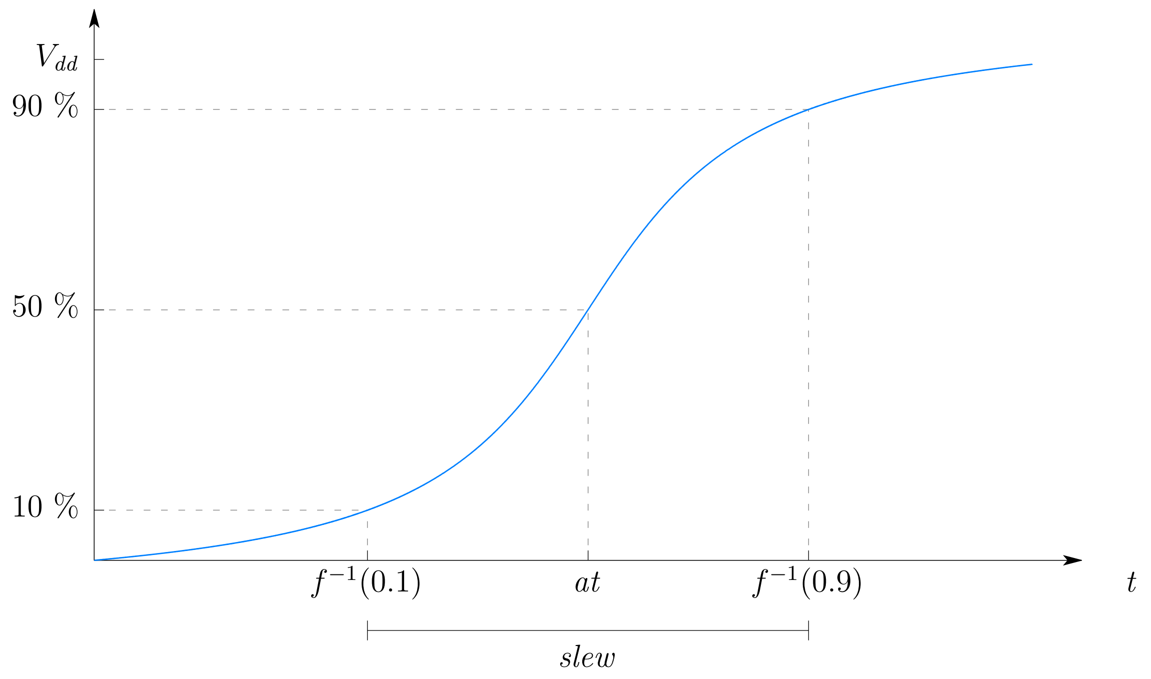


Figure 2.1: A sample voltage curve $f(t)$ of a rising signal. The *slew* is the time the signal takes to rise from $\frac{1}{10}V_{dd}$ to $\frac{9}{10}V_{dd}$ and the arrival time *at* is the time the signal takes to rise from the ground voltage to $\frac{1}{2}V_{dd}$.

voltage curve of a rising signal is given in Figure 2.1. We define the time of arrival corresponding to the signal as the point in time in which the voltage surpasses $\frac{1}{2}V_{dd}$. The *slew* of the signal is the timespan in which the signal rises from $\frac{1}{10}V_{dd}$ to $\frac{9}{10}V_{dd}$. Usually, not only constraints on the time of arrival of a signal are imposed, but also bounds on the slew have to be obeyed.

Before we describe the static timing analysis, a model to obtain signal delays for every logic path, we quickly describe the pins of combinatorial cells P_g. These vertices $P_g \subset V$ in the timing graph correspond to input and output pins of a collection of *gates* $\mathcal{G}$, which are implementations of Boolean functions. An illustration of a simple timing graph with the corresponding gates is given in Figure 2.2. Instead of building every gate separately on a transistor level, these gates are typically selected from a technology-dependent library of *books* $\mathcal{B}$. This library contains a set of pre-designed layouts for at least a complete set of logical functions such that every Boolean function can be expressed. The books may differ in their size, number of transistors and threshold voltages (V_t level), which influence the timing characteristics of the gate. An implementation that uses a larger number of transistors may be faster at the cost of more placement usage and power consumption. Again, we encounter a resource sharing problem that consists of selecting the right implementations such that timing constraints can be met while minimizing resource consumption.

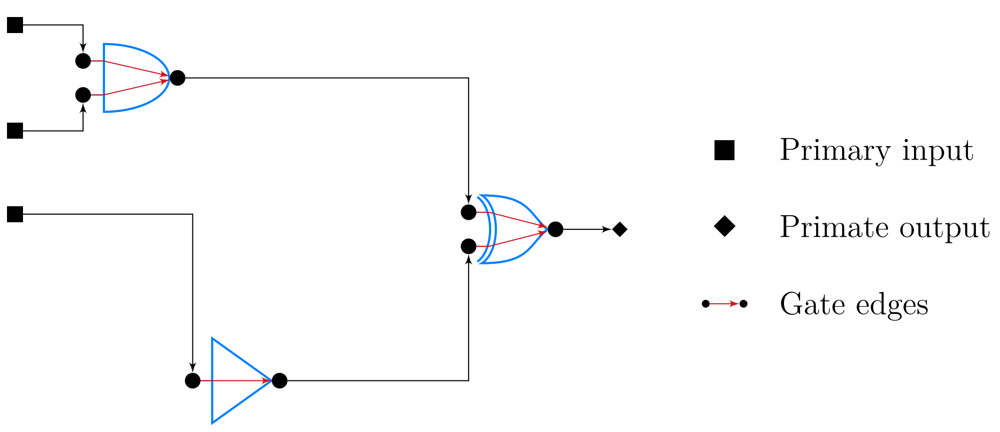


Figure 2.2: An example of a timing graph. Blue gates indicate how the graph arises from the underlying chip.

An assignment of all gates to books $\beta : \mathcal{G} \to \mathcal{B}$ is called technology mapping. If the assignment differs from the initial solution only in the size of the book and its threshold voltage (i.e., a gate is mapped to a possibly different implementation of the same logical function), we call β a *cell selection*. Even when only optimizing threshold voltages of the gates, finding a cell selection that minimizes power consumption is a difficult problem as we will see in Chapter 5. If we fix the technology mapping, we can assign every edge in the timing graph $(v, w) \in E(D)$ some delay value $\text{delay}_{v,w}(\beta) \geq 0$. For gate edges this value may be part of the library, but the delay of a gate edge usually does not only depend on the book of the corresponding gate, but also on that of adjacent ones.

2.1 Timing Analysis

If we fix the technology mapping β and delay values of every edge, we can use static timing analysis to compute the time it takes for a signal to traverse the timing graph. To this end, we assume that for all input ports $p \in P_{inp}$ fixed *arrival times* $\text{at}(p) \in \mathbb{R}$ are known. These represent the time at which the signal arrives at these inputs. Similarly, the output ports $p \in P_{out}$ have *required arrival times* $\text{rat}(p)$, which represent the latest acceptable time at which the signal may arrive at these ports. For an inclusion-wise maximal path $P \in \mathcal{P}$ in the timing graph, we can define its delay by

$$\text{delay}(P) = \sum_{(v,w) \in E(P)} \text{delay}_{v,w}(\beta).$$

It is easy to see that the timing constraints are satisfied, if and only if $\text{delay}(P) \leq \text{rat}(t) - \text{at}(s)$ for every s-t path P in the timing graph. At first, this description seems to be impractical due to the exponential size of $|\mathcal{P}|$, but we may always sample a violated path in polynomial time. Indeed, we will see in Section 3.3.3 that this formulation can be very useful.

One way to turn the path formulation into polynomial constraints is given by introducing arrival times. As the timing graph is acyclic, we can propagate the arrival times in topological order by setting

$$\operatorname{at}(w) := \max_{(v,w)\in E(D)} \operatorname{at}(v) + \operatorname{delay}_{v,w}(\beta).$$

Analogously we can propagate required arrival times in reverse topological order by setting

$$\operatorname{rat}(v) := \min_{(v,w)\in E(D)} \operatorname{rat}(w) - \operatorname{delay}_{v,w}(\beta).$$

Comparing the arrival times with the required arrival times shows us if the chip meets the timing requirements. For this, we define the *slack* of a pin $p \in V(D)$ as

$$\operatorname{slack}(p) := \operatorname{rat}(p) - \operatorname{at}(p).$$

A major goal of timing optimization is to find a technology mapping that guarantees $\operatorname{slack}(p) \geq 0$ for all pins $p \in V(D)$.

2.2 The Linear Timing Model

The delay of a wire segment increases roughly quadratically with its length. Thus, in particular on low layers, delays can quickly become prohibitive. In contrast, an optimally buffered point-to-point connection has roughly linear delay. A *buffer* is a gate which reinforces the electrical signal by repeating it. It can either implement the identity function or invert the electrical signals. An even number of inverting buffers can be used to ensure logical correctness of the chip. In early design stages, before buffers have been inserted, one usually linearizes delays.

In the linear timing model, all net edges $(p, q) \in E(D)$ have a specific fixed delay

$$\operatorname{delay}_{p,q}(\beta) = d^{edge}_{p,q}.$$

Note that the delay does not depend on the technology mapping β. For a gate edge $(v, w) \in E(D)$ of some gate $g \in \mathcal{G}$ we have

$$\operatorname{delay}_{v,w}(\beta) = d^{gate}_{\beta(g)}.$$

In particular, the delay of the gate g does not depend on the mapping $\beta(g')$ for

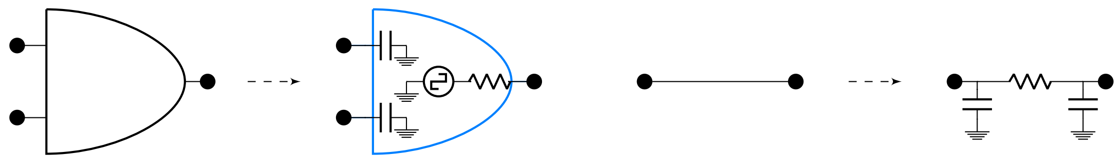


Figure 2.3: Transformation of a gate and a wire into an RC circuit.

$g \neq g' \in \mathcal{G}$.

2.3 Elmore Delay

After buffer insertion a linear delay model is too optimistic. A pin of high capacitance has to be connected large gates in order to reach a reasonable timing. The Elmore delay model [Elm48] is significantly more accurate than the linear model but is still simple enough to be computed efficiently. It has the useful property that it is pessimistic, i.e., the real delay is never underestimated. Therefore, a solution that satisfied all timing constraints under Elmore delay is also feasible under more accurate models. However, it may use an excessive amount of inverters and too many routing resources.

Elmore delay is an RC delay model, it assumes that the interconnect is given by a network of resistors and capacitors. An approach by Fishburn et al. [FD85] transforms books and wiring segments into such a network. Here, a book $b \in \mathcal{B}$ is characterized by four values: a size x_b, a base resistance $\widehat{r_b}$, a base pin capacity $\widehat{c_b}$ and a base power factor α_b. For a given $b \in \mathcal{B}$ we now replace every input pin with a capacitor of a grounded capacity $x_b\widehat{c_b}$ and an ideal power source. We connect the power source to the output pin by a resistor of resistance $\frac{\widehat{r_b}}{x_b}$. The power consumption of the book is given by $\alpha_b x_b$. To model wires we use a base resistance $\widehat{r_w}$ and a base capacitance $\widehat{c_w}$. We insert a capacitor of capacity $\frac{\widehat{c_w}}{2}$ before and after the resistor of resistance $\widehat{r_w}$. This transformation is illustrated in Figure 2.3.

We can now explain how the Elmore delay of an edge in the timing graph can be computed. For a gate edge $(v, w) \in E(D)$ of a gate $g \in \mathcal{G}$ we set

$$\text{delay}(v, w) = \frac{\widehat{r_g}}{x_g}\text{downcap}(w).$$

Here, downcap(w) computes the *downstream capacitance* of the pin w, this is the sum of all wire capacitances $\widehat{c_w}$ of the Steiner tree connecting w to downstream gates and all input pin capacitances $x_{g'}\widehat{c_{g'}}$ of these connected gates. For an edge (p, q) this Steiner tree represents the wiring that connects the vertices $\delta^+_D(p) \cup \{p\}$. We can give it a natural orientation by directing the

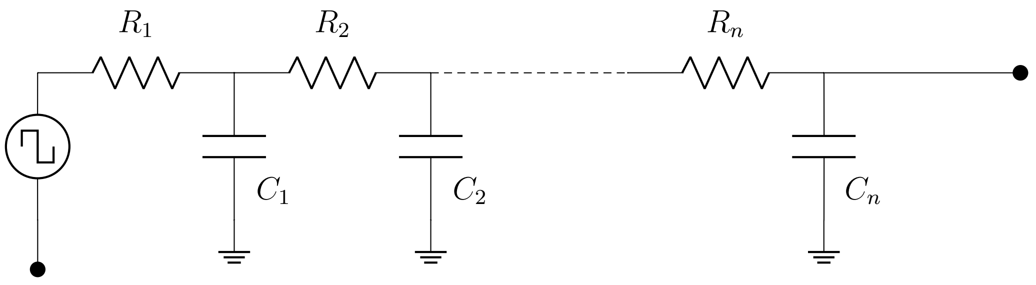


Figure 2.4: The shielding issue. The effective capacitance measured at the first resistor may be significantly smaller than $\sum_{i=1}^{n} C_i$.

edges from the logical source to the sinks. Therefore, it can be interpreted as a directed arborescence Y^p. Let $Y[p,q]$ denote the unique $p-q$ path in that arborescence. We can now set

$$\text{delay}(p,q) = \sum_{e=(v,w)\in E(Y[p,q])} \widehat{r_e}\left(\frac{\widehat{c_e}}{2} + \text{downcap}(w)\right).$$

In this formula downcap(w) is the sum of all wire and pin capacitances of the sub-arborescence rooted at $w \in V(Y^p)$. With a discrete library the sizes x_g are usually fixed, sometimes we relax the problem and allow the size to be within a specified region $l_g \le x_g \le u_g$.

We see that the delay of a path P can be written as

$$d(P) = \sum_{k=1}^{K} c_k \prod_{g\in\mathcal{G}} x_g^{b_{g,k}}$$

where $c_k > 0, b_{g,k} \in \{-1,0,1\}$. Functions of this form (even for general $b_{g,k} \in \mathbb{R}$) are called *posynomial.* Posynomial functions can be turned into convex functions by a simple variable transformation [HH16].

2.4 Higher Order Delay Models

While the Elmore delay model can be computed efficiently, it is too inaccurate to be used in late stages of the design process. One reason is that it often significantly overestimates capacitances. A worst-case example is given by a path of n resistors of resistance $R_1,\ldots,R_n$ where after every resistor i we insert an antenna of capacity C_i. This instance is depicted in Figure 2.4. According to the Elmore delay, we have a delay of $R_1\sum_{i=1}^{n} C_i$ at the first resistor. In reality, the downstream resistors shield away a major part of the capacitance that is near the end of the path.

The *effective capacitance* $\sum_{i=1}^{n} C_i^{\mathit{effective}}$ is the real capacitance visible at the first resistor (subtracting the capacitance that is shielded away by downstream resistors). More accurate delay models like RICE [RP94] approximately solve differential equations to determine these effective capacitance values. Current-based models as described by Croix et al. [CW03] are slightly less accurate but much faster and, therefore, commonly used.

2.5 Power Analysis

While power was mostly a secondary objective in older technologies, it is becoming increasingly important as transistor sizes shrink. Modern chips have a huge power dissipation, which can even become the limiting design bottleneck.

The power consumption P_{total} depends on the technology mapping $\beta : \mathcal{G} \to \mathcal{B}$ and on the interconnect. A part of this power consumption is a permanent leakage, which always occurs when the chip is powered on and is constant over time. We denote this by the *static power* P_{static}. The second part is the dynamic power P_{dynamic}. In our model the dynamic power consumption only occurs when the transistors switch and depends on the switching frequency χ_g which can also depend on the specific usage of the chip. In addition to the selection of gates, nets that switch frequently should be as short as possible to minimize power dissipation. For simplicity, one usually assumes that these power functions are separable and can be computed by

$$P_{\mathit{static}}(\beta) = \sum_{g \in \mathcal{G}} \text{static\_power}(\beta(g)),$$

$$P_{\mathit{dynamic}}(\beta) = \sum_{g \in \mathcal{G}} \text{dynamic\_power}(\beta(g)).$$

There are also other kinds of dynamic power, for example dynamic leakage. In practice, tight slew bounds make sure that our model is reasonably accurate. The interested reader is referred to the book chapter by Held and Hu for more details [HH16].

In the model by Fishburn et al. [FD85] a given gate $g \in \mathcal{G}$ is assumed to have a continuous size $x_g \in [L, U]$ such that its input capacitance is given by $\widehat{c_g} x_g$. As explained previously, gates may also have different threshold voltages V_t. This can be achieved by using different substrates for manufacturing transistors. A lower threshold voltage increases the static power consumption and lowers the threshold at which the transistor can switch its state, lowering the delay. As there is usually a small discrete set of possible threshold voltages, we also speak of V_t levels.

A typical assumption is that the static power consumption of a gate is

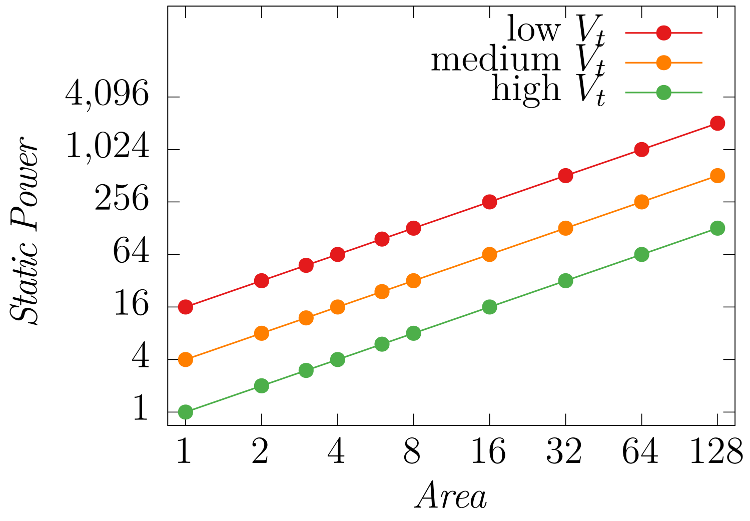


Figure 2.5: The cell library for an inverter as given in the ISPD 2013 contest. The axes are plotted logarithmically. The three different colors indicate different threshold voltages. High V_t corresponds to the slowest implementation of a fixed size, while low V_t is the fastest one.

proportional to the size of the gate and exponential in the V_t level of the gate. In contrast, the dynamic power consumption does not depend on the V_t level, which means that if for a given gate $g \in \mathcal{G}$ the switching activity χ_g is high it can be preferable to accelerate it by lowering the V_t level instead of increasing the size.

An example of a cell library is given in the ISPD 2013 contest for gate-sizing [Ozd+13]. Here, we have $|X_g| = 10$ and for each of these sizes there are 3 possible V_t levels which gives us 30 possible implementations for the gate g. The library for an inverter is visualized in Figure 2.5.

2.6 Timing Metrics

In the previous sections, we explained how the timing of a chip can be modeled by introducing the timing graph and propagating arrival times. However, early in the design process not all timing constraints can be met. Instead of simply checking for feasibility, we try to find objective functions that can drive our optimization routines and penalize timing fails.

These metrics will allow us to compare different solutions, which do not completely meet all timing constraints. A different application is to use the timing metrics directly within optimization routines. A simple example is an algorithm by Kahng et al. [Kah+13] which counts the number of failing paths passing through a gate.

We can evaluate the timing of the chip whenever we have delays available for all edges in the timing graph. This motivates the following definition. A *timing solution* is a tuple (D, d, T), where D is a timing graph, $d : E \to \mathbb{R}_{\geq 0}$ is a delay function and $T \geq 0$ is a deadline. In chip design T roughly corresponds to the cycle time at which the chip operates. A path P in the timing graph meets its timing constraints when the sum of its edge delays $d(P) := \sum_{e \in E(P)} d(e)$ does not exceed T. Note that we can always assume that our delay constraints are encoded by such a deadline by inserting a super-source and a super-sink with arcs that correspond to the arrival times at the source pins and the required arrival times at the sink pins.

A *timing metric* is a map $\tau : (D, d, T) \mapsto \tau(D, d, T) \in \mathbb{R}$ that assigns every timing solution a real number that should correspond to the timing feasibility of the instance. If D and T are fixed, we write $\tau(d)$ for $\tau(D, d, T)$.

We will now give an overview of the most relevant timing metrics that are used in chip design. The most common is the *worst slack* timing metric

$$\text{WS}(d) := \min_{P \in \mathcal{P}} \min\Big\{T - d(P), 0\Big\}.$$

The most important property of the worst slack metric is that it is non-negative if and only if all timing constraints are met.

To define more advanced timing metrics we will introduce a couple of additional definitions. Recall that in the timing graph $P_{inp} \subset V(D)$ are precisely the vertices without incoming edges, and $P_{out} \subset V(D)$ the vertices without outgoing edges. For a fixed $v \in V(D)$ we define the longest path through v by $P_{max}(v) = \text{argmax}_{P \in \mathcal{P}, v \in V(D)} d(P)$ and similarly for an edge $e \in E(D)$ we define $P_{max}(e) = \text{argmax}_{P \in \mathcal{P}, e \in E(P)} d(P)$. In case of a tie we choose the path lexicographically minimal with respect to some fixed order on the edges. Note that P_{max} can be evaluated in linear time by traversing the graph in topological and reverse topological order.

We can now define the total negative slack (also called “figure of merit” or “sum of negative slacks”) timing metric. It is defined as

$$\text{TNS}(d) := \sum_{v \in P_{out}} \min\Big\{T - d(P_{max}(v, d)), 0\Big\}.$$

One can easily see that $\text{TNS} \leq \text{WS}$.

A timing metric that was proposed by Reimann et al. [RSR16b], which we name “true total negative slack”, is defined as follows: Let $Q = \cup_{e \in E(D)} P_{max}(e, d)$. Note that we may have $|Q| < |E(D)|$.

$$\text{TTNS}(d) := \sum_{P \in Q} \min\Big\{T - d(P), 0\Big\}.$$

The TTNS can also be computed by traversing D in reverse topological order and for fixed $v \in V(D)$ summing up the longest paths through every but the most critical edge in $\delta^-(v)$, where ties are broken arbitrarily. This yields a number $\rho(v) \in \mathbb{R}$. One can see that $\text{TTNS}(d) = \text{TNS}(d) + \sum_{v \in V} \rho(v)$. In particular we have $\text{TTNS}(d) \leq \text{TNS}(d)$.

A natural extension would be to sum up the slack of all paths. Namely we could compute the path total negative slack as follows

$$\text{PTNS}(d) := \sum_{P \in \mathcal{P}} \min\Big\{T - d(P), 0\Big\}.$$

It turns out that there is a problem with this approach. It is easy to see by shifting the deadline by 1 that computing PTNS would also allow us to compute the negative number of violated paths:

$$\text{NPATH}(d) := -|\{P \in \mathcal{P} : d(P) > T\}|.$$

However, this is a hard problem as shown by Mihalák et al. [MSW14]:

Theorem 2.1. (Mihalák et al.) [MSW14] *Given an acyclic digraph $G = (V, E)$ and two vertices $s, t \in V(G)$, and some $L \in \mathbb{R}$ then it is #P-complete to count the number of s-t paths of length at most L.*

Here #P is a class of NP-hard counting problems (as these do not have a certificate of polynomial size). A trivial modification to the given reduction from the partition problem shows that unless P=NP there is no polynomial algorithm to compute PTNS or Npath. We remark that there is a polynomial algorithm to approximate NPATH, however its runtime of $\mathcal{O}(mn^3\epsilon^{-1}\log n)$ does not seem practical [MSW14].

We end this section with the remark that a method was proposed by Kong [Kon02] to estimate the number of violated paths. It works by counting the number of all paths in a graph, which arises from the timing graph by deleting edges (p, q) if either $\text{slack}(p) \geq 0$ or $\text{slack}(q) \geq 0$. This can be achieved by propagating the number of paths through a pin in both forward and backward topological order and by multiplying these values. However, this is not equivalent to computing NPATH. A counterexample was given by Daboul [Dab15].

2.7 An Overview

Before we analyze some mathematical problems arising in the design process, it is helpful to roughly understand the physical design flow. As an input the flow starts with a pre-optimized logic description of the chip. Corresponding logic

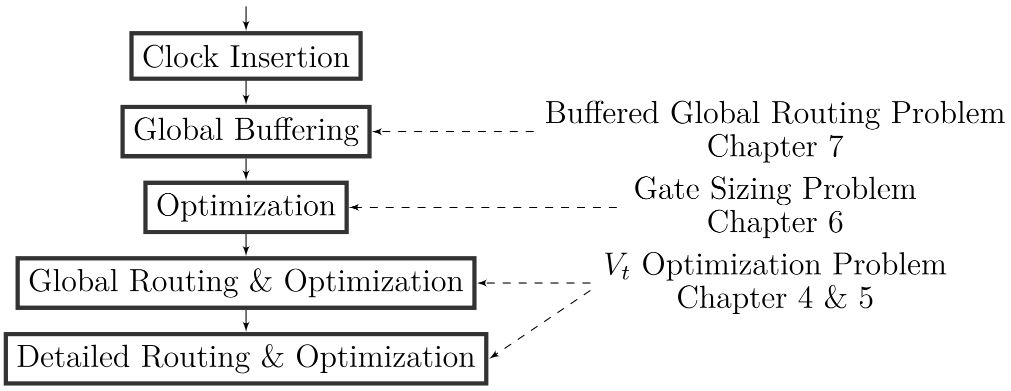

Clock Insertion
Global Buffering
Optimization
Global Routing & Optimization
Detailed Routing & Optimization
Buffered Global Routing Problem
Chapter 7
Gate Sizing Problem
Chapter 6
V_t Optimization Problem
Chapter 4 & 5

Figure 2.6: An excerpt of an industrial design flow proposed by Li et al. [Li+12]. On the right a list of mathematical problems discussed in this thesis is shown. Each problem is connected to its corresponding design steps.

gates are subsequently placed onto the chip area in the *global placement step* and optimized. An extended overview on a modern design flow was given by [Li+12]. A final step of the flow is to compute a *detailed routing*, which consists of wires connecting all gates on a chip. In addition to being overlap free, various other technological constraints have to be met. As it is infeasible to optimize at this level of accuracy, earlier steps of the flow use simple approximations that should make sure that a later detailed routing can still be found.

One way to ensure that enough wiring space is available to find a detailed routing is to coarsify the chip and to add capacity constraints on the estimated wiring in this simplified graph. The arising multicommodity flow formulation was first considered by [SK87] and the corresponding process of computing rough outlines for the later detailed wiring is called *global routing*.

In the clock network synthesis step thousands of clock trees have to be inserted. As the step can be very disruptive for the timing and wiring space usage of the chip, it is usually performed early in the design flow [Li+12]. For optimization before this step, the effects on the final result at the end of the flow can be marginal. Therefore, we will not focus much on the steps that precede clock network synthesis. Instead, our primary focus will be the global buffering step and subsequent optimization steps with and without a global routing.

A sketch of the design flow is given in Figure 2.6. In Chapter 7 we will discuss an algorithm for the global buffering problem. Our implementation of the resource sharing algorithm for gate sizing described in Chapter 6 is applied in the subsequent optimization step. After global routes have been computed, disruptive optimizations like gate sizing and buffering should be avoided. As V_t

changes often leave gate footprints unchanged and do not require movements, the V_t optimization problem is well suited for applications after a global or even detailed routing. We will discuss it in Chapter 4 and 5. In the next chapter we familiarize the reader with the global routing problem and the resource sharing problem.

Chapter 3

Interconnect Optimization

This thesis deals with different resource sharing problems that occur in the chip design process. One fundamental resource is wiring space. To understand how we solve the overall problem of distributing routing and timing resources to individual connections, we first have to explain how this problem is to be solved if only a single resource, namely wiring space, is to be optimized.

The wiring on a chip is structured in *routing layers*, these have a preference direction that either allows vertical or horizontal wiring. The set of feasible wiring segments gives rise to the so-called *track graph*. This graph has an edge for every possible wiring segment that adheres to the preference direction of the particular layer. Vertical wire segments which allow switching between adjacent layers are called vias. Maximal subgraphs of wiring on a chip are named *routes*. Every route connects a logical source to a set of sinks. These pins connected by a route are called a *net* and denoted as $N \in \mathcal{N}$. The set of all nets $\mathcal{N}$ is the *netlist*.

To ensure correct function, all wire segments have to be disjointly packed onto the chip while obeying various technology dependent spacing constraints. Optimizing a chip on this level of accuracy does not seem to be feasible with current technology and algorithms. Therefore, one usually relaxes the disjointness constraints. Instead, we coarsen the track graph into a grid graph and introduce capacity constraints on the edges. This process is illustrated in Figure 3.1.

In this thesis we will not consider detailed routing problems. Instead, we will always use the simplifying assumption that interconnects are computed in the global routing graph. However, especially for buffer insertion, we may compute routes that already connect to the specified metal shapes corresponding to a given net. For computing the routing space consumption, the route is then first projected into the global routing graph.

The global routing problem consists of finding a feasible fractional Steiner tree packing in this grid graph. Approximation algorithms for fractional multicommodity flows can be used to solve this problem and were already used in chip design over 30 years ago [SK87].

Since its initial formulation, many new constraints have been introduced to the global routing problem. Modern algorithms do not only solve a packing problem, but also optimize timing and perform simultaneous buffer insertion or placement optimization.

3.1 Previous Work

As one of the most fundamental problems in chip design, the global routing problem has received a significant amount of attention in the past. The multicommodity flow formulation [SK87; CC91] has been extended to integrate net delay bounds and buffer insertion by [Alb+02], who presented a fully polynomial approximation scheme (FPTAS) for two-terminal nets.

A state-of-the-art algorithm for the classic global routing problem without timing constraints was given by Müller, Radke and Vygen [MRV11]. Their algorithm is based on a special multiplicative Lagrangean multiplier framework, called resource sharing.

Based on this algorithm Saccardi and Hähnle presented a novel approach which works with a rhomboidal subdivision of the chip [HS19]. By a structure theorem this allows them to compute global routes with a pin-level accuracy. This significantly improves the correlation with a later detailed routing.

Giving an exhaustive overview on previous global routing approaches would exceed the scope of this thesis. Many recent algorithms used in contests or during industrial design flows belong to a class of rip up and reroute heuristics. They often work on a 2d model of the routing graph and iteratively rewire the nets to reduce overuse of routing edges. Heuristically updating a cost function can work well in practice, but lacks a provable performance guarantee. Some notable examples are NTHU-Route [Cha+10], FastRoute [YYC09] and BoxRouter [Cho+09]. A different notable approach uses an integer program formulation for the global routing problem. The GRIP algorithm [WDL11] solves the overall integer program by solving smaller subinstances with a price-and-branch heuristic.

Integrating timing constraints into global routing has a rich history. First approaches by Huang et al. [Hua+93] used delay bounds to reject Steiner trees which do not meet these allocated budgets. This approach was later refined by Hong et al. [Hon+97] to path-based delay bounds. These approaches have several serious drawbacks. By seeing delay bounds as a hard constraint rather than a flexible resource, timing has to be significantly relaxed to prevent a couple of failing paths from dominating the overall solution. This weakness was first addressed by Vygen [Vyg04], who showed how to implement net based delay bounds into the resource sharing framework. The drawback is that only a subset of the timing constraints are considered.

More recently, Held, Müller, Rotter, Scheifele, Traub, and Vygen [Hel+17] showed how to integrate global static timing constraints into the resource sharing framework. This is the first solution which does not have the aforementioned problems of using fixed delay budgets or only considering a subset of the timing constraints. An alternative way of integrating path based timing constraints was given by Hähnle [Häh15].

Rotter [Rot17] presented a resource sharing formulation that combines routing and buffer insertion. He also implemented a heuristic oracle function that allowed him to obtain first promising results on practical instances. However, compared to the previous state-of-the-art design flow some metrics were still inferior. In Chapter 7 we will present a refined implementation which removes remaining drawbacks and significantly improves the design flow.

The algorithm of Rotter still used the classic routing graph description of [SK87; CC91] but heuristically connects global routes to pin positions before buffer insertion. It seems tempting to combine ideas of Rotter with the work of Saccardi and Hähnle [HS19] in the future to directly compute global routes with pin-level accuracy.

Previous approaches usually follow up a timing-unaware global routing step with a successive layer assignment and buffering step. An example is given by the CATALYST algorithm [WA04], which was the previous default method in the PDS-Turbo design flow [Li+12] used by IBM.

We will give a more detailed overview about previous work on the buffering problem in Chapter 7.

3.2 Global Routing

For our applications, we will assume that the chip has a rectangular outline and a set of wiring layers. Let the *chip image* be defined as $\mathcal{I} = \square \times \{0, \dots, Z\}$, where $\square = [0, w] \times [0, h]$ is the area of the chip and Z is the number of routing layers. Layer '0' corresponds to the placement area $\square \times \{0\}$. All repeaters and gates are inserted on this layer and the routing space has to be accessed by using vias.

Before we consider the step of buffer insertion, we will first explain how the global routes can be computed. Therefore, assume for the moment that all gates are part of the input and only a global routing should be computed for the nets. In addition to the global routing graph, we are given a set $\mathcal{T}$ of *wire types*. In conjunction with an axis-parallel line segment $I \subset \mathcal{I}$, a wire type $\tau \in \mathcal{T}$ determines a space consumption, as well as electrical resistance and capacitance of the metal shape pair (I, τ). In practice, not all wire types can be used on every routing layer. This can be modeled by assigning the combination

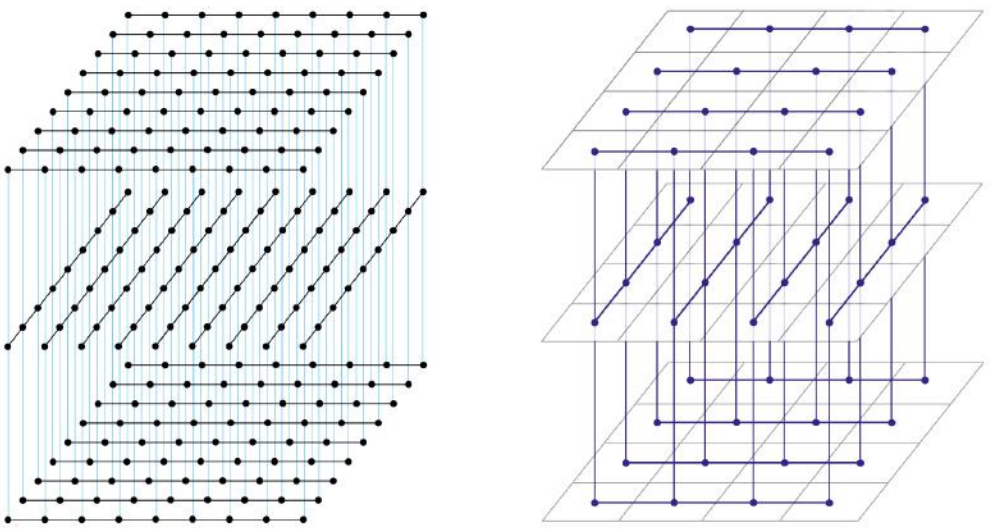


Figure 3.1: An illustration of the track graph and the corresponding global routing graph. On the left side, the track graph is shown. The edges of the track graph correspond to possible wiring segments on the chip. The right side shows a possible global routing graph. Instead of computing disjoint wiring segments, the global routing graph can be used to assign the individual nets to certain routing corridors. These only obey a capacity constraint on the total use of a global routing edge, which should assert that in the end a feasible solution can still be found. The vertical distance between two adjacent layers was artificially increased for better visibility.

an infinite resistance and capacitance.

Each net $N \in \mathcal{N}$ consists of a source pin $s \in \Box \times \{0, \dots, Z\}$ and a set of sink pins $T \subset \Box \times \{0, \dots, Z\}$. In particular for non-linear timing models, the order in which sinks are connected to the root is essential.

A *topology* for a net N is an arborescence A rooted in s, such that T is the set of leaves. We require that the root has exactly one successor and all inner vertices have outdegree at most 2. This degree constraint is only for convenience, as we can always reduce degrees by introducing additional vertices to the arborescence.

An *(embedded) Steiner tree* for a net N is a topology A together with a map into the chip image $p : V(A) \to \mathcal{I}$. p has to satisfy $p(s) = s, p(t) = t$ for the source s and any sink $t \in T$. We further require that the image $(p(v), p(w))$ for edges $(v, w) \in E(A)$ are either of length 0 or axis-parallel routing stick figures, i.e., they have to correspond to an edge in the global routing graph.

With these definitions we can define a first version of the Global Routing problem, by looking for an (embedded) Steiner tree for every net $N \in \mathcal{N}$, such that overall capacity bounds are obeyed. When it is clear from the context that we describe an embedded Steiner tree in the chip image, we sometimes omit “embedded” and simply write Steiner tree to simplify notation.

Problem 1: Traditional Global Routing Problem

Input: The global routing graph G, edge lengths $l : E(G) \to \mathbb{R}_{\geq 0}$, a netlist $\mathcal{N}$, wire types $\mathcal{T}$ and a routing space oracle $\mathrm{usg} : E(G) \times \mathcal{T} \to \mathbb{R}_{\geq 0}$.

Task: Compute Steiner trees (A_N, p) for all $N \in \mathcal{N}$ with $p(V(A_N)) \subseteq E(G)$ with wire type assignments $\tau_e \in \mathcal{T}$ for all $e \in E(A_N)$. For a global routing edge $\{i, j\} \in E(G)$ let $Q^N_{i,j} := \{(v, w) \in E(A_N) | (p(v), p(w)) = (i, j)\}$ be the edges mapped to it by A_N. The Steiner trees have to obey capacity constraints

$$\sum_{N\in\mathcal{N}} \sum_{e\in Q^N_{i,j}} \mathrm{usg}(p(e), \tau_e) \leq 1 \quad \text{for all } \{i, j\} \in E(G).$$

The objective is to minimize netlength: $\sum_{N\in\mathcal{N}} l(E(A_N))$.

The main goal of this Chapter is to first introduce the min-max resource sharing problem, which allows us to solve the fractional relaxation of the traditional global routing problem up to the Steiner ratio. Finally, we will focus on the more general problem of simultaneous routing and buffering.

3.3 Min-Max Resource Sharing

We follow the definitions introduced in [MRV11] and use the following problem formulation.

Problem 2: Min-Max Resource Sharing Problem

Input: A finite set $\mathcal{R}$ of *resources*, a finite set $\mathcal{C}$ of *customers*. For every customer $c \in \mathcal{C}$ a convex set B_c. For all customer resource pairs $(c, r) \in \mathcal{C} \times \mathcal{R}$ a convex usage function $\mathrm{usg}_{c,r} : B_c \to \mathbb{R}_{\geq 0}$.

Task: Compute a vector of solutions $b = (b_c)_{c\in\mathcal{C}}$, such that $b_c \in \mathcal{B}_c$ for all $c \in \mathcal{C}$. The objective is to minimize the maximum resource consumption

$$\lambda(b) = \max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} \mathrm{usg}_{c,r}(b(c)).$$

The intuition behind this definition is that a set of limited resources is to be distributed to the customers. A solution $b \in B_c$ decides how much of a given resource the customer uses. In our applications the sets B_c, which are also called *blocks*, will always be compact. Therefore, the optimum solution is well defined. The sets of possible solutions B_c for a customer can be exponentially or even infinitely large; therefore they may only be given implicitly.

To be able to solve the resource sharing problem, we assume that *oracle functions* are given. For a customer $c \in \mathcal{C}$ an *oracle function* $f_c : \mathbb{R}^{\mathcal{R}}_{\geq 0} \to B_c$ is an approximate minimizer of the weighted resource minimization problem. Formally, we assume that there is a constant $\sigma \geq 1$, such that

$$y^\top f_c(y) \leq \sigma \inf_{b \in \mathcal{B}_c} \sum_{r \in \mathcal{R}} y_r \mathrm{usg}_{c,r}(b) \qquad \text{for all } y \in \mathbb{R}^{\mathcal{R}}_{\geq 0}.$$

This generalization of the multicommodity flow problem allows us to see the fractional relaxation of the traditional global routing problem from a different perspective. We consider the nets $\mathcal{N} =: \mathcal{C}$ as customers of a resource sharing problem. The feasible solutions are given by the convex hull of all feasible Steiner trees for the net. We introduce a resource for every edge in the global routing graph and a netlength resource; thus $\mathcal{R} = E(G) \cup \{\text{netlength}\}$.

The usage function $\mathrm{usg}_{N,e}$ specifies how much the wiring segments of a Steiner tree A_N with given wire code assignment consume from the global routing edge $e \in E(G)$ relative to its capacity. By performing a binary search, we may assume that we know the best possible netlength in form of a budget. This directly corresponds to a resource which measures how much the budget is violated. In the next section, we will give an outline of an algorithm that can solve the resource sharing problem if appropriate oracle functions are known.

Due to its general formulation, the resource sharing problem is well suited for its application in chip design. Numerous objectives can be encoded by defining appropriate resources. By focusing on relative resource consumptions it also provides a natural way to compare design objectives that have different units.

3.3.1 The Resource Sharing Algorithm

In this section we will describe the algorithm of Müller, Radke, Vygen [MRV11] to solve the aforementioned resource sharing problem. To this end, assume that we are given an instance of the resource sharing problem and oracle functions for each customer. The main idea is to use a multiplicative update scheme which will generate the costs to use in our oracle function. The pseudocode of it is given as Algorithm 3.1.

The most important step of the algorithm is the exponential price update $\mathrm{price}(r) \leftarrow e^{\gamma\xi \mathrm{usg}_{c,r}(b)} \cdot \mathrm{price}(r)$. Rip up and reroute heuristics for the global routing problem usually use similar exponential price updates to avoid overuse of routing edges. However, the algorithm shows that this price update can

Algorithm 3.1: Resource Sharing Algorithm by [MRV11].

```
Input: An instance of the (fractional) min-max resource sharing
       problem. Oracle functions $f_c$ for all $c \in \mathcal{C}$. $\gamma > 0, t \in \mathbb{N}$.
Output: A solution $x_c = \sum_{b \in B_c} x_{c,b} b$ for all $c \in \mathcal{C}$.
1  for $r \in \mathcal{R}, c \in \mathcal{C}, b \in B_c$ do
2      price($r$) $\leftarrow 1, X_c \leftarrow 0, x_{c,b} \leftarrow 0$          ▷ Initialization step
3  for $p = 1 \dots t$ do
4      for $c \in \mathcal{C}$ s.t. $X_c < p$ do
5          $b \leftarrow f_c$(price)
6          $\xi \leftarrow \min\{p - X_c, \min\{1/\text{usg}_{c,r}(b) : \text{usg}_{c,r}(b) > 0, r \in \mathcal{R}\}\}$
7          $x_{c,b} \leftarrow x_{c,b} + \xi, X_c \leftarrow X_c + \xi$
8          for $r \in \mathcal{R}$ do
9              price($r$) $\leftarrow e^{\gamma \xi \text{usg}_{c,r}(b)}$price($r$)     ▷ Exponential price update
10 for $c \in \mathcal{C}, b \in B_c$ do
11     $x_{c,b} \leftarrow x_{c,b}/t$                                   ▷ Scale the solution by $1/t$
```

be done in a way that provably converges. A main result of [MRV11] is the following theorem.

Theorem 3.1 (Müller, Radke, Vygen [MRV11])**.** *For any $\omega > 0$, Algorithm 3.1 can be used to compute a solution to the min-max resource sharing problem with approximation ratio $\sigma(1+\omega)$ in runtime $\mathcal{O}(\theta(|\mathcal{C}|+|\mathcal{R}|)\log|\mathcal{R}|(\log\log|\mathcal{R}|+\omega^{-2}))$. As above σ is the approximation guarantee of the oracle functions and θ the time to evaluate an oracle function.*

Let $\lambda^\star$ denote the optimum maximum resource usage of the given instance. In practice, we usually have $\frac{1}{2} \leq \lambda^\star \leq 2$, as otherwise the instance is either easy to solve or completely infeasible. If we use this assumption, the analysis of the resource sharing algorithm yields a slightly better runtime of $\mathcal{O}(\theta(|\mathcal{C}| + |\mathcal{R}|)\log|\mathcal{R}|\omega^{-2})$. The main problem with this is that the runtime is still linear in the number of resources $|\mathcal{R}|$. In Section 3.3.3 we will encounter a resource sharing problem with an exponential number of resources. An important indicator to estimate the hardness of a resource sharing problem is the problem width. It is defined to be the worst possible overuse of a resource. More precisely

$$\Lambda = \sum_{c \in \mathcal{C}} \max\{1, \sup_{r \in \mathcal{R}, b \in B_c} \text{usg}_{c,r}(b)\}.$$

For the traditional global routing problem, it is reasonable to assume that $\Lambda = \mathcal{O}(|\mathcal{N}|)$, as a single net certainly can be accommodated in any global routing edge. Müller, Radke and Vygen give the following useful bound, that allows problem formulations with exponentially many resources.

Theorem 3.2 (Müller, Radke, Vygen [MRV11]). *Let $0 < \delta, \delta' < 1$. Given an instance of the min-max resource sharing problem with $\lambda^\star \leq 1$, we can compute a $(\sigma(1+\delta) + \frac{\delta'}{\lambda^\star})$ approximation in time $\mathcal{O}(\sigma\theta \log |\mathcal{R}| \Lambda(\delta\delta')^{-1})$.*

The above algorithm can be used to solve the fractional min-max resource sharing problem. For the classic global routing problem, this yields a convex combination of Steiner trees for each net. As the discrete variant of the global routing contains the edge disjoint Steiner tree packing problem in grid graphs, we cannot hope for good algorithms to solve it. However, Müller Radke and Vygen show that rounding every net to a solution randomly, depending on its weight in the convex combination, yields a good solution. More formally, they show the following theorem.

Theorem 3.3 (Müller, Radke, Vygen [MRV11]). *Consider values $x_{c,b}$ computed by Algorithm 3.1. After scaling, we have $\sum_{b \in B_c} x_{c,b} = 1$. If we randomly round the solution by choosing solution b with probability $x_{c,b}$, we obtain solutions $\widehat{b_c}$ for all $c \in \mathcal{C}$. Let $\lambda = \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} \sum_{b \in B_c} x_{c,b} usg_{c,r}(b)$. Let $\widehat{\lambda} = \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} usg_{c,r}(\widehat{b_c})$. For $r \in \mathcal{R}$ let $\rho_r := \max\{usg_{c,r}(b)/\lambda | b \in B_c, c \in \mathcal{C}, x_{c,b} > 0\}$ and $\delta > 0$. Then $\widehat{\lambda} < \lambda(1+\delta)$ with probability at least $1 - \sum_{r \in \mathcal{R}} e^{-h(\delta)/p_r}$, where $h(\delta) := (1+\delta)\ln(1+\delta) - \delta$.*

This step is also called randomized rounding. In practice, it often leads to a good solution, which can be further refined by post-optimization. In the following sections, we will show how timing can be modeled within the resource sharing framework.

Depending on the resources in the problem formulation an oracle as used by the resource sharing algorithm may have different forms. For the timing constrained global routing problem we will discuss the oracle in Section 3.4. For the variant of simultaneous buffer insertion the oracle will be explained in Section 3.5.

3.3.2 Arrival Time Customers

In this section we will present an approach by Held, Müller, Rotter, Scheifele, Traub, and Vygen [Hel+17] which extends the classic global routing resource sharing formulation to incorporate timing constraints. Let D be the timing graph as defined in Chapter 1. As before, we assume that input ports $p \in P_{inp}$ with arrival times $\mathrm{at}(p) \in \mathbb{R}$ are given. Similarly, the output ports $p \in P_{out}$ have required arrival times $\mathrm{rat}(p)$ and timing may be propagated by $\mathrm{at}(w) = \max_{(v,w) \in E(G)} \mathrm{at}(v) + \mathrm{delay}_{v,w}$ and $\mathrm{rat}(v) := \min_{(v,w) \in E(G)} \mathrm{rat}(w) - \mathrm{delay}_{v,w}$. Our goal is to add resources and customers such that the resource usage is ≤ 1 if and only if all timing constraints are satisfied.

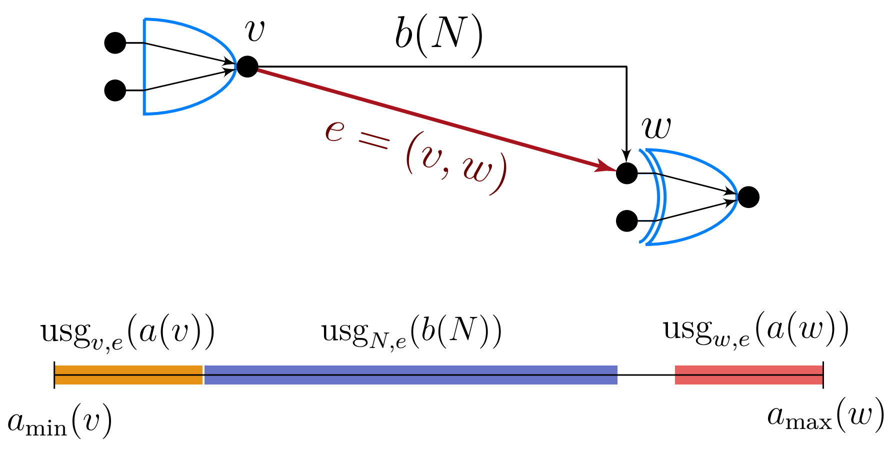


Figure 3.2: A visualization of the resource usage associated to a computed interconnect solution $b(N)$ and the choice of arrival times on the start and endpoints. By assigning a later arrival time to w we could free up usage of the resource and compensate for a slow solution $b(N)$. However, we simultaneously increase the usage of succeeding edge delay resources.

As a first step, we compute intervals of possible arrival times for all pins of the timing graph. These are given by lower and upper bounds $d_{\text{lb}} : E(D) \to \mathbb{R}_{\geq 0}$, $d_{\text{ub}} : E(D) \to \mathbb{R}_{\geq 0}$, such that $0 < d_{\text{lb}}(e) \leq \text{delay}_e(b(N)) \leq d_{\text{ub}}$. Here, $\text{delay}_e(b(N))$ denotes the delay along timing edge $e \in E(D)$ for a given Steiner tree $b(N)$ of a net $N \in \mathcal{N}$. In practice, we can use the fastest possible wire type and the best possible routing layers to compute lower bounds on the delay. As we usually do not allow arbitrary detours, upper bounds on the delay can be derived. The practical performance can be significantly improved if these lower and upper bounds are close and several ways exist to further improve them. We will not discuss the details in this thesis but refer the interested reader to [Hel+17]. By propagating these lower and upper bounds, we can compute intervals $B_v = [a_{\min}(v), a_{\max}(v)]$, for all $v \in V(D)$. Every feasible solution for all nets $b(N) \in B_N$ will correspond to choices for arrival times $a(v) \in B_v$. Note that for input ports $p \in P_{inp}$ we always have $a_{\min}(p) = a_{\max}(p) = at(p)$. Similarly, output ports $p \in P_{out}$ satisfy $a_{\min}(p) = a_{\max}(p) = rat(p)$. We extend the resource sharing formulation of the classic global routing problem by adding *arrival time customers* for every vertex in the timing graph, i.e., $\mathcal{C} = \mathcal{N} \cup V(D)$. The corresponding block of a customer $v \in V(D)$ is given by B_v. We extend the set of resources by adding a delay resource for every edge in the timing graph, i.e., $\mathcal{R} = E(G) \cup E(D) \cup \{\text{netlength}\}$. It remains to define the corresponding usage functions of a timing edge $e = (v, w) \in e(D)$. The resource will be used by the nets $N \in \mathcal{N}$ and the arrival time customers $v, w \in V(D)$. Finally, we define the following usage functions:

$$\begin{aligned}
\operatorname{usg}_{v,e}(a(v)) &:= \frac{a(v) - a_{\min}(v)}{a_{\max}(w) - a_{\min}(v)} \\
\operatorname{usg}_{N,e}(b(N)) &:= \frac{\operatorname{delay}_e(b(N))}{a_{\max}(w) - a_{\min}(v)} \\
\operatorname{usg}_{w,e}(a(w)) &:= \frac{a_{\max}(w) - a(w)}{a_{\max}(w) - a_{\min}(v)}.
\end{aligned}$$

The construction is visualized in Figure 3.2. We will now argue why this usage functions correctly translates timing constraints into resource consumption. All timing constraints can be met if and only if for all $v \in V(D)$, one can find arrival times $a(v) \in [a_{\min}(v), a_{\max}(v)]$, such that $a(v) + \operatorname{delay}_e(b(N)) \leq a(w)$ for all $e = (v, w) \in E(D)$. Recall that we fix the arrival time intervals for primary inputs and outputs. Fix an arbitrary edge $e = (v, w) \in E(D)$. The usage of the corresponding edge delay resource is satisfied if and only if the following holds.

$$\begin{aligned}
& \operatorname{usg}_{v,e}(a(v)) + \operatorname{usg}_{N,e}(b(N)) + \operatorname{usg}_{w,e}(a(w)) \leq 1 \\
\Leftrightarrow \quad & \frac{a(v) - a_{\min}(v)}{a_{\max}(w) - a_{\min}(v)} + \frac{\operatorname{delay}_e(b(N))}{a_{\max}(w) - a_{\min}(v)} + \frac{a_{\max}(w) - a(w)}{a_{\max}(w) - a_{\min}(v)} \leq 1 \\
\Leftrightarrow \quad & \operatorname{usg}_{v,e}(a(v)) + \operatorname{delay}_e(b(N)) \leq a(w)
\end{aligned}$$

This proves that the construction indeed works and a solution which does not overuse any delay resource corresponds to obeying global timing constraints. It is left to explain how these arrival times can be computed during the resource sharing algorithm, i.e., to give oracle functions. For this, notice that the weighted usage function

$$\sum_{e \in E(D)} \operatorname{price}(e) \cdot \operatorname{usg}_{v,e}(a(v))$$

is a separable function which is linear in $a(v)$. Therefore, a minimizer can be found by evaluating it at the interval borders $\{a_{\min}(v), a_{\max}(v)\}$. This approach is not very stable and arrival times can oscillate a lot. To improve stability and the rate of convergence, [Hel+17] propose to update the solution of arrival time customers more often. This does not affect the convergence guarantee of the overall resource sharing algorithm and can be done efficiently as [Hel+17] show. For the details of this update, we refer the reader to [Hel+17]. We will now show an alternate way of modeling timing resources in the resource sharing framework.

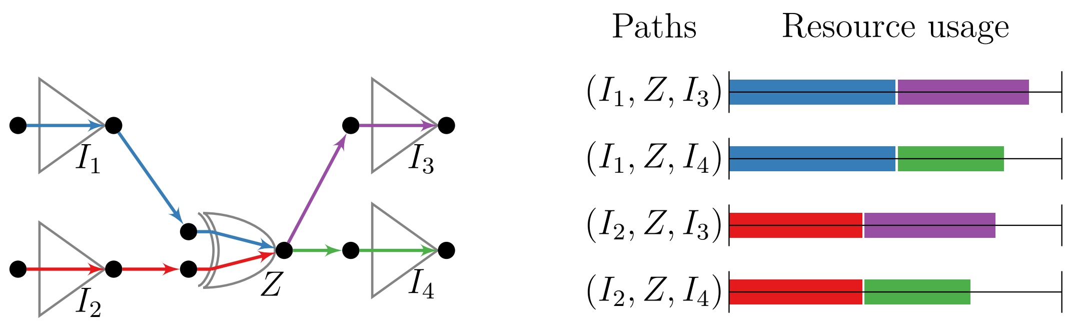


Figure 3.3: An illustration of timing path resources. The sample chip has 4 timing paths, each of which corresponds to an individual resource. The net customers which contain a given edge in a timing path will consume from the associated path resource.

3.3.3 Timing Path Resources

A natural way to model timing is to add an individual constraint for every path $P \in \mathcal{P}(D)$ in the timing graph D. Here, $\mathcal{P}(D)$ denotes the set of all inclusion-wise maximal paths in D. This directly corresponds to a resource sharing formulation, in which we introduce an additional resource for every such path. In the classic global routing problem, the resource set would be then given by $\mathcal{R} = E(G) \cup \mathcal{P}(D) \cup \{\text{netlength}\}$. For an s-t path P in the timing graph, let $e \in E(P)$ be an edge corresponding to net $N \in \mathcal{N}$, the usage of this path resource by net customer N is simply given by

$$\text{usg}_{N,P}(b(N)) = \frac{\text{delay}_e(b(N))}{\text{rat}(t) - \text{at}(s)}.$$

It is clear that in this way global timing constraints can be represented. However, at first this formulation does not look like it leads to a polynomial time algorithm. As shown in Theorem 3.2 the resource sharing algorithm can be analyzed in a way such that the dependency on the amount of resources is only logarithmic. It remains to show that we can compute timing prices for an edge in the timing graph, i.e., price(e) for $e \in E(D)$ in polynomial time. For convenience, we define $\omega_e :=$ price(e). Using the special structure of the resource sharing prices, Hähnle [Häh15] proved the following theorem. It shows that computing the prices only takes linear time.

Theorem 3.4. *(Hähnle [Häh15; Dab+18a]) The edge weights ω_e for $e \in E(D)$ can be computed in each iteration of the resource sharing algorithm in time $\mathcal{O}(|E(D)| + |V(D)|)$.*

Algorithm 3.2: Path Resource Timing Weights

```
Input: A timing graph $D$. Cumulative usages $y_e$ for $e \in E(D)$. $\gamma > 0$.
Output: Edge weights $\omega_e$ for $e \in E(D)$.
1 for $v \in P_{inp}, w \in P_{out}$ do
2     $\omega_{\mathcal{P}_{[P_{\text{inp}},v]}} = 1 \leftarrow 1, \quad \omega_{\mathcal{P}_{[w,P_{\text{out}}]}} \leftarrow 1$        ▷ Initialization step
3 for $v \in V(D)$ in topological order do
4     $\omega_{\mathcal{P}_{[P_{\text{inp}},v]}} = \sum_{(u,v)\in E} \left(e^{\gamma \cdot y_{(u,v)}} \cdot \omega_{\mathcal{P}_{[P_{\text{inp}},u]}}\right)$
5 for $v \in V(D)$ in reverse topological order do
6     $\omega_{\mathcal{P}_{[v,P_{\text{out}}]}} = \sum_{(v,w)\in E} \left(e^{\gamma \cdot y_{(v,w)}} \cdot \omega_{\mathcal{P}_{[w,P_{\text{out}}]}}\right)$
7 for $e = (v,w) \in E(D)$ do
8     $\omega_e = \exp(\gamma \cdot y_e) \cdot \omega_{\mathcal{P}_{[P_{\text{inp}},v]}} \cdot \omega_{\mathcal{P}_{[w,P_{\text{out}}]}}$        ▷ Compute $w_e$
9 return $(\omega_e)_{e\in E(D)}$
```

Proof By the definition of the resource sharing algorithm, we have

$$\omega_e = e^{\gamma \xi \widehat{\text{usg}}_{c,e}}.$$

Here, $\widehat{\text{usg}}_{c,e}$ should denote the sum of all usages of timing edge $e \in E(D)$ of previously computed solution vectors by the resource sharing algorithm. To ease notation, we write $y_e := \xi \widehat{\text{usg}}_{c,e}$. For a path, the total usage is then given as $y_P := \sum_{e\in E(P)} y_e$. We conclude that

$$\begin{aligned}
\omega_e &= \sum_{P\in\mathcal{P}: e\in E(P)} \exp(\gamma \cdot y_P) \\
&= \sum_{P\in\mathcal{P}_{[P_{\text{inp}},v]}} \sum_{Q\in\mathcal{P}_{[w,P_{\text{out}}]}} \exp\left(\gamma \cdot \sum_{f\in P\cup Q\cup\{e\}} y_f\right) \\
&= \exp(\gamma \cdot y_e) \cdot \underbrace{\left(\sum_{P\in\mathcal{P}_{[P_{\text{inp}},v]}} \exp\left(\gamma \cdot \sum_{f\in P} y_f\right)\right)}_{=: \omega_{\mathcal{P}_{[P_{\text{inp}},v]}}} \\
&\quad \cdot \underbrace{\left(\sum_{Q\in\mathcal{P}_{[w,P_{\text{out}}]}} \exp\left(\gamma \cdot \sum_{f\in Q} y_f\right)\right)}_{=: \omega_{\mathcal{P}_{[w,P_{\text{out}}]}}} \\
&= \exp(\gamma \cdot y_e) \cdot \omega_{\mathcal{P}_{[P_{\text{inp}},v]}} \cdot \omega_{\mathcal{P}_{[w,P_{\text{out}}]}},
\end{aligned}$$

where $\mathcal{P}_{[P_{\text{inp}},v]}$ denotes the set of all paths from an input vertex to v and $\mathcal{P}_{[v,P_{\text{out}}]}$ denotes the set of all paths from v to an output vertex. All variables

$\omega_{\mathcal{P}_{[P_{\text{inp}},v]}}, \omega_{\mathcal{P}_{[v,P_{\text{out}}]}}$ $(v \in V(D))$ can be computed in linear time by traversing D once in topological and once in reverse topological order. A possible implementation is presented as Algorithm 3.2.

□

3.3.4 A first comparison

In the last two sections, we presented two ways to model timing in the resource sharing framework. The obvious question is which of the solutions should be preferred.

A general answer is unlikely to exist. However each formulation has its down and upsides. Timing path resources are very natural and easy to explain. They can also have a simpler implementation as they don't require to propagate arrival time bounds and no further oracle problem has to be solved. On top of that all timing paths are simultaneously represented. This avoids situations where a single failing path worsens unrelated nets with the same endpoint. Arrival time customers do not yield a stable result without iterating their oracle, in contrast path resources work with the natural implementation.

Depending on the problem width, the theoretic runtime when using path resources may be worse. However, even if the problem width is unbounded, a different polynomial analysis can be made for path resources [Häh15]. For some algorithms, explicit arrival times may be needed. While it is also possible with the path resources to propagate fractional arrival times, these values are already available when using arrival time customers. In fact, their update mechanism may yield more stable arrival times that could improve the performance of an algorithm which relies on them.

As a first implementation we recommend timing path resources, due to their ease of implementation and the upsides pointed out above. For a detailed analysis, both variants should be tried. In Chapter 7 we will compare both possibilities for BonnRouteBuffer.

3.4 Topology Generation

Consider the sub-problem of finding a solution for single nets that minimizes the weighted resource usage. The resource sharing algorithm requires that this subproblem can at least be approximated. In practice, it is possible to find Steiner trees which are only a few percent away from the optimum length.

For the moment we assume that edge costs $c : E(G) \to \mathbb{R}_{\geq 0}$ for the global routing graph are known. Later, we will obtain these costs by applying the resource sharing algorithm. Together with the edge cost function, the global routing graph induces a metric space $(V(G), \mathrm{dist})$. By approximating the corresponding Steiner Tree problem, we could route a single net in a way that approximately minimizes the weighted cost of the used edges. We will augment this problem, by also considering a rough estimate of the arising delays. More formally, we define the topology generation problem as follows.

Problem 3: Topology Generation Problem

Input: A metric space (M, dist), a net N with pin positions $p : N \to M$, a source $s \in N$. Delay budgets $\mathrm{rat} : N \backslash \{s\} \to \mathbb{R}_{\geq 0}$.

Task: Compute a topology for N, i.e., an arborescence A rooted in s, such that T is the set of leaves. The root has exactly one successor in A and all inner vertices have outdegree exactly 2. A mapping $p' : N \to M$, such that $p'(q) = p(q)$ for $q \in N$, such that

$$\mathrm{delay}(s, q) \leq \mathrm{rat}(q) \quad \text{for } q \in N \backslash \{s\}.$$

Here, let $Y[s, q]$ be the unique $s - q$ path in A for $q \in N$. Then, the delay is defined as

$$\mathrm{delay}(s, q) := C \sum_{(v,w) \in Y[s,q]} \mathrm{dist}(p'(v), p'(w)).$$

Here $C > 0$ is a technology-dependent delay constant. The objective, which we minimize, is

$$\sum_{(v,w) \in E(A)} \mathrm{dist}(p'(v), p'(w)).$$

We also call the pair (A, p') a *placed topology.* Held and Rotter [HR13] give a bicriteria approximation algorithm, which has the following guarantee.

Theorem 3.5 (Held and Rotter [HR13])**.** *Let* (M, dist) *be a metric space,* N *a net with pin positions* $p : N \to M$*, and* $s \in N$ *a source. Let* $\mathrm{rat} : N \backslash \{s\} \to \mathbb{R}_{\geq 0}$ *be such that a solution achieving those delay budgets exists. Let* A_0, p_0 *be an initial placed topology. For each* $\epsilon > 0$ *we can compute a placed topology* (A, p')*,* *such that*

$$\text{delay}(s,q) \leq (1+\epsilon)\text{rat}(q) \quad \textit{for } q \in N\backslash\{s\}.$$

and

$$\sum_{(v,w)\in E(A)} \text{dist}(p'(v),p'(w)) \leq \left(1+\frac{2}{\epsilon}\right) \sum_{(v,w)\in E(A)} \text{dist}(p_0(v),p_0(w)).$$

The algorithm runs in time $\mathcal{O}(|N|\log|N| + \psi(A_0))$, where $\psi(A_0)$ is the time to evaluate dist *for all $e \in E(A_0)$ and* $\text{dist}(s,q)$ *for all $q \in N$.*

If the initial topology A_0, p_0 is a spanning tree, the above result also follows from a theorem of Khuller et al. [KRY95]. However, Held and Rotter further generalized the problem by showing that their algorithm yields a similar bound for a different delay function proposed by Bartoschek et al. [Bar+10], which has an additional penalty term for bifurcations. In practice this leads to solutions in which most terminals have lower depths in the topology.

The main idea of the algorithm is to traverse the initial topology A_0, p_0 and to disconnect a partial tree if the delay constraints would be violated. In a second step, these disconnected subtrees are reconnected to the source with low delay.

For solving the oracle problem of timing-constrained global routing, we would have to optimize congestion and timing costs that do not depend on each other. In this case "dist" and "delay" are independent and we have to optimize a function of the form

$$\sum_{(v,w)\in E(A)} \text{dist}(p(v),p(w)) + \sum_{t\in N} \lambda_t \text{delay}(s,t).$$

Unfortunately, this problem is very difficult. Chuzhoy et al. [Chu+08] show that if it can be approximated within $o(\log\log|N|)$, then every problem in NP can be solved in time $\mathcal{O}(n^{\log\log\log n})$. Therefore, efficient algorithms that are provably good in theory and practice are unlike to exist. As a workaround, algorithms like the previously mentioned Bicriteria by Held and Rotter can be applied heuristically to find reasonable solutions in practice.

If the delay model is allowed to have bifurcation penalties, even stronger inapproximability results are known. By a result of Hähnle and Rotter [Rot17], there is no $\mathcal{O}(|N|^{1-\beta})$ for any $\beta > 0$ unless P=NP.

An important special case arises when delay and congestion cost are linearly dependent and we have criticalities for all terminals and no bifurcation penalties are present. For this setting the currently best algorithm is due to Held and Khazraei [HK20]. It achieves a 2.39 approximation.

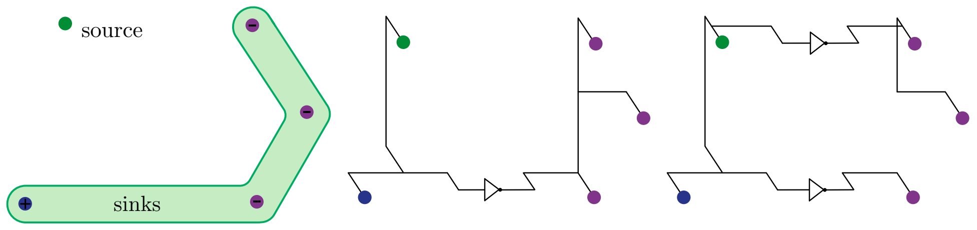


Figure 3.4: An illustration of the buffered global routing oracle problem. Given a net and prices for delay, congestion, power and placement, we are looking for a buffered Steiner tree of minimum cost. The bottom left sink has positive polarity, while the other sinks have negative polarity.

3.5 Buffering

Rotter [Rot17] extended the resource sharing formulation of the timing-constrained global routing problem to also include the buffer insertion step. In this section we will present a first look at the oracle problem he considered.

First, we will define the problem formally. Let L be a finite set of inverters and buffers. Inverters will switch the electrical polarity of a signal while buffers do not alter it. We assume that for every net $N \in \mathcal{N}$ with source $s \in N$, sink polarities $\text{pol} : N\{s\} \to \{+, -\}$ are given. A buffer solution has to insert an odd number of inverters on the s-t path for $t \in N$ if and only if $\text{pol}(t) = -$.

For buffering, placement space has to be considered as a new important resource. Rotter simply interpreted buffers as points. However, to make sure that enough space is available to accommodate the buffering solution after legalizing the placement by removing overlap he assumed that a set of placement bins B is given. A natural way to obtain placement bins is to add a bin for every global routing tile $v \in V(G)$. The usage of these bins corresponds to the bounding box size of the placed repeaters. While it works for the majority of nets, this model is not very accurate and can lead to large movements in the legalization step. We will discuss ways to improve it in Chapter 7.

The problem of computing buffered interconnect for all nets can be seen as a resource sharing problem as before. The corresponding oracle problem consists of buffering and routing a single net. As depicted in Figure 3.4 topology and buffering can significantly influence the usage of various resources.

Problem 4: Buffered Global Routing Oracle Problem

Input: A global routing graph G, a net $N \in \mathcal{N}$ with source $s \in N$, a repeater library L, wire types $\mathcal{T}$, sink polarities $\text{pol} : N\{s\} \to \{+, -\}$. The following cost functions.

- Congestion costs $c_{\text{cong}} : \mathcal{I} \times \mathcal{I} \to \mathbb{R}_{\geq 0}$.
- Delay costs $c_{\text{delay}} : N \to \mathbb{R}_{\geq 0}$.
- Placement costs $c_{\text{place}} : \mathcal{I} \to \mathbb{R}_{\geq 0}$.
- A power consumption cost $c_{\text{power}} \in \mathbb{R}_{\geq 0}$.

Task: Compute a Steiner tree (A, p) for all $N \in \mathcal{N}$ with wire type assignments $\tau_e \in \mathcal{T}$ for all $e \in E(A)$, buffer assignments $b : V(A) \to L \cup \{\varnothing\}$.

The objective is to minimize:

$$\sum_{(v,w) \in E(A)} c_{\text{cong}}(p(v), p(w))\text{usg}(p(v), p(w), \tau) + \sum_{t \in N} c_{\text{delay}}(t)\text{delay}_{A,p,b,\tau}(s, t) + \sum_{v \in V(A)} c_{\text{place}}(p(v))\text{size}(b(v)) + c_{\text{power}} \sum_{v \in V(A)} \text{power}_{A,p}(b(v))$$

Here, $\text{usg}(p(v), p(w), \tau)$ denotes the amount of wiring space the global routing edge $\{p(v), p(w)\}$ consumes from the corresponding edge resource, given wire type assignments τ. $\text{size}(b(v))$ denotes the bounding box size of an inverter $b \in L$. The function $\text{power}_{A,p,\tau}(b(v))$ computes the total power of an inverter $b(v)$, which also depends on the downstream capacitance and the fixed net switching frequency. Finally, $\text{delay}_{A,p,b,\tau}(s, t)$ is the Elmore delay on the s-t path.

The solution has to insert an odd number of inverters on the s-t path for $t \in N$, if and only if $\text{pol}(t) = -$.

We cannot hope to solve the buffered global routing oracle problem for general fanouts. Even if we do not allow buffering, i.e., $L = \varnothing$ and under the simplifying assumption that delays are linear, the problem remains difficult. As this generalizes the problem given in Section 3.4, we can apply the result of Chuzhoy et al. [Chu+08] to see that a $o(\log\log |N|)$ approximation implies that every problem in NP can be solved in time $\mathcal{O}(n^{\log\log\log n})$. However, if we assume that the Steiner tree (A, p) and its wire type assignment τ are fixed, and we only look for the assignment of buffers $b : V(A) \to L \cup \{\varnothing\}$, the problem can be solved via dynamic programming. As described by Van Ginneken [Van90] one can proceed from the sinks in a bottom-up fashion and prune dominated candidates. Bartoschek et al. [Bar+09] show that even local topology changes can be allowed, improving compared to algorithms which do not alter the input tree. We will extend an algorithm of Rotter [Rot17] to solve the oracle problem heuristically in Chapter 7. The main idea is to proceed in multiple stages. First, Rotter computes a 2d topology with small linear delay, which is subsequently embedded into the routing graph and then buffered. It is expected that this approach will fail for a small fraction of the nets. If blockages are present on the chip, approaches which first compute the topology and then buffer can lead to infeasible solutions. Rotter addresses this problem by applying a slower multi-label algorithm implementation by Natura [Nat17], which simultaneously buffers and routes the most difficult nets. It is significantly slower compared to the multi-stage approach, but necessary to avoid electrical violations.

Chapter 4

Time-Cost Tradeoff Problems in Chip Design

In the last chapter, we presented multiple resource sharing problems that occur in chip design. In the next two chapters, we will focus on a minimal but very important resource sharing problem in the design process. It arises when only two types of resources, namely timing resources and a single cost resource are considered. Most parts of this chapter have been published in [Dab+18b], which is joint work with Stephan Held, Jens Vygen and Sonja Wittke.

Let d be an upper bound on the number of gates on any path in the timing graph. Previously, Wittke [Wit11] devised a d approximation for the V_t optimization problem by using the primal-dual Bar-Yehuda and Even [BE81] algorithm. In her problem formulation and her practical implementation she required a separable delay function, i.e., no interaction between gates on a path is allowed.

Our main contributions in this chapter are as follows. We present a new implementation and proof for the Bar-Yehuda and Even algorithm [BE81] applied to V_t optimization. Unlike previous work by Wittke [Wit11], neither our proof nor the new implementation requires a separable delay function. Instead, we show that two mild assumptions are sufficient which mostly hold for non-linear delay models used in practice. Our refined implementation is built directly around an industrial sign-off timer. Even when applied after one of the most successful algorithms for gate sizing and V_t assignment [RSR16a] we observe large leakage reductions of up to 8%. After global routing the reduction grows up to 34% without changing any footprints. Our algorithm is extended to also compute lower bounds which prove that we solve several instances almost optimally and others much better than the worst case guarantee.

On a small level, all combinatorial cells used on a chip consist of transistors. Modern CPUs usually contain more than 10^9 of these tiny devices. Every transistor has three important electrical contacts. When the voltage measured between gate and drain contact surpasses a certain level $V_t > 0$, a current may flow from the source to the drain. This voltage V_t is also called *threshold voltage*. For a transistor, multiple possible realizations with different threshold voltages

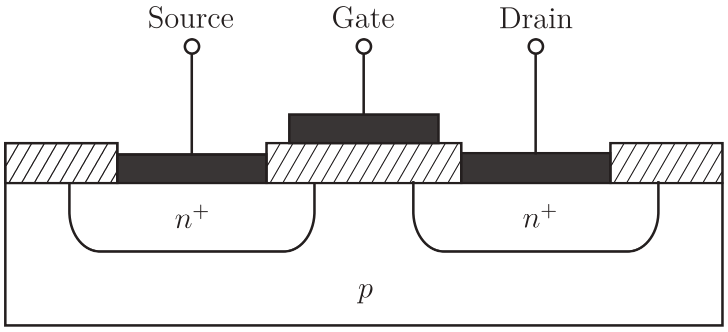


Figure 4.1: A cross-sectional view of a field-effect transistor. If the voltage between gate and drain is sufficiently high, a current may flow from source to drain. By changing the substrate used in manufacturing, delay properties can be changed at the cost of power consumption.

are available. However, there is a tradeoff between performance and power consumption. As different V_t levels require a separate manufacturing process, only a small amount of 2–4 alternative V_t levels is used. In Section 2.5 we saw that while the size of a gate which has an approximately linear influence on the power, the static power consumption usually depends exponentially on the used V_t level of a gate. The large differences in the leakage power consumption of the individual threshold implementations and the discrete nature of the problem make this task very challenging in theory and practice.

4.1 Previous Work

Previous approaches to threshold voltage (V_t) optimization are often combined with simultaneous sizing as in recent sensitivity-based [Hu+12; Kah+13] or Lagrangian relaxation approaches [Fla+14; RSR16a], among which [Fla+14] reported the best results on the ISPD 2012 and 2013 gate sizing contest benchmarks. Its integration into an industrial design environment [RSR15; RSR16a] also achieved substantial power reductions on industrial designs. However, threshold voltage optimization finds its individual application in post-routing power reduction [Abr+11; RS12]. Most modern cell libraries offer different V_t choices with the same footprint. Thus, the voltage threshold of a gate can be changed without requiring routing changes.

Shah et al. [Sha+05] propose a continuous formulation for simultaneous gate sizing and V_t assignment, in which the V_t levels are always snapping to integral values. However, the relaxation is not convex and is not known to be efficiently solvable with any useful approximation guarantee.

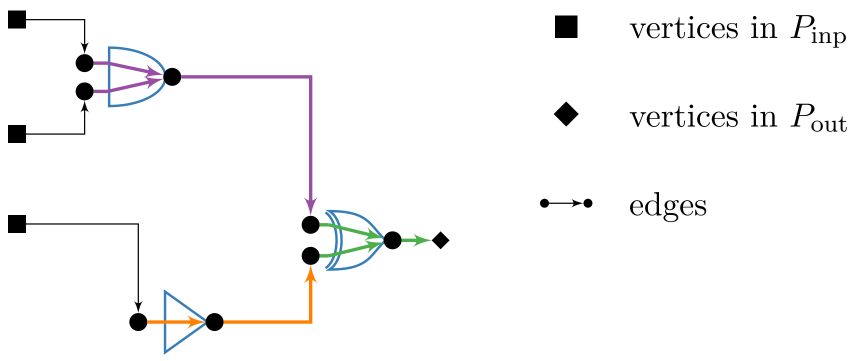


Figure 4.2: An example of a time-cost tradeoff instance in the timing graph. Edges sets E_g are depicted in the same color for $g \in \mathcal{G}$.

Liu and Hu [LH10] combine Lagrangian relaxation with dynamic programming for rounding to a discrete solution, resolving inconsistencies due to reconvergent paths heuristically. This approach was later refined in by Ozdal, Burns, and Hu[OBH12].

Algorithms for pure threshold voltage optimization include the conjugate gradient method applied to a certain continuous problem relaxation [Abr+11] or greedy algorithms [RS12].

The problem of choosing a V_t level for each gate while satisfying the timing constraints is similar to the discrete time-cost tradeoff (TCT) problem in directed graphs. Here, we are given an acyclic digraph where every vertex has a set of possible execution times with associated costs. The task is to choose a realization for every vertex such that the maximum execution time of a path respects some delay bound and the total cost is minimized. We will not go into more details here, as an extended analysis of this problem is presented in the next chapter.

The remainder of this chapter is organized as follows. In Section 4.2, we give a formal problem definition. Then in Section 4.3, we present the new approximation algorithm together with an example and a theoretical quality analysis. In Section 4.4, we shortly present further variants of the problem and algorithm, and a four-step flow for V_t assignment. Finally, we present experimental results in Section 4.5.

4.2 Problem Formulation

As explained in Chapter 2, we assume that the timing can be modeled as a directed acyclic graph $D = (V, E)$ (the timing graph), a set of gates $\mathcal{G}$, and a set of V_t levels $\{1, \ldots, z\}$. As before, we call vertices $v \in V$ without entering edges input vertices $v \in P_{\text{inp}}$; analogously we define output vertices

P_{out}. Each gate $g \in \mathcal{G}$ is represented in D by a subset E_g of edges of D where $E_g \cap E_{g'} = \varnothing$ for any pair of different gates $g, g' \in \mathcal{G}$. A sample timing graph is shown in Figure 4.2. For easier notation we assume that every gate has some implementation for each V_t level in $\{1, \ldots, z\}$. Here, 1 is the fastest V_t level (lowest V_t) and z the slowest one (highest V_t).

A V_t assignment is a map $\varphi : \mathcal{G} \to \{1, \ldots, z\}$. We denote the assignment that maps every gate to the fastest implementation by $\mathbb{1}$. The delay of a timing edge $e \in E(D)$ depends on the assignment φ and is denoted by $d_\varphi : E \to \mathbb{R}_{\geq 0}$. For a path P in D we define $d_\varphi(P) = \sum_{e \in E(P)} d_\varphi(e)$. The power of a gate is given by a function $\text{power}(g, i)$, where $i \in \{1, \ldots, z\}$ specifies the chosen V_t level.

For a path P, its delay bound T, and assignment φ, the slack of a path P is defined by $\text{slack}(P, \varphi) = T - d_\varphi(P)$. For pins $v \in V$ and edges $e \in E$ we denote by $\text{slack}(v, \varphi)$ and $\text{slack}(e, \varphi)$ the minimum slack of a path that contains v and e, respectively. With this definition, the total negative slack (TNS) defined in Section 2.6 can be written as

$$\text{TNS}(\varphi) = \sum_{v \in P_{\text{out}}} \min\{0, \text{slack}(v, \varphi)\}.$$

We consider the following V_t optimization problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{g \in \mathcal{G}} \text{power}(g, \varphi(g)) \\ \text{subject to} \quad & \varphi : \mathcal{G} \to \{1, \ldots, z\} \\ & \text{TNS}(\varphi) = \text{TNS}(\mathbb{1}). \end{aligned} \tag{4.1}$$

We will also discuss some variants of this problem in Section 4.4.

4.3 V_t Optimization Algorithm

To devise an approximation algorithm, we make the following observation. In chip design, there is usually an upper bound $d \in \mathbb{N}$ on the amount of gates in any path. This is due to a target frequency which cannot be reached if too many gates are used.

We will now describe our proposed V_t optimization algorithm (Algorithm 4.1). We start by assigning every gate $g \in \mathcal{G}$ to the highest available V_t level $\varphi(g) = z$. Over the course of the algorithm we will maintain a reduced cost function c_g which is guiding the optimization globally. Initially, c_g is given by the additional cost needed to accelerate gate g to the next lower V_t level

$$c_g \;=\; \text{power}(g, z-1) - \text{power}(g, z).$$

Algorithm 4.1: V_t Optimization Algorithm

```
1  for $g \in \mathcal{G}$ do
2      $\varphi(g) \leftarrow z$                                   ▷ initially use slowest $V_t$
3      $c_g \leftarrow \text{power}(g, z-1) - \text{power}(g, z)$   ▷ reduced costs
4  while $\exists v \in P_{\text{out}} : \text{slack}(v, \varphi) < \min\{0, \text{slack}(v, \mathbb{1})\}$ do
5      $P \leftarrow$ most critical path ending in $v$ w.r.t. $\varphi$
6      $\mathcal{G}(P) \leftarrow$ gates on timing path $P$
7      $g^* \leftarrow \text{argmin}_{g \in G_P} c_g$               ▷ cheapest $g$ w.r.t. $c_g$
8      $\gamma \leftarrow c_{g^*}$
9      for $g \in \mathcal{G}(P)$ do
10         $c_g \leftarrow c_g - \gamma$                             ▷ reduce $c_g$ for gates on $P$
11     $\varphi(g^*) \leftarrow \varphi(g^*) - 1$                  ▷ accelerate $g^*$
12     if $\varphi(g^*) > 1$ then
13         $c_{g^*} \leftarrow \text{power}(g^*, \varphi(g^*) - 1) - \text{power}(g^*, \varphi(g^*))$
14                                                                ▷ re-initialize $c_{g^*}$
15     else
16         $c_{g^*} \leftarrow \infty$
17 return $\varphi$
```

Then we proceed to iteratively accelerate a path P that violates the timing constraints. We do this by accelerating the cheapest gate g^* on P with respect to the reduced cost function c_g. Note that this accelerates all paths through g, not only P. We then reduce the values c_g for every gate on P by exactly c_{g^*}. This process is iterated until the timing constraints are met.

Note that our particular reduced cost update is important for achieving globally good solutions. If a gate occurs frequently on some violated path P, it becomes more attractive to be accelerated due to its lowered cost. The cost update is a core ingredient for proving the approximation guarantee of our algorithm in Section 4.3.2, and also needed for good practical results as the following example demonstrates.

4.3.1 Example

Consider the instance in Figure 4.3, where an inverter drives $K \gg 1$ other inverters (the instance can be easily adjusted to avoid high fanouts using a higher depth). If the deadline is $T = 3$ and the inverters have either delay 1 or 2 with an acceleration cost of 1, the optimum is given by only accelerating the driving inverter.

Already for $d = 2$ a simple greedy approach that sorts all acceleration possibilities of the gates by the gain $\frac{\Delta\text{delay}}{\Delta\text{power}}$ and iteratively accelerates a gate that minimizes this (negative) ratio, e.g. a discrete variant of the TILOS

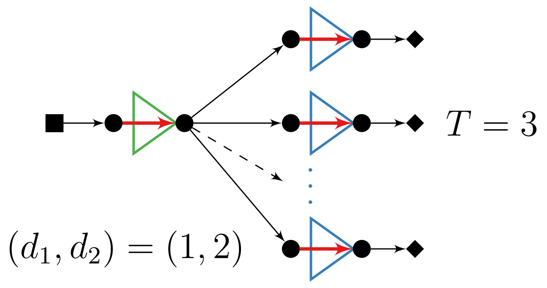


Figure 4.3: An example where a greedy algorithm might speed up all K gates on the right, while our primal-dual algorithm chooses the left gate and at most one gate from the right side. (see Section 4.3.1)

algorithm [FD85], does not achieve a constant approximation guarantee. It may instead choose to accelerate all K inverters instead (in fact it will always do so if we reduce their acceleration cost by some small constant $\epsilon > 0$). Similarly, always choosing the cheapest gate on a critical path (without our reduced cost update) will lead to the same behaviour.

Our primal-dual algorithm might also start accelerating one of the inverters on the right. However, this reduces the cost of the driving inverter on the common path to 0 (or ϵ). This one will, thus, be the cheapest choice in the next iteration, and our algorithm will accelerate at most two inverters in total.

4.3.2 Algorithm Analysis

Before we analyse the algorithm, we point out the theoretic connection to the set cover problem. In the set cover problem we are given a set system $\mathcal{S} = \{S_1, \ldots, S_r\}$ where $\cup_{i=1}^r S_i =: \mathcal{U}$ and a cost function $\text{cost} : \mathcal{S} \to \mathbb{R}_{\geq 0}$. We are then looking for a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that $\cup_{S \in \mathcal{S}'} S = \mathcal{U}$ and $\sum_{S \in \mathcal{S}'} \text{cost}(S)$ is minimized.

If we assume that our instance has only two V_t levels, that is $z = 2$, we can formulate it as a set cover problem in the following way. We call a set of gates $G \subseteq \mathcal{G}$ critical if in every timing feasible solution at least one gate in G has the lower V_t level. Let $\mathcal{P}$ denote the set of all paths from an input vertex to an output vertex. For a path $P \in \mathcal{P}$ we denote by $\mathcal{G}(P)$ the gates that have some edge on P. Our universe will then be $\mathcal{U} = \{G \subseteq \mathcal{G}(P) : P \in \mathcal{P}, G \text{ is critical}\}$. For every gate $g \in \mathcal{G}$ we define $S_g = \{G \subseteq \mathcal{G} : g \in G, G \in \mathcal{U}\}$.

It is easy to see that for a set $X \subseteq \mathcal{G}$ we have $\cup_{g \in X} S_g = \mathcal{U}$ if and only if accelerating the gates in X yields a timing feasible solution. In the special case $z = 2$, Algorithm 4.1 is an adaptation of the primal-dual algorithm of Bar-Yehuda and Even [BE81] for the set cover problem. The algorithm has an approximation guarantee of $\max_{u \in \mathcal{U}} |\{S \in \mathcal{S} : u \in S\} \leq \max_{P \in \mathcal{P}} |\mathcal{G}(P)|$. We will now give an elementary proof of this bound for arbitrary z.

To prove quality guarantees of our new algorithm, we make two mild assumptions:

A1 Lowering the voltage threshold of a gate does not increase the delay of any edge in E.

A2 The delay $d_\varphi(P)$ of a path P can only be reduced by lowering $\varphi(g)$ for a gate $g \in \mathcal{G}(P)$.

The first assumption is usually fulfilled if the input pin capacitances of a gate g do not depend on its voltage threshold $\varphi(g)$. The second assumption would follow from the first assumption in a path-based timing analysis. In any case, deviations from these assumptions in practice are usually small. We can prove the following worst-case guarantee.

Theorem 4.1. *Assume that A1 and A2 hold. Algorithm 4.1 returns a feasible solution $\bar{\varphi}$ to Problem (4.1).*

The power increase over the cheapest possible solution, choosing z everywhere, is at most d times greater than the power increase of an optimum solution φ^*:

$$\begin{aligned}\sum_{g\in\mathcal{G}} \Big(\text{power}(g,\bar{\varphi}(g)) - \text{power}(g,z)\Big) \\ \le d \sum_{g\in\mathcal{G}} \Big(\text{power}(g,\varphi^*(g)) - \text{power}(g,z)\Big),\end{aligned}$$

where d is the maximum number of gates on any path in the timing graph D.

The algorithm can be implemented to run in time $\mathcal{O}(z|\mathcal{G}|\theta)$, where θ is the running time for identifying and traversing a critical path.

Proof Obviously, the algorithm stops only when φ is a feasible solution. It stops after at most $(z-1)|\mathcal{G}|$ iterations of the while loop, proving the total running time bound.

Let $\mathcal{U} := \{(P,\varphi) : P$ is a path from $v \in P_{\text{inp}}$ to $w \in P_{\text{out}}$ with $\text{slack}(P,\varphi) < \min\{0, \text{slack}(w,\mathbb{1})\}\}$ be the set of pairs with path P and V_t assignment φ for which P is too slow. Suppose we add $y(P,\varphi) := 0$ for all $(P,\varphi) \in \mathcal{U}$ in the initialization (before line 4), and $y(P,\varphi) := \gamma$ before line 11 of the algorithm (for the current values of P, φ, and γ). These numbers are needed only for the following analysis.

For any gate $g \in \mathcal{G}$ and any $i \in \{2,\ldots,z\}$ we have, while $\varphi(g) = i$, the invariant

$$c_g + \sum_{(P,\widehat{\varphi})\in\mathcal{U}: g\in\mathcal{G}(P), \widehat{\varphi}(g)=i} y(P,\widehat{\varphi}) \;=\; \text{power}(g,i-1) - \text{power}(g,i). \tag{4.2}$$

Moreover, we have $c_g \ge 0$ at any stage, and $c_g = 0$ when $g = g^*$ is accelerated in Line 11. Let $\bar{\varphi}$ denote the output of the algorithm. At termination, we have for $g \in \mathcal{G}$ with $\bar{\varphi}(g) < i$ the property

$$\sum_{(P,\widehat{\varphi})\in\mathcal{U}:g\in\mathcal{G}(P),\widehat{\varphi}(g)=i} y(P,\widehat{\varphi}) \;=\; \operatorname{power}(g,i-1) - \operatorname{power}(g,i). \tag{4.3}$$

Let φ^* be an optimum solution. By definition, and by assumptions A1 and A2, for every $(P,\widehat{\varphi}) \in \mathcal{U}$ there exists a $g \in \mathcal{G}(P)$ with

$$\varphi^*(g) \;<\; \widehat{\varphi}(g). \tag{4.4}$$

Using equations (4.2), (4.3) and (4.4) we conclude:

$$\begin{aligned}
&\sum_{g\in\mathcal{G}}\Big(\operatorname{power}(g,\bar{\varphi}(g)) - \operatorname{power}(g,z)\Big)\\
&= \sum_{g\in\mathcal{G}}\sum_{i=\bar{\varphi}(g)+1}^{z}\Big(\operatorname{power}(g,i-1) - \operatorname{power}(g,i)\Big)\\
&\overset{(4.3)}{=} \sum_{g\in\mathcal{G}}\sum_{i=\bar{\varphi}(g)+1}^{z}\;\sum_{(P,\widehat{\varphi})\in\mathcal{U}:g\in\mathcal{G}(P),\widehat{\varphi}(g)=i} y(P,\widehat{\varphi})\\
&= \sum_{g\in\mathcal{G}}\;\sum_{(P,\widehat{\varphi})\in\mathcal{U}:g\in\mathcal{G}(P),\widehat{\varphi}(g)>\bar{\varphi}(g)} y(P,\widehat{\varphi})\\
&\overset{(4.5)}{\leq} \sum_{(P,\widehat{\varphi})\in\mathcal{U}} y(P,\widehat{\varphi})\;|\mathcal{G}(P)|\\
&\overset{(4.6)}{\leq} d\sum_{(P,\widehat{\varphi})\in\mathcal{U}} y(P,\widehat{\varphi})\\
&\overset{(4.4)}{\leq} d\sum_{(P,\widehat{\varphi})\in\mathcal{U}} y(P,\widehat{\varphi})\;|\{g\in\mathcal{G}(P) : \varphi^*(g) < \widehat{\varphi}(g)\}|\\
&= d\sum_{g\in\mathcal{G}}\;\sum_{(P,\widehat{\varphi})\in\mathcal{U}:g\in\mathcal{G}(P),\varphi^*(g)<\widehat{\varphi}(g)} y(P,\widehat{\varphi})\\
&= d\sum_{g\in\mathcal{G}}\sum_{i=\varphi^*(g)+1}^{z}\;\sum_{(P,\widehat{\varphi})\in\mathcal{U}:g\in\mathcal{G}(P),\widehat{\varphi}(g)=i} y(P,\widehat{\varphi})\\
&\overset{(4.2)}{\leq} d\sum_{g\in\mathcal{G}}\sum_{i=\varphi^*(g)+1}^{z}\Big(\operatorname{power}(g,i-1) - \operatorname{power}(g,i)\Big)\\
&= d\sum_{g\in\mathcal{G}}\Big(\operatorname{power}(g,\varphi^*(g)) - \operatorname{power}(g,z)\Big).
\end{aligned}$$

In (4.5) we use that the contribution to y of an element $(P,\widehat{\varphi}) \in \mathcal{U}$ can be counted at most once for every gate $g \in \mathcal{G}(P)$, which are $|\mathcal{G}(P)|$ many. For (4.6) note that every path contains at most d gates. □

The theorem obviously implies the following bound on the total power consumption.

Corollary 4.2. *Assume that A1 and A2 hold. The solution $\bar{\varphi}$ of Algorithm 4.1 exceeds the power of an optimum solution φ^* by at most a factor d:*

$$\sum_{g\in\mathcal{G}} \text{power}(g,\bar{\varphi}(g)) \leq d \sum_{g\in\mathcal{G}} \text{power}(g,\varphi^*(g)),$$

where k is the maximum number of gates on any path in G.

Note that in our setting the number of gates on a longest path is a small constant compared to the total number of gates. The running time θ is usually linear in $|E|$, but Algorithm 4.1 can as well be used with a path-based timing analysis. See also Sections 4.4.6 and 4.5.1 for practical running time reductions.

Algorithm 4.1 also provides a lower bound on the optimum power consumption. As we will see this bound is much tighter than k in practice (cf. Section 4.5).

Corollary 4.3. *Assume that A1 and A2 hold. Let φ^* be an optimum solution. Let $y(P,\varphi)$ be defined as in the proof of Theorem 4.1. At any point of Algorithm 4.1 we have*

$$\sum_{g\in\mathcal{G}} \text{power}(g,\varphi^*(g)) \geq \sum_{(P,\varphi)\in\mathcal{U}} y(P,\varphi) + \sum_{g\in\mathcal{G}} \text{power}(g,z).$$

Proof This is part of the inequality chain (in particular the last two inequalities) in the proof of Theorem 4.1. □

The lower bound can easily be computed by summing up the reduced costs of all accelerated gates and adding the high VT power. As the y values are non-decreasing over the course of the algorithm, they determine a lower bound at every intermediate step.

Note that our algorithm guarantees a close to optimum solution for low power designs with small depth.

Consider an instance with small d, where the optimum solution φ^* uses only slightly more power than the solution φ^z, which assigns every gate to the slowest available realization.

More precisely, assume

$$\sum_{g\in\mathcal{G}} \text{power}(g,\varphi^*(g)) \leq (1+\epsilon) \sum_{g\in\mathcal{G}} \text{power}(g,\varphi^z(g))$$

for some $\epsilon > 0$. By considering a modified instance where φ^z has cost 0 one can easily see that our algorithm will return a solution $\bar{\varphi}$ such that

$$\sum_{g\in\mathcal{G}} \text{power}(g,\bar{\varphi}(g)) \leq (1+d\epsilon) \sum_{g\in\mathcal{G}} \text{power}(g,\varphi^z(g)).$$

The approximation ratio that we obtain is therefore given by

$$\rho \leq \frac{1 + d\epsilon}{1 + \epsilon}.$$

If we have $d = 10$ and there is an optimum solution that uses 1% more power than φ^z, i.e. $\epsilon = 0.01$ we are guaranteed to obtain a $\frac{1.1}{1.01} \approx 1.089$ approximation.

We point out that the primal-dual cost update is essential to obtain a good approximation guarantee. Other algorithms that greedily accelerate the critical path do usually not give any guarantee. In the example in Section 4.3.1, our algorithm accelerates at most $k = 2$ inverters, as Theorem 4.1 guarantees.

4.3.3 Sharpness Of The Analysis

It can be seen that our analysis in Theorem 4.1 is tight. Indeed, suppose we have an inverter chain of $d + 1$ gates with cycle time $T = 1$, where all gates have delay 0 for low V_t and power 0 for high V_t, the first gate has power $1 + \epsilon$ for low V_t, and delay 1 for high V_t, and the other gates have power 1 for low V_t and delay $\frac{1}{d}$ for high V_t. The algorithm will put all but the first gate on low V_t and spend power d, while the optimum with power $1 + \epsilon$ is exactly the opposite. However, the cell library assumed in this example has unrealistically varying delay power tradeoffs for the different gates.

4.4 Variants and Implementation

In the following, we discuss several enhancements to improve the applicability on industrial designs.

4.4.1 Handling Critical Subpaths

Algorithm 4.1 finds a solution that maximizes the TNS while approximating the minimum power consumption. Assume we are given an instance with two inverter chains of length 1 and one inverter chain of length 3 that all share a common output vertex $t \in P_{\text{out}}$. If we have a cycle time of $T = 1$ and all inverters have fast delay 1 and slow delay 2, the optimum solution will completely accelerate the long path with three inverters as the optimum attainable TNS is -2. This instance is depicted in Figure 4.4. Note that the critical inverters on the short paths are not accelerated, even though doing so would remove the timing violation on the corresponding paths.

This shows that one should consider a more accurate timing metric than TNS. Reimann et al. suggested to use the so-called TTNS (true TNS) to evaluate the timing on less critical subpaths [RSR16b]. The TTNS is maximized if every

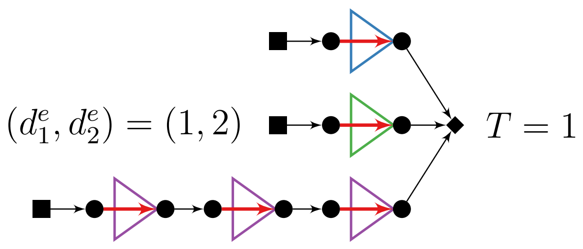


Figure 4.4: Situation in which some paths are not optimized due to a hopeless path. All gates have a fast delay of $d_1^e = 1$ and a slow delay of $d_2^e = 2$.

path with negative slack is as fast as possible. Further details on the TTNS are explained in Chapter 2.6.

Our algorithm can be extended to find a solution with $\text{TTNS}(\varphi) = \text{TTNS}(\mathbb{1})$ by changing the condition of the while loop in line 4 to look for an edge $e \in E$ for which $\text{slack}(e, \varphi) < \min\{0, \text{slack}(e, \mathbb{1})\}$ and selecting a most critical path through that edge in line 5. It is straightforward to prove that this is also a k-approximation in terms of a cheapest solution in which every negative path is as fast as possible. We can also stop once $\text{TTNS}(\varphi) \geq \Theta$ for a given threshold $\Theta \leq \text{TTNS}(\mathbb{1})$. In our experiments, we chose Θ as the TTNS of the initial solution, for which we want to improve the leakage power.

4.4.2 Power Recovery

Once Algorithm 4.1 terminates some gates can usually be decelerated again without introducing timing violations. For example, a fanout inverter on the right side in Figure 4.3 can be decelerated after the driving inverter on the left has been accelerated. For the Bar-Yehuda and Even algorithm the so-called reverse delete step [GW96] serves this purpose. In this post-processing routine the gates are considered in the reverse order in which they were accelerated by the algorithm and decelerated if this does not introduce any timing violation.

Alternatively, it is also tempting to decelerate gates in non-increasing order of their static leakage. As this order experimentally led to better leakage reductions, we incorporated it to post-process the result of Algorithm 4.1.

In contrast to the greedy approaches described in Section 4.3.1, Algorithm 4.1 together with the recovery step solves the instance in Figure 4.3 optimally.

4.4.3 Breaking Ties

By A1 we assume that lowering the voltage threshold does not increase the delay of any edge. We verify this by measuring the slack gain after every acceleration and rejecting the change if the path slack degraded. This hardly ever occurs but we also use the slack change to break ties if there are multiple gates with the same reduced cost. To avoid excessive runtime increases we configure the signoff timer to only update delays and slacks locally for these slack change evaluations.

4.4.4 Slew Violation Removal

Extensive V_t optimization may lead to violations of slew limits at some sink pins. We neglect these violations during the course of Algorithm 4.1 and the power recovery step in the previous section.

However, at the very end we eliminate slew limit violations by lowering V_t levels or, only if the footprint does not need to be preserved, by changing gate sizes. In our experiments, this affects a negligible fraction of the gates.

4.4.5 Preprocessing

In practice, instances often contain hopeless paths where all gates have to be assigned to the lowest V_t level. We can significantly speed up our algorithm by identifying and accelerating these paths in a preprocessing step.

For the TTNS optimization we set all gates to the fastest alternative and fix this lowest V_t level for all gates whose slack is still negative. It is easy to see that such gates need to be set to the fastest alternative in every feasible solution.

For the case of maximizing the TNS this approach can pre-assign more gates to low V_t than necessary. For instance in Figure 4.4, the upper two gates do not influence the TNS. In this case we can only fix gates whose deceleration would decrease the TNS. We can identify these gates by propagating slack deltas in reverse topological order.

4.4.6 Disjoint Paths for Running Time Reduction

We can obtain a significant practical speedup by considering not only a single most critical path but a set of gate-disjoint critical paths independently. In every iteration we compute a set of disjoint violated P_{inp}-P_{out}-paths and for each of these paths we accelerate exactly one gate with the minimum reduced cost. This reduces the number of iterations and the number of global timing updates before collecting the path(s), leading to a significant running time reduction.

We collect these paths by traversing the timing graph in reverse topological order while blocking gates that already occur in some path. Note that the bound in Theorem 4.1 still holds, because in the proof we do not use that Algorithm 4.1 selects a most critical path. It is sufficient to pick any violated path.

However as we will see later (Section 4.5), in practice the leakage might degrade a little when using too many paths simultaneously. Thus, we only select a fraction of the most critical paths. To this end we use a sliding slack window that selects a subset of the timing endpoints. Initially, we select all timing endpoints which are within $r := 1\,\text{ps}$ from the global worst slack.

To always select a good portion of failing paths, we adjust the window as follows. If we selected less than $\frac{\alpha \cdot |\mathcal{G}'|}{1000}$ paths in an iteration, where $\mathcal{G}' \subseteq \mathcal{G}$ is the set of gates with negative slack at the start of the algorithm and $\alpha > 0$ is a parameter, we increase $r \leftarrow 1.15r$. Otherwise, we set $r \leftarrow 1.15^{-1}r$. Due to the multiplicative update of r the slack window will quickly be large enough to select a sufficient amount of paths. Therefore the exact choice of r or the search factor of 1.15 do not play a large role with respect to the measured runtime.

For our experiments we used $\alpha = 1$ unless stated otherwise. In Section 4.5.1 we analyse the dependency on α experimentally.

4.4.7 Overall V_t Assignment Flow

Sometimes we do not want to obtain the best possible TTNS. Instead, the goal is to achieve the quality of the initial solution φ^I using less power. In any case we always maximize the worst slack and the TNS. This can be achieved by multiple calls to slight variants of Algorithm 4.1, which are described in our overall optimization flow in Algorithm 4.2.

Algorithm 4.2: Optimization Flow

```
1 Run Algorithm 4.1 until the TNS is maximized.
2 Run the variant of Algorithm 4.1 described in Section 4.4.1, but
    without initialization (lines 1–3 of Algorithm 4.1), and stop as soon as
    $\text{TTNS}(\varphi) \geq \text{TTNS}(\varphi^I)$. If after accelerating a gate the initial leakage
    power $\sum_{g \in \mathcal{G}} \text{power}(g, \varphi^I(g))$ is exceeded we stop the algorithm.
3 Power recovery (Section 4.4.2).
4 Slew violations removal (Section 4.4.4), and (when sizing is allowed)
    placement legalization.
```

4.5 Experimental Results

We evaluated our implementation of Algorithms 4.1 and 4.2 on the industrial 22nm microprocessor instances that were also used in [RSR16a]. For these instances, we can use one of the most successful algorithms by Reimann et al. [RSR16a] for initial gate sizing and V_t assignment, and measure the additional leakage power reduction achieved by our algorithm.

Our implementation is integrated into the IBM microprocessor design flow. For every instance $z = 3$ V_t levels were available. We used the sign-off timing engine EinsTimer for all timing calculations inside our algorithm and for the numbers in the tables. Wire delays were estimated using the MAISE delay model [LF08], which is the default in the design environment. Here we apply our algorithm on a highly optimized netlist. First, we use the gate sizing algorithm by Reimann et al. [RSR16a] to reduce the static power consumption by up to 10% and the total power by up to 8.3%. Then, we are able to obtain additional static power reductions of up to 8% by using our algorithm. The experiments were performed on a cluster of Linux servers with Intel Xeon CPUs with clock frequencies between 2.6 and 3.4 GHz.

Unfortunately, only the instances from [RSR16a] at the beginning of the physical design flow were available to us, but not the final gate sizing instances used in [RSR16a]. Thus, we reran the physical design flow, which is the reason why our numbers slightly deviate from [RSR16a]. Note that Reiman et al. [RSR16a] also reran the whole flow compared to their previous work [RSR15].

We also report the lower bound $P^{\text{lb}}_{\text{static}}$ on the minimum leakage for TNS maximization, which is computed according to Corollary 4.3. Algorithm 4.2 stops once the initial leakage power or TTNS are reached, way before the TTNS is maximized. At this point the bound induced by the y-variables is not valid for the current TTNS, but only for the maximum TTNS. Thus, we always report the valid bound for TNS maximization.

We ran three variants of our algorithm. We disabled the preprocessing step as we stop the optimization as soon as the initial power is exceeded, which is not possible when a preprocessing is used. The results are given in Table 4.2. The rows of Table 4.2 refer to the following experiments/flows:

- Initial: An unrouted industrial instance after placement, and full timing optimization including a net-based layer, wiring width, and spacing assignment. Wires are estimated as short Steiner trees on the respective layers assuming the assigned width and spacing. The snapshot is the result of an industrial design flow.

- Lagrange [RSR16a]: We use the implementation of the gate sizing and V_t assignment algorithm by Reimann et al. [RSR16a], which is integrated into the industrial design environment. Note that [RSR16a] is an industrial adaption of [Fla+14], the winner of the ISPD'13 contest. We observe similar power improvements and running times as the original paper [RSR16a]. While they report a running time of about 13 hours for the largest instance uP_10 with 126k gates we measured around 11.5 hours [RSR16a].

- Alg. 4.2 TNS-opt: We apply our optimization flow (Algorithm 4.2) on the result of the Lagrange flow but omit the second step which optimizes the TTNS. The purpose of this step is primarily to serve for comparison with the lower leakage bound $P^{\text{lb}}_{\text{static}}$.

- Alg. 4.2: We apply our optimization flow (Algorithm 4.2) on the result of the Lagrange flow.
- Alg. 4.2 post-GR: We apply our optimization flow (Algorithm 4.2) on the result of the Lagrange flow followed by an industrial timing-driven global routing. Here, gate sizing is forbidden for slew recovery to preserve gate footprints.

The columns show:

- the instance names,
- the particular flow,
- the number of gates $|\mathcal{G}|$,
- the maximum number d of gates on a signal path,
- the worst slack WS,
- the total negative endpoint slack TNS,
- the true total negative slack TTNS [RSR16b],
- the leakage power consumption $P^{\text{apx}}_{\text{static}}$ before power recovery,
- the leakage power $P^{\text{recov}}_{\text{static}}$ after power recovery and the leakage power $P^{\text{fixup}}_{\text{static}}$ after violation fixup,
- its relative difference to the Lagrange flow in percent $\Delta P^{\text{fixup}}_{\text{static}}$,
- the lower bound $P^{\text{lb}}_{\text{static}}$ for TNS optimization according to Corollary 4.3,
- the ratio between the lower bound and the given solution,
- the total power consumption P_{total},
- its relative difference to the Lagrange flow in percent ΔP_{total} and
- the running time of the V_t assignment or gate sizing algorithm respectively.

Slew limit violations were negligible at the end of each flow.

Algorithm 4.2 without global routing shows significant reductions of the leakage power compared to the result of "Lagrange". On uP_14 the reduction is 8.7%. Here we are provably less than 4% away from the optimum solution.

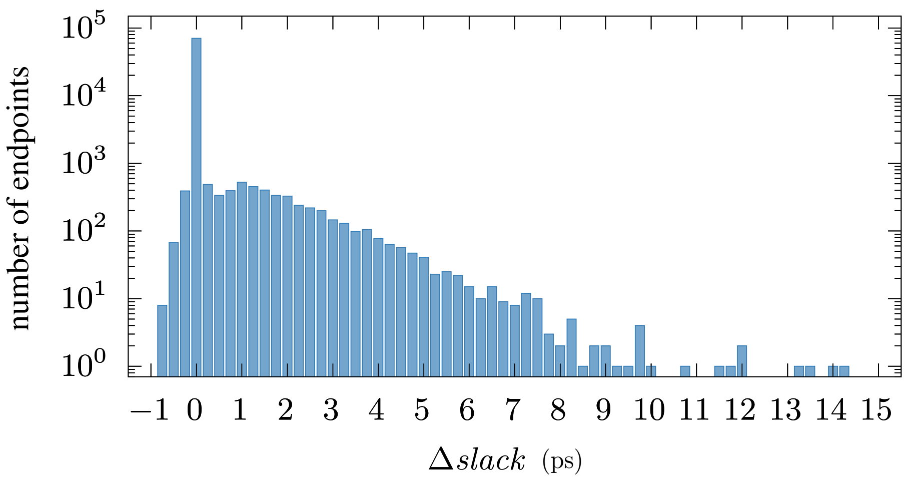


Figure 4.5: A histogram of the 75787 endpoint slack differences for all instances combined, considering only infeasible endpoints. Scale is logarithmic.

Algorithm 4.2 maximizes the TNS, which, thus, is never worse than the TNS of “Lagrange” and in most cases it yields a better TNS. For the three instances uP_02, uP_05, and uP_12, the power limit was reached in Step 2. The subsequent power recovery pushed the leakage power below the limit.

The effect of the power recovery is usually small, but on some instances as uP_01 it can save up to 2% of power in the post-GR mode. Similarly the increase of power by the violation fix up is not significant as only few violations are introduced to begin with. For cap violations there were at most 2 additional cap violations on a single instance and in total there is 1 cap violation less after violation fixup.

We point out that in every run the TNS was maximized by our algorithm. To verify this we analyzed the change of endpoint slacks on the 75787 endpoints of all instances uP_01-uP_14 between the end of Algorithm 4.2 and the reference algorithm [RSR16a] for which either of the algorithms did not meet the slack target. Due to slight timing degradations by the power recovery, the worst degradation at a single pin which we measured was -0.79ps. The best improvement of a pin was 14.3ps. The average improvement is 0.12ps, and the total improvement across all instances is 9.2ns. A histogram of the endpoint slacks can be found in Figure 4.5.

By degrading the TTNS a major leakage reduction is possible as e.g. instance uP_13 shows where the TNS-opt flow reduces the leakage by 28%. The worst approximation guarantee we obtain is 3.54 on instance uP_11. In any case the computed guarantee is significantly better than k which ranges from 13 to 50.

When used after timing-aware global routing, the leakage reduction by Algorithm 4.2 is even more significant. Algorithm 4.2, which does not require any re-routes, reduces the leakage power by up to 34%. The reason is that the

	Instance with 110k gates					
α	0.1	0.25	0.5	0.75	1.0	2.0
Iterations	15929	6226	3180	2174	1669	952
Runtime $[h{:}m]$	2:13	1:09	0:47	0:41	0:37	0:30
Leakage $[\mu W]$	40.73	40.72	40.68	40.68	40.63	40.77

	Instance with 1500k gates					
α	0.1	0.25	0.5	0.75	1.0	2.0
Iterations	14974	5948	3119	2187	1637	1112
Runtime $[h{:}m]$	16:15	11:31	6:34	6:15	5:27	4:53
Leakage $[\mu W]$	361.40	361.47	361.75	361.74	362.17	362.49

Table 4.1: Running times and total leakages on two designs depending on α, i.e. the number of selected paths.

timing-aware global wires mostly result in faster signal delays and the design flow uses slightly pessimistic delay estimates before global routing. Thus, the TNS and TTNS after global routing improve compared to the Steiner estimates with layer assignment. This becomes also noticeable in the mostly reduced lower bounds on the leakage power. However, some wire delays and sometimes even the WS degrade after global routing, e.g. on uP_01 and uP_11.

4.5.1 Running Time Evaluation

In addition to the results in Table 4.2 we also tried to analyse the scalability and running time efficiency of our algorithm. To do this we measured the running time on two larger designs, a large one with 1.5 million gates and a moderate instance with 110k gates. To investigate the scaling of the algorithm we disabled any preprocessing. As indicated in Section 4.4.6 the number of paths that is selected in every iteration has a big impact on the total running time. Therefore, we tried various different values for the parameter α introduced in Section 4.4.6.

For the larger instance approximately 1 million acceleration operations were performed in order to reach a timing feasible solution. A naive implementation of our algorithm would thus perform 1 million iterations, each of which requires a full timing analysis of the instance.

In practice the situation looks much better. Indeed, we can accelerate about 1000 paths independently on this instance as can be seen in Table 4.1. Note that even if the number of iterations is about inversely proportional to α the running time doesn't fully scale as we evaluate slack changes as described in

Section 4.4.3.

As bigger instances usually allow more paths to be collected, the number of iterations is almost constant, implying an almost linear practical running time of our algorithm. If α is too large, we obtain slightly worse results, thus our default choice of $\alpha = 1$.

With preprocessing and $\alpha = 1$ these two instances run in 0:20 and 2:31 hours respectively.

Instance	**Flow**	$\lvert\mathcal{G}\rvert$	d	**WS** [ps]	**TNS** [ns]	**TTNS** [ns]	$P^{\text{apx}}_{\text{static}}$ [μW]	$P^{\text{recov}}_{\text{static}}$ [μW]	$P^{\text{fixup}}_{\text{static}}$ [μW]	$\Delta P^{2}_{\text{static}}$	$P^{\text{lb}}_{\text{static}}$ [μW]	$\frac{P_{\text{static}}}{P^{\text{lb}}_{\text{static}}}$	P_{total} [μW]	ΔP_{total}	**Time** [h:m:s]
uP_01	Initial	99k	28	-69.5	-101.4	-602.6			81.7	+13.2%			95.1	+11.2%	
	Lagrange [RSR16a]			-69.5	-96.4	-590.5			72.2		21.2	3.40	85.5		6:27:19
	Alg. 4.2 TNS-opt			-69.5	-94.8	-894.5	56.3	56.3	56.4	-21.8%	21.2	2.66	69.8	-18.4%	49:05
	Alg. 4.2			-69.5	-95.1	-590.5	71.2	70.3	70.5	-2.4%	21.2	3.32	83.8	-2.0%	1:00:46
	Alg. 4.2 post-GR			-69.9	-78.2	-448.6	66.0	64.7	64.9	-10.1%	20.7	3.06	78.2	-8.6%	51:14
uP_02	Initial	10k	50	-156.9	-1.9	-10.0			1.2	+6.8%			2.4	+4.4%	
	Lagrange [RSR16a]			-157.0	-1.9	-10.8			1.1		0.9	1.25	2.3		45:16
	Alg. 4.2 TNS-opt			-157.0	-1.8	-18.4	1.1	1.1	1.1	-6.1%	0.9	1.18	2.3	-3.0%	0:49
	Alg. 4.2			-157.0	-1.8	-11.6	1.1	1.1	1.1	-1.0%	0.9	1.24	2.3	-0.5%	1:15
	Alg. 4.2 post-GR			-154.6	-1.9	-9.8	1.1	1.1	1.1	-0.9%	0.9	1.24	2.3	-1.0%	0:45
uP_03	Initial	9k	22	7.0	-0.0	-0.0			2.7	+2.0%			52.7	+0.7%	
	Lagrange [RSR16a]			7.0	-0.0	-0.0			2.7		1.7	1.54	52.4		57:52
	Alg. 4.2 TNS-opt			7.0	-0.0	-0.0	2.5	2.5	2.5	-5.5%	1.7	1.46	52.3	-0.2%	4:32
	Alg. 4.2			7.0	-0.0	-0.0	2.5	2.5	2.5	-5.1%	1.7	1.46	52.3	-0.2%	4:23
	Alg. 4.2 post-GR			7.0	-0.0	-0.0	2.3	2.2	2.3	-15.4%	1.7	1.30	51.4	-1.9%	4:02
uP_04	Initial	7k	25	-11.2	-0.7	-0.7			1.6	+0.3%			2.9	+0.3%	
	Lagrange [RSR16a]			-11.2	-0.7	-0.7			1.6		1.6	1.01	2.9		31:18
	Alg. 4.2 TNS-opt			-11.2	-0.7	-0.7	1.6	1.6	1.6	-0.1%	1.6	1.01	2.9	-0.0%	0:08
	Alg. 4.2			-11.2	-0.7	-0.7	1.6	1.6	1.6	-0.1%	1.6	1.01	2.9	-0.0%	0:08
	Alg. 4.2 post-GR			-5.5	-0.4	-0.4	1.6	1.6	1.6	-0.2%	1.6	1.01	2.9	-0.6%	0:06
uP_05	Initial	16k	22	-76.6	-36.6	-64.0			20.3	+2.8%			67.3	+1.0%	
	Lagrange [RSR16a]			-76.6	-36.8	-64.2			19.7		9.3	2.12	66.6		31:20
	Alg. 4.2 TNS-opt			-76.6	-36.7	-72.3	18.9	18.9	18.9	-4.0%	9.3	2.03	65.8	-1.2%	10:03
	Alg. 4.2			-76.6	-36.7	-64.5	19.7	19.7	19.7	-0.2%	9.3	2.11	66.6	-0.1%	10:18
	Alg. 4.2 post-GR			-69.5	-27.3	-46.9	18.0	18.0	18.0	-8.8%	9.0	1.93	63.4	-4.8%	7:09
uP_06	Initial	77k	29	-108.9	-15.9	-25.6			35.7	+5.4%			147.6	+1.3%	
	Lagrange [RSR16a]			-108.9	-14.5	-24.0			33.9		23.5	1.44	145.8		3:20:22
	Alg. 4.2 TNS-opt			-108.9	-13.3	-26.5	31.9	31.9	32.2	-5.0%	23.5	1.37	144.1	-1.2%	25:04
	Alg. 4.2			-108.9	-13.3	-23.9	32.2	32.1	32.5	-4.2%	23.5	1.38	144.3	-1.0%	24:52
	Alg. 4.2 post-GR			-107.7	-9.8	-15.2	28.2	28.1	28.3	-16.4%	22.3	1.21	140.2	-3.9%	17:10
uP_07	Initial	72k	25	-33.9	-38.6	-231.6			60.8	+9.1%			73.2	+7.5%	
	Lagrange [RSR16a]			-33.9	-39.2	-229.2			55.7		19.0	2.93	68.1		4:34:21
	Alg. 4.2 TNS-opt			-33.9	-36.6	-343.3	47.8	47.7	47.9	-14.1%	19.0	2.52	60.2	-11.5%	46:30
	Alg. 4.2			-33.9	-36.7	-228.6	54.5	53.9	54.1	-3.0%	19.0	2.85	66.4	-2.4%	51:00
	Alg. 4.2 post-GR			-32.4	-28.6	-153.0	49.7	49.1	49.2	-11.6%	18.4	2.59	61.6	-9.5%	44:02
uP_08	Initial	18k	28	-72.6	-35.1	-176.4			16.8	+8.1%			85.9	+2.7%	
	Lagrange [RSR16a]			-72.6	-34.5	-176.8			15.5		6.1	2.53	83.7		1:13:35
	Alg. 4.2 TNS-opt			-72.6	-34.5	-248.8	11.8	11.8	11.8	-23.8%	6.1	1.93	79.9	-4.5%	11:04
	Alg. 4.2			-72.6	-34.5	-176.4	14.7	14.6	14.7	-5.6%	6.1	2.39	82.7	-1.1%	13:50
	Alg. 4.2 post-GR			-66.6	-26.7	-124.0	13.8	13.7	13.7	-11.7%	6.1	2.24	80.7	-3.5%	12:01
uP_09	Initial	18k	22	-23.2	-8.8	-36.2			14.5	+10.3%			47.8	+3.2%	
	Lagrange [RSR16a]			-22.7	-8.6	-37.1			13.1		5.8	2.26	46.3		1:15:09
	Alg. 4.2 TNS-opt			-22.7	-8.2	-54.1	11.4	11.4	11.4	-13.0%	5.8	1.97	44.6	-3.7%	10:02
	Alg. 4.2			-22.7	-8.2	-36.8	12.7	12.6	12.7	-3.6%	5.8	2.18	45.9	-1.0%	10:54
	Alg. 4.2 post-GR			-22.4	-6.6	-24.2	11.6	11.5	11.6	-11.8%	5.6	1.99	44.5	-3.9%	9:33
uP_10	Initial	126k	23	-43.8	-76.0	-342.6			91.6	+17.0%			395.2	+5.3%	
	Lagrange [RSR16a]			-39.9	-80.5	-395.3			78.3		25.8	3.04	375.2		9:14:53
	Alg. 4.2 TNS-opt			-40.0	-73.9	-531.1	67.0	67.0	67.2	-14.2%	25.8	2.61	364.0	-3.0%	1:45:27
	Alg. 4.2			-39.9	-74.0	-392.8	73.5	73.2	73.4	-6.3%	25.8	2.85	370.3	-1.3%	1:55:15
	Alg. 4.2 post-GR			-31.4	-30.8	-119.0	52.1	51.5	51.7	-34.0%	25.9	2.01	343.1	-8.6%	1:47:15
uP_11	Initial	25k	38	-140.7	-167.2	-886.7			39.7	+6.4%			61.6	+4.0%	
	Lagrange [RSR16a]			-140.3	-165.2	-878.4			37.3		10.1	3.67	59.2		1:36:54
	Alg. 4.2 TNS-opt			-140.3	-165.4	-990.7	27.0	27.0	27.0	-27.5%	10.1	2.66	48.9	-17.4%	12:07
	Alg. 4.2			-140.3	-165.1	-877.8	35.9	35.9	36.0	-3.6%	10.1	3.54	57.8	-2.2%	15:42
	Alg. 4.2 post-GR			-142.6	-157.8	-826.6	35.5	35.5	35.6	-4.6%	9.9	3.50	57.4	-2.9%	13:18
uP_12	Initial	18k	38	-417.8	-342.0	-696.1			5.1	+5.7%			25.5	+1.9%	
	Lagrange [RSR16a]			-417.8	-342.5	-699.4			4.8		3.7	1.30	25.0		1:10:42
	Alg. 4.2 TNS-opt			-417.8	-340.5	-753.7	4.5	4.5	4.6	-4.9%	3.7	1.23	24.8	-1.0%	3:43
	Alg. 4.2			-417.8	-340.5	-712.3	4.8	4.8	4.8	-0.2%	3.7	1.29	25.0	-0.0%	4:19
	Alg. 4.2 post-GR			-416.2	-332.9	-682.8	4.8	4.8	4.8	-0.3%	3.7	1.29	24.9	-0.3%	2:35
uP_13	Initial	20k	17	-47.6	-20.8	-103.4			19.6	+6.8%			80.2	+1.9%	
	Lagrange [RSR16a]			-47.6	-20.6	-103.6			18.4		6.5	2.85	78.7		1:08:54
	Alg. 4.2 TNS-opt			-47.6	-20.4	-152.7	13.2	13.2	13.2	-28.1%	6.5	2.05	73.5	-6.6%	8:09
	Alg. 4.2			-47.6	-20.5	-103.3	18.3	18.1	18.1	-1.5%	6.5	2.81	78.4	-0.4%	10:28
	Alg. 4.2 post-GR			-42.9	-17.9	-88.8	17.2	17.1	17.1	-6.9%	6.3	2.65	77.2	-1.9%	8:42
uP_14	Initial	13k	13	-54.8	-5.1	-9.2			8.2	+0.1%			17.9	+0.0%	
	Lagrange [RSR16a]			-54.8	-5.1	-9.2			8.2		7.3	1.13	17.9		23:16
	Alg. 4.2 TNS-opt			-54.8	-5.1	-9.2	7.5	7.5	7.5	-8.7%	7.3	1.03	17.2	-4.0%	0:32
	Alg. 4.2			-54.8	-5.1	-9.2	7.5	7.5	7.5	-8.7%	7.3	1.03	17.2	-4.0%	0:32
	Alg. 4.2 post-GR			-54.7	-5.0	-9.0	7.4	7.4	7.5	-8.9%	7.2	1.03	17.2	-4.1%	0:33

Table 4.2: Results on 22nm microprocessor instances. Alg. 4.2 runs as a postprocessing of the Lagrange [RSR16a] algorithm.

Chapter 5

Theoretic Bounds for Time-Cost Tradeoff Problems

In the last chapter, we described a practical algorithm for the V_t assignment problem. Its counterpart in combinatorial graph theory is the discrete time-cost tradeoff (TCT) problem. Here, we are given a set of jobs V and a partial order $(V, \prec)$. Every job has a set of possible execution times which may differ in their cost. The problem is to assign an execution time for every job such that a global deadline is met while minimizing the cost. Equivalently, such an instance can be seen as a directed graph. In this chapter we will analyze the approximability of this problem. Parts of this chapter have been previously published in [DHV20]. The results are joint work with Stephan Held and Jens Vygen, with the exception of Section 5.7 and 5.8 where we study further variants of the time-cost tradeoff problem.

The depth of an instance is the number of jobs in a longest chain and is denoted by d. In the last chapter we showed a practical d approximation. In this chapter we observe that the problem can be regarded as a special case of finding a minimum-weight vertex cover in a d-partite hypergraph. Next, we study the natural LP relaxation which can be solved in polynomial time for fixed d and — for time-cost tradeoff instances — up to an arbitrarily small error in general.

Based on prior work of Lovász [Lov75] and of Aharoni, Holzman and Krivelevich [AHK96], we describe a deterministic algorithm with an approximation ratio slightly less than $\frac{d}{2}$ for minimum-weight vertex cover in d-partite hypergraphs for fixed d and given d-partition. This is tight and also yields a $\frac{d}{2}$-approximation algorithm for general time-cost tradeoff instances.

Note that in the classic description we assume that the execution times are separable and depend on at most a single job. Our algorithm from Chapter 4 did not need this assumption. Therefore, it is likely hard to beat in practice, even though only providing a worse theoretic bound. In fact, our lower bounds from Chapter 4 already proved that we compute much better solutions than the guarantee of d and even $\frac{d}{2}$.

We also study the inapproximability and show that no better approximation ratio than $\frac{d+2}{4}$ is possible, assuming the Unique Games Conjecture and P $\neq$ NP. This strengthens a result of Svensson [Sve12], who showed that under the same assumptions no constant-factor approximation algorithm exists for general time-cost tradeoff instances (of unbounded depth). Previously, only APX-hardness was known for bounded depth.

At the end of the chapter, we will analyze some slight variations of the classic time-cost tradeoff problem. First, we consider the power recovery problem in which we aim to find a cheapest solution that is still feasible. Then we analyze a variant in which timing constraints are moved to the objective function.

5.1 Previous Work

The (deadline version of the discrete) time-cost tradeoff problem was introduced in the context of project planning and scheduling more than 60 years ago [KW59]. An instance of the *time-cost tradeoff problem* consists of a finite set V of jobs, a partial order $(V, \prec)$, a deadline $T > 0$, and for every job v a finite nonempty set $S_v \subseteq \mathbb{R}^2_{\geq 0}$ of time/cost pairs. An element $(t, c) \in S_v$ corresponds to a possible choice of performing job v with delay t and cost c. The task is to choose a pair $(t_v, c_v) \in S_v$ for each $v \in V$ such that $\sum_{v \in P} t_v \leq T$ for every chain P (equivalently: the jobs can be scheduled within a time interval of length T, respecting the precedence constraints), and the goal is to minimize $\sum_{v \in V} c_v$.

The partial order can be described by an acyclic digraph $G = (V, E)$, where $(v, w) \in E$ if and only if $v \prec w$. Every chain of jobs corresponds to a path in G, and vice versa.

De et al. [De+97] proved that this problem is strongly NP-hard. Indeed, there is an approximation-preserving reduction from vertex cover [GW04], which implies that, unless P = NP, there is no 1.3606-approximation algorithm [DS05]. Assuming the Unique Games Conjecture and P $\neq$ NP, Svensson [Sve12] could show that no constant-factor approximation algorithm exists.

Even though the time-cost tradeoff has been extensively studied due to its numerous practical applications, only few positive results about approximation algorithms are known. Skutella [Sku98] described an algorithm that works if all delays are natural numbers in the range $\{0, \ldots, l\}$ and returns an l-approximation. If one is willing to relax the deadline, one can use Skutella's bicriteria approximation algorithm [Sku98]. For a fixed parameter $0 < \mu < 1$, it computes a solution in polynomial time such that the optimum cost is exceeded by a factor of at most $\frac{1}{1-\mu}$ and the deadline T is exceeded by a factor of at most $\frac{1}{\mu}$. Unfortunately, for many applications, including VLSI design, relaxing the deadline is out of the question.

The instances of the time-cost tradeoff problem that arise in the context of VLSI design usually have a constant upper bound d on the number of vertices on any path [Dab+18b]. This is due to a given target frequency of the chip, which can only be achieved if the logic depth is bounded. For this important special case, we will describe better approximation algorithms.

The special case $d = 2$ reduces to weighted bipartite matching and can thus be solved optimally in polynomial time. However, already the case $d = 3$ is APX-hard. This was observed by Deĭneko and Woeginger [DG01] who devised an approximation-preserving reduction from vertex cover in cubic graphs (which is known to be APX-hard [AK00]).

On the other hand, it is easy to obtain a d-approximation algorithm: either by applying the Bar-Yehuda–Even algorithm for set covering [BE81; Dab+18b] or (for fixed d) by simple LP rounding.

As we will observe in Section 5.3, the time-cost tradeoff problem with depth d can be viewed as a special case of finding a minimum-weight vertex cover in a d-partite hypergraph. Lovász [Lov75] studied the unweighted case and proved that the natural LP has integrality gap $\frac{d}{2}$. Aharoni, Holzman and Krivelevich [AHK96] showed this ratio for more general unweighted hypergraphs by randomly rounding a given LP solution. Gutswami, Sachdeva and Saket [GSS00] proved that approximating the vertex cover problem in d-partite hypergraphs with a better ratio than $\frac{d}{2} - 1 + \frac{1}{2d}$ is NP-hard, and better than $\frac{d}{2}$ is NP-hard if the Unique Games Conjecture holds.

5.2 Results and Outline

In the following sections, we first reduce the time-cost tradeoff problem with depth d to finding a minimum-weight vertex cover in a d-partite hypergraph. Then we simplify and derandomize the algorithm of Lovász [Lov75] and Aharoni et al. [AHK96] and show that it works for general nonnegative weights. This yields a simple deterministic $\frac{d}{2}$-approximation algorithm for minimum-weight vertex cover in d-partite hypergraphs for fixed d and given d-partition. To obtain a $\frac{d}{2}$-approximation algorithm for the time-cost tradeoff problem, we need a slightly stronger bound because the vertex cover LP can only be solved approximately (unless d is fixed). This will imply our main approximation:

Theorem 5.1. *There is a polynomial-time $\frac{d}{2}$-approximation algorithm for the time-cost tradeoff problem, where d denotes the depth of the instance.*

The algorithm is based on rounding an approximate solution to the vertex cover LP. The basic idea is quite simple: we partition the jobs into levels and carefully choose an individual threshold for every level, then we accelerate all jobs for which the LP solution is above the threshold of its level. We get a solution that costs less than $\frac{d}{2}$ times the LP value. Since the integrality gap is $\frac{d}{2}$ [Lov75; AHK96] (even for time-cost tradeoff instances; see Section 5.3), this ratio is tight.

The results by [GSS00] suggest that this approximation guarantee is essentially best possible for general instances of the vertex cover problem in d-partite hypergraphs. Still, better algorithms might exist for special cases such as the time-cost tradeoff problem.

Finally, we show that much better approximation algorithms are unlikely to exist even for time-cost tradeoff instances. More precisely, we prove:

Theorem 5.2. *Let $d \in \mathbb{N}$ with $d \geq 2$ and $\rho < \frac{d+2}{4}$ be constants. Assuming the Unique Games Conjecture and* $\mathrm{P} \neq \mathrm{NP}$*, there is no polynomial-time ρ-approximation algorithm for time-cost tradeoff instances with depth d.*

This gives strong evidence that our approximation algorithm is best possible up to a factor of 2. To obtain our inapproximability result, we leverage Svensson's theorem on the hardness of vertex deletion to destroy long paths in an acyclic digraph [Sve12] and strengthen it to instances of bounded depth by a novel compression technique.

Section 5.3 introduces the vertex cover LP and explains why the time-cost tradeoff problem with depth d can be viewed as a special case of finding a minimum-weight vertex cover in a d-partite hypergraph.

In Section 5.4 we describe our approximation algorithm, which rounds a solution to this LP. Then, in Sections 5.5 and 5.6 we prove our inapproximability result.

5.3 The Vertex Cover LP

Let us define the *depth* of an instance of the time-cost tradeoff problem to be the number of jobs in the longest chain in $(V, \prec)$, or equivalently the number of vertices in the longest path in the associated acyclic digraph $G = (V, E)$. We write $n = |V|$ and the depth will be denoted by d throughout this chapter.

First, we note that one can restrict attention to instances with a simple structure, where every job has only two alternatives and the task is to decide which jobs to accelerate. This has been observed already by Skutella [Sku98]. The following definition describes the structure that we will work with.

Definition 5.3. *An instance I of the time-cost tradeoff problem is called* normalized *if for each job $v \in V$ the set of time/cost pairs is of the form $S_v = \{(0, c), (t, 0)\}$ for some $c, t \in \mathbb{R}_+ \cup \{\infty\}$.*

In a normalized instance, every job has only two possible ways of being executed. The slow execution is free and the fast execution has a delay of zero. Therefore, the time-cost tradeoff problem is equivalent to finding a subset $F \subseteq V$ of jobs that are to be executed fast. The objective is to minimize the total cost of jobs in F. Note that for notational convenience we allow one of the alternatives to have infinite delay or cost, but of course such an alternative can never be chosen in a feasible solution of finite cost, and it could be as well excluded.

We call two instances I and I' of the time-cost tradeoff problem *equivalent* if any feasible solution to I can be transformed in polynomial time to a feasible solution to I' with the same cost and vice-versa. We include a proof of Skutella's observation for sake of completeness.

Proposition 5.4 (Skutella [Sku98]). *For any instance I of the time-cost tradeoff problem one can construct an equivalent normalized instance I' of the same depth in polynomial time.*

Proof Let v be a job of instance I with $S_v = \{(t_1, c_1), \ldots, (t_r, c_r)\}$. By sorting and removing dominated pairs, we may assume $t_1 < \ldots < t_r$ and $c_1 > \ldots > c_r$.

To construct I', we replace v by $r+1$ copies $v_0, v_1, \ldots, v_r$ of v, each with the same predecessors and successors. We define $S_{v_i} := \{(0, c_i - c_{i+1}), (t_{i+1}, 0)\}$, where $c_0 := \infty$, $c_{r+1} := 0$, and $t_{r+1} := \infty$.

As the slow alternatives of the copies v_i have increasing delay in i, an optimum solution always sets consecutive jobs $v_j, v_{j+1}, \ldots v_r$ to the fast solution. As the last slow solution has infinite delay and the first one has infinite cost, $1 \leq j \leq r$. Then the total cost at v is given by $\sum_{i=j}^{r}(c_i - c_{i+1}) = c_j - c_{r+1} = c_j$. As accelerated jobs have delay 0, the longest path through a copy of v is determined by v_{j-1}, which has delay t_j.

Note that it is easy to convert the corresponding solutions of both instances into each other in polynomial time. □

The structure of only allowing two execution times per job gives rise to a useful property, as we will now see. As noted above, for a normalized instance I the solutions correspond to subsets of jobs $F \subseteq V$ to be accelerated. Consider the clutter $\mathcal{C}$ of inclusion-wise minimal feasible solutions to I. Denote by $\mathcal{B} = \operatorname{bl}(\mathcal{C})$ the blocker of $\mathcal{C}$, i.e., the clutter over the same ground set V whose members are minimal subsets of jobs that have nonempty intersection with every element of $\mathcal{C}$.

Let $T > 0$ be the deadline of our normalized time-cost tradeoff instance and t_v denote the slow delay of executing job $v \in V$. By the properties of a normalized instance, the elements of $\mathcal{B}$ are the minimal chains $P \subseteq V$ with $\sum_{v \in P} t_v > T$. The well-known fact that $\operatorname{bl}(\operatorname{bl}(\mathcal{C})) = \mathcal{C}$ [EF70; Isb58] immediately implies the next proposition, which also follows from an elementary calculation.

Proposition 5.5. *A set $F \subseteq V$ is a feasible solution to a normalized instance I of the time-cost tradeoff problem if and only if $P \cap F \neq \varnothing$ for all $P \in \mathcal{B}$.* □

Therefore, our problem is to find a minimum-weight vertex cover in the hypergraph $(V, \mathcal{B})$. If our time-cost tradeoff instance has depth d, this hypergraph is d-partite and a d-partition can be computed easily:

Proposition 5.6. *Given a time-cost tradeoff instance with depth d, we can partition the set of jobs in polynomial time into sets $V_1, \dots, V_d$ (called* layers*) such that $v \prec w$ implies that $v \in V_i$ and $w \in V_j$ for some $i < j$. Then, $|P \cap V_i| \leq 1$ for all $P \in \mathcal{B}$ and $i = 1, \dots, d$.*

Proof Such a partition can be found by constructing the acyclic digraph $G = (V, E)$ with $(v, w) \in E$ if and only if $v \prec w$ and setting $V_i := \{v \in V : l(v) = i\}$, where $l(v)$ denotes the maximum number of vertices in any path in G that ends in v. □

This also leads to a simple description as an integer linear program. The feasible solutions correspond to the vectors $x \in \{0, 1\}^V$ with $\sum_{v \in P} x_v \geq 1$ for all $P \in \mathcal{B}$. We consider the following linear programming relaxation, which we call the *vertex cover LP*:

$$\begin{array}{lll} \text{minimize:} & \sum\limits_{v \in V} c_v \cdot x_v & \\ \text{subject to:} & \sum\limits_{v \in P} x_v \geq 1 & \text{for all } P \in \mathcal{B} \\ & x_v \geq 0 & \text{for all } v \in V. \end{array} \tag{5.1}$$

Let LP denote the value of this linear program for a given instance. Unless P=NP, one cannot solve this linear program exactly in polynomial time:

Proposition 5.7. *If the vertex cover LP* (5.1) *can be solved in polynomial time for normalized time-cost tradeoff instances, then* P = NP.

Proof By the equivalence of optimization and separation [GLS81], it suffices to show that the separation problem is NP-hard. In fact, we show that deciding whether a given vector x is infeasible for a given instance is NP-complete. To this end, we transform the Partition problem, which is well known to be NP-complete: given a list $a_1, \dots, a_n$ of positive integers, is there a subset $I \subseteq \{1, \dots, n\}$ with $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$? Given an instance $a_1, \dots, a_n$ of Partition, construct a time-cost tradeoff instance with $2n$ jobs v_{ij} ($i = 1, \dots, n$, $j = 0, 1$), where $v_{ij} \prec v_{i'j'}$ whenever $i < i'$. The fast execution time is 0 for all jobs, and the slow execution time is also 0 for v_{i0} but a_i for v_{i1}. The deadline is $T := \frac{A-1}{2}$, where $A = \sum_{i=1}^{n} a_i$. Let $x_{v_{i0}} := 0$ and $x_{v_{i1}} := \frac{2a_i}{A+1}$. Then x is a feasible solution to the LP if for all subsets $I \subseteq \{1, \dots, n\}$ $\sum_{i \in I} a_i \leq T$ or $\sum_{i \in I} x_{v_{i1}} \geq 1$, which means $\sum_{i \in I} a_i \leq \frac{A-1}{2}$ or $\sum_{i \in I} a_i \geq \frac{A+1}{2}$, or equivalently $\sum_{i \in I} a_i \neq \frac{A}{2}$. $\square$

However, we can solve the LP up to an arbitrarily small error; in fact, there is a fully polynomial approximation scheme (as essentially shown by [KK82]):

Proposition 5.8. *For normalized instances of the time-cost tradeoff problem with bounded depth, the vertex cover LP* (5.1) *can be solved in polynomial time. For general normalized instances and any given $\epsilon > 0$, a feasible solution of cost at most* $(1+\epsilon)$LP *can be found in time bounded by a polynomial in n and $\frac{1}{\epsilon}$.*

Proof If the depth is bounded by a constant d, the number of constraints is bounded by the polynomial $|V|^d$, so the first statement follows from any polynomial-time linear programming algorithm.

Otherwise, we solve the LP up to a factor $1+\epsilon$ for any given $0 < \epsilon \leq 1$ as follows. Implement an approximate separation oracle by first rounding up the components of a given vector x to integer multiples of $\frac{\epsilon}{2d}$ and then applying dynamic programming to check whether $\sum_{v \in P} \frac{\epsilon}{2d} \lceil \frac{2dx_v}{\epsilon} \rceil \geq 1$ for all $P \in \mathcal{B}$. This requires $\mathcal{O}(\frac{dn^2}{\epsilon})$ time.

Run the ellipsoid method with this oracle. It computes an optimum solution x to a relaxed linear program, hence with cost at most LP. Moreover, $(1+\epsilon)x$ is a feasible solution to the original LP (5.1) because for every $P \in \mathcal{B}$ we have $\frac{\epsilon}{2} + \sum_{v \in P} x_v \geq \sum_{v \in P} \left(x_v + \frac{\epsilon}{2d}\right) \geq 1$, implying $(1+\epsilon) \sum_{v \in P} x_v \geq (1+\epsilon)(1-\frac{\epsilon}{2}) \geq 1$. $\square$

We remark that the d-partite hypergraph vertex cover instances given by [AHK96] can be also considered as normalized instances of the time-cost tradeoff problem; see Figure 5.1. This shows that the integrality gap of LP (5.1) is at least $\frac{d}{2}$.

Theorem 5.9 ([AHK96])**.** *Let $d \geq 2$. For instances with depth d, the integrality gap of the linear program (5.1) is at least $\frac{d}{2}$.*

Proof Consider an acyclic digraph with $n = d(k+1)$ vertices $V = \{(i,j) | i = 1, \dots, d, j = 0, \dots, k\}$ for odd $k \in \mathbb{N}$. We add edges between (i,j) and (i',j') whenever $i' = i+1$. The slow variant of job (i,j) takes duration j without any cost. By paying a cost of 1 the duration drops to 0. Let the deadline be given by $T = \frac{dk}{2}$. The instance for $d = 3$ is depicted in Figure 5.1.

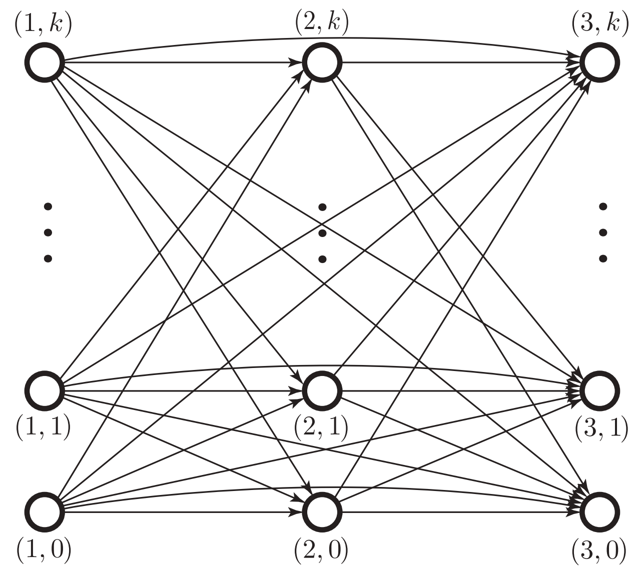


Figure 5.1: The structure of the time-cost tradeoff instance attaining an asymptotic gap of $d/2$ (here, $d = 3$). Figure reprinted from our publication [DHV20].

Consider the fractional solution of assigning vertex (i, j) a value of $x_{(i,j)} = \frac{j}{T}$. The cost of this solution is given by

$$c_{\text{frac}} = \sum_{(i,j)\in V} x_{(i,j)} = \sum_{(i,j)\in V} \frac{j}{T} = \frac{d}{T} \sum_{j=0}^{k} j = \frac{dk(k+1)}{2T} = k + 1.$$

We need to show that the fractional solution satisfies all constraints. Consider a violated path $P = (1, j_1), \ldots, (d, j_d)$. By construction the value $x_{(i,j)} = \frac{j}{T}$ corresponds exactly to the ratio between the delay of (i, j) and the deadline. Therefore, P has a total x value of ≥ 1 if and only if the path has a total delay of $\geq T$.

Fix any integral solution. Let τ_i be the number of vertices in $V_i = \{(i, 0), \ldots, (i, k)\}$ that the solution accelerates. Thus, the total cost of the solution is $c_{\text{int}} = \sum_{i=1}^{d} \tau_i$. Clearly, the delay of the slowest job in level i in the integral solution is at least $k - \tau_i$. As the levels V_i and V_{i+1} are completely connected for any $i = 1, \ldots, d - 1$, we can conclude that

$$\sum_{i=1}^{d} (k - \tau_i) \leq T \quad \text{and hence} \quad \sum_{i=1}^{d} \tau_i \geq dk - T = \frac{dk}{2}.$$

We can thus bound the integrality gap by

$$\frac{c_{\text{int}}}{c_{\text{frac}}} \geq \frac{dk/2}{k+1} \xrightarrow[k\to\infty]{} \frac{d}{2}.$$

□

Since $|P| \leq d$ for all $P \in \mathcal{B}$, the Bar-Yehuda–Even algorithm [BE81] can be used to find an integral solution to the time-cost tradeoff instance of cost at most $d \cdot \mathrm{LP}$, in fact we proved this in the last chapter.

In the following we will improve on this. From now on, we assume that we are given a d-partition of a hypergraph and an LP solution; for time-cost tradeoff instances we get this from Propositions 5.6 and 5.8.

5.4 Rounding Fractional Vertex Covers in *d*-Partite Hypergraphs

In this and the following section, we show how to round a fractional vertex cover in a d-partite hypergraph with given d-partition. Together with the results of the previous section, this yields an approximation algorithm for time-cost tradeoff instances and will prove Theorem 5.1.

We will first start with an algorithm that only achieves a guarantee of $\frac{d}{2} + (\lceil \frac{d}{2} \rceil - \frac{d}{2})\frac{1}{d} \leq \lceil \frac{d+1}{2} \rceil$. It is still interesting as its analysis will reveal several properties of the vertex cover LP. In the next section we will show an improved algorithm, which is even slightly better than $\frac{d}{2}$.

For our analysis, we will also need to consider the dual linear program of the vertex cover LP 5.1:

$$\begin{aligned} \text{maximize:} \quad & \sum_{P \in \mathcal{B}} y_P \\ \text{subject to:} \quad & \sum_{P \in \mathcal{B}: v \in P} y_P \leq c_v, \qquad v \in V \\ & y_P \geq 0, \qquad P \in \mathcal{B}. \end{aligned} \tag{5.2}$$

For any instance of the d-partite hypergraph vertex cover problem, we are given a partition of the vertex set $V_1, \dots, V_q$, such that $|P \cap V_i| \leq 1$ for any hyperedge $P \in \mathcal{B}$. We call V_j a *layer* of G. For instances of the time-cost tradeoff problem we explained how to obtain such a partition in Proposition 5.6.

For our algorithm, we need the following important observation, which is due to linear program duality.

Lemma 5.10. *Let $(x_v^*)_{v \in V}$ be an optimum solution to the primal linear program* (5.1) *of cost LP. Let V_i be a layer in the d-partite hypergraph. Then, we have $\sum_{v \in V_i : x_v^* > 0} c_v \leq LP$.*

Proof Let $(y_P)_P$ be a corresponding dual solution. By complementary slackness we have

$$\sum_{v\in V:x_v^*>0} c_v = \sum_{v\in V:x_v^*>0} \sum_{P\in\mathcal{B}:v\in P} y_P. \tag{5.3}$$

Clearly every hyperedge $P \in \mathcal{B}$ has at most one vertex in V_i, therefore we have

$$\sum_{v\in V_i:x_v^*>0} \sum_{P\in\mathcal{B}:v\in P} y_P \leq \sum_{P\in\mathcal{B}} y_P.$$

Putting everything together we see that

$$\sum_{v\in V_i:x_v^*>0} c_v \leq \sum_{P\in\mathcal{B}} y_P = \mathrm{LP}.$$

Here the last equality follows from linear program duality. □

Corollary 5.11. *Let $(x_v^*)_v$ be an optimum solution to the primal linear program* (5.1) *of cost LP. Then, $\sum_{v\in V:x_v^*>0} c_v \leq d \cdot LP$.*

Proof We apply Lemma 5.10 to each layer and obtain:

$$\sum_{v\in V:x_v^*>0} c_v = \sum_{i=1}^{d} \sum_{v\in V_i:x_v^*>0} c_v \leq d \cdot \mathrm{LP}.$$

□

The statement of this Corollary was also observed previously by Klein and Wexler [KW16]. Because of its simplicity we present their proof here.

Lemma 5.12 ([KW16])**.** *Let $(x_v^*)_v$ be an optimum solution to the primal linear program* (5.1) *of cost LP. Also assume that $c_v = 1$ for all $v \in V$. Then, $\sum_{v\in V:x_v^*>0} x_v^* \geq \frac{1}{d} \cdot LP$.*

Proof By contradiction, suppose that the average of nonzero x_v^* variables is below $\frac{1}{d}$. Let $\epsilon = \min_{x_v^*>0} x_v^*$. If $\epsilon \geq \frac{1}{d}$ we obtain a contradiction. Otherwise, construct a solution by setting $h = \epsilon\left(\epsilon - \frac{1}{d}\right)^{-1}$ and $x'_v = x_v^* - h(x_v^* - \frac{1}{d})$. It is easy to see, that $x'_v \geq 0$ for all $v \in V$. Note that $h < 0$.

Let $\{v_1, \ldots, v_d\} \in \mathcal{B}$ be a hyperedge. As x_v^* is feasible we have $\sum_{i=1}^d x_{v_i}^* \geq 1$. Equivalently we observe $\sum_{i=1}^d (x_{v_i}^* - \frac{1}{d}) \geq 0$. Therefore,

$$\sum_{i=1}^{d} x'_{v_i} = \sum_{i=1}^{d} x_{v_i}^* - h\sum_{i=1}^{d}\left(x_{v_i}^* - \frac{1}{d}\right) \geq 1.$$

It follows that x'_v is feasible.

Note, that by our assumption $\sum_{v\in V}\Big(x_v^* - \frac{1}{d}\Big) < 0$. As h is negative the total cost of the new solution has decreased, which is a contradiction to the optimality of $(x_v^*)_{v\in V}$.

□

Our next goal is to devise a $\frac{d}{2} + (\lceil \frac{d}{2} \rceil - \frac{d}{2})\frac{1}{d}$ approximation algorithm.

5.4.1 A $\frac{d}{2} + (\lceil \frac{d}{2} \rceil - \frac{d}{2})\frac{1}{d}$ approximation

In this section we will improve the simple d approximation of the previous chapter to a $\frac{d}{2} + (\lceil \frac{d}{2} \rceil - \frac{d}{2})\frac{1}{d}$ approximation. To obtain our guarantee, we first show how to eliminate vertices with large value.

Lemma 5.13. *Let d be a constant and $\alpha \geq \frac{d}{2}$. Assume that we can solve the linear program* (5.1) *and that there is a polynomial-time algorithm $\mathcal{A}$ for the vertex cover problem for $\leq d$ partite hypergraphs, which computes a solution of cost at most $\alpha \cdot \mathrm{LP}$ whenever $x_v^* < \frac{2}{d}$ for all $v \in V$ in an optimum solution of the linear program* (5.1)*. Then there is an α-approximation algorithm for general instances.*

Proof The algorithm works as follows. Given an instance I with $\leq d$ paritions of the vertex set, first solve the LP (5.1) to obtain an optimum solution x^* (cf. Proposition 5.8). If $x_v^* < \frac{2}{d}$ for all $v \in V$, apply $\mathcal{A}$ and output the result. Otherwise let $v_0 \in V$ with $x_{v_0}^* \geq \frac{2}{d}$. Let I' be the instance where $x_{v_0} = 1$ is fixed to be in the vertex cover. Recursively solve I' and output the result, appending v_0.

The algorithm terminates because the amount of unfixed vertices strictly decreases in every iteration. It is also obvious that the output is a feasible solution. We show that it yields an α-approximation by induction on the amount of unfixed vertices. The case when we call $\mathcal{A}$ directly is trivial.

Let LP and LP′ denote the values of the linear program for the instance I and I', respectively. We have $\mathrm{LP}' \leq \mathrm{LP} - c_{v_0} \cdot x_{v_0}^*$. By induction our algorithm computes a solution to I' of cost at most $\alpha \cdot \mathrm{LP}'$; finally it accelerates v_0 to obtain an overall solution of cost at most

$$\begin{aligned} &\alpha\mathrm{LP}' + c_{v_0} \\ &\leq \alpha\Big(\mathrm{LP} - c_{v_0} \cdot x_{v_0}^*\Big) + c_{v_0} \\ &= \alpha\mathrm{LP} + \Big(1 - \alpha x_{v_0}^*\Big)c_{v_0} \qquad (5.4) \\ &\leq \alpha\mathrm{LP}, \end{aligned}$$

where we used $x_{v_0}^* \geq \frac{2}{d} \geq \frac{1}{\alpha}$ in the last inequality. □

Lemma 5.13 shows that we may assume that all vertices have small value in the LP solution. Then all elements of $\mathcal{B}$ contain vertices of nonzero LP value in many levels (more than $\frac{d}{2}$). Therefore choosing $\lceil \frac{d}{2} \rceil$ levels and returning vertices positive LP value in these levels yields a feasible solution, which is not too expensive by Lemma 5.10.

For odd d we can design a slightly improved algorithm:

Theorem 5.14. *Let d be a constant. Assume we can solve the linear program (5.1) in polynomial time. There is a polynomial time algorithm which, for any instance of the d-partite hypergraph vertex cover problem with given partition, computes a solution of cost at most $\frac{d}{2} + (\lceil \frac{d}{2} \rceil - \frac{d}{2})\frac{1}{d}$ times the optimum value of the linear program (5.1).*

Proof By Lemma 5.13 we can assume that we have an optimum solution $(x^*_v)_{v \in V}$ to the linear program (5.1) with the property $x^*_v < \frac{2}{d}$ for all $v \in V$.

First, consider the case of even d, we have to show a $\frac{d}{2}$ approximation guarantee.

Due to Corollary 5.11 we have $\sum_{v \in V : x^*_v > 0} c_v \leq d \cdot \mathrm{LP}$. Now consider some hyperedge $P \in \mathcal{B}$. The number of vertices $v \in P$ with $x^*_v > 0$ is more than $\frac{d}{2}$, so

$$|\{v \in P : x^*_v > 0\}| + \frac{d}{2} > d \geq |P|.$$

This shows that choosing all vertices v with $x^*_v > 0$ in $\frac{d}{2}$ of the levels $V_1, \ldots, V_d$ yields a solution that covers at least one vertex of every hyperedge. This is equivalent to being a feasible solution to the covering problem. By Lemma 5.10 the total cost is at most $\frac{d}{2}\mathrm{LP}$. Note that the choice of the levels was arbitrary.

Now assume that $d \geq 3$ is odd. We have to present a $\frac{d}{2} + \frac{1}{2d}$ approximation. First, choose a level V_j such that $\sum_{v \in V_j} c_v x^*_v$ is maximum. Now we "buy this level". More precisely we set $x'_v := \lceil x_v \rceil$ for $v \in V_j$ and $x'_v = x_v$ for $v \in V \setminus V_j$. By Lemma 5.10 we pay at most LP for this.

Let LP' be the cost of an optimum solution to this modified linear program where we fixed all variables for $v \in V_j$ to x'_v. By the choice of j,

$$\mathrm{LP}' \leq \left(1 - \frac{1}{d}\right)\mathrm{LP} = \frac{d-1}{d}\mathrm{LP}.$$

We now proceed with the remaining $d-1$ layers as above, obtaining a solution of $\frac{d-1}{2}\mathrm{LP}'$. If we add what we paid for layer V_j, we get an overall solution of total cost

$$\mathrm{LP} + \frac{d-1}{2}\mathrm{LP}'$$

$$\begin{aligned}&\leq \mathrm{LP} + \frac{d-1}{2}\cdot\frac{d-1}{d}\mathrm{LP}\\ &= \frac{d^2+1}{2d}\mathrm{LP} = \left(\frac{d}{2}+\frac{1}{2d}\right)\mathrm{LP}.\end{aligned}$$

□

5.4.2 An improved rounding algorithm

We will now improve the guarantee of the rounding algorithm from $\frac{d}{2}+(\lceil\frac{d}{2}\rceil-\frac{d}{2})\frac{1}{d}$ to $\frac{d}{2}$. A further advantage of the algorithm is, that we will not need to solve the LP again and do not even need an explicit list of the edge set of the hypergraph. This is interesting if d is not constant, as there can be exponentially many hyperedges. The algorithm only requires the vertex set, a d-partition, and a feasible solution to the LP (a fractional vertex cover). For normalized instances of the time-cost tradeoff problem such a fractional vertex cover can be obtained as in Proposition 5.8, and a d-partition by Proposition 5.6.

Our algorithm is based on two previous works for the unweighted d-partite hypergraph vertex cover problem. For rounding a given fractional solution, Lovász [Lov75] obtained a deterministic polynomial-time $(\frac{d}{2}+\epsilon)$-approximation algorithm for any $\epsilon > 0$. Let us quickly sketch his idea.

First, Lovász constructs a family of matrices $A_{d,w} = (a_{ij})_{i=1,\dots,d,j=0,\dots,w}$, with the property that:

- each row of $A_{d,w}$ is a permutation of $\{0,\dots,w\}$
- the sum of each column is at most $\leq \lceil\frac{dw}{2}\rceil$.

Now, for a fractional solution x to the d-partite hypergraph vertex cover problem, a (large) constant C is chosen, such that $x_vC \in \mathbb{N}$. The idea is to set $w = \lfloor 2(C-1)/d\rfloor$. Then, for every $j \in \{0,\dots,w\}$ we may obtain a feasible cover for every $j = 0,\dots,w$ by rounding all x_v for $v \in V_i$ to 1 if and only if $x_vC > a_{ij}$ (where $V_1,\dots,V_d$ is the given d-partition of our hypergraph). A simple analysis shows that returning the cheapest such cover is a $\frac{d}{2}\frac{C}{C-1}$ approximation, which converges to $\frac{d}{2}$ for $C \to \infty$.

Based on this, Aharoni, Holzman and Krivelevich [AHK96] described a randomized recursive algorithm that works in more general unweighted hypergraphs. We simplify their algorithm for d-partite hypergraphs, which will allow us to obtain a deterministic polynomial-time algorithm that also works for the weighted problem and always computes a $\frac{d}{2}$-approximation. At the end of this section, we will slightly improve on this guarantee in order to compensate for an only approximate LP solution.

We will first describe the algorithm in the even simpler randomized form; then we will derandomize it. Like Lovász, our algorithm computes a threshold

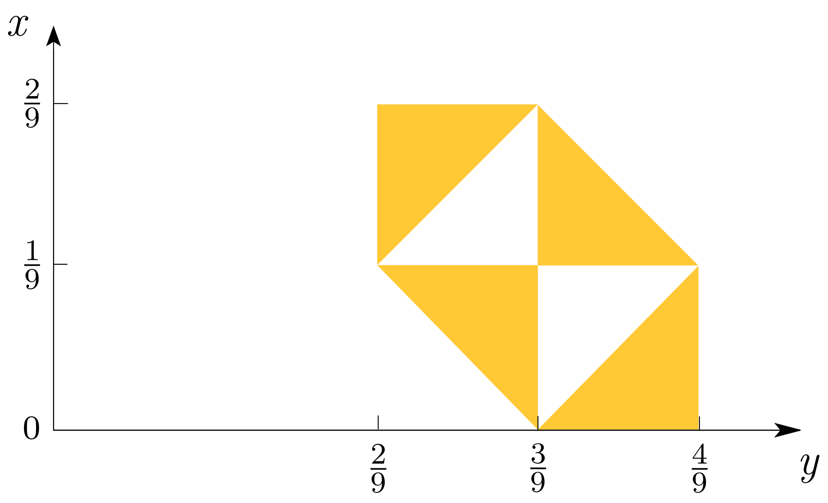


Figure 5.2: Selecting a pair (x, y) by uniformly sampling a point in the yellow area gives an example of how to choose random numbers x, y (and $z = 1 - x - y$) as in Lemma 5.15. Figure reprinted from our publication [DHV20].

for each layer to determine whether a variable x_v is rounded up or down. To compute the random thresholds, and to allow efficient derandomization later, we will use a probability distribution with the following properties.

Lemma 5.15. *There is a probability distribution that selects* $x \in [0, \frac{2}{9}], y \in [\frac{2}{9}, \frac{4}{9}], z \in [\frac{4}{9}, \frac{6}{9}]$*, such that* $x + y + z = 1$ *and* x, y, z *are uniformly distributed in their respective intervals.*

Proof Generate three random numbers in base 3, $a = 0.a_1a_2a_3, \ldots$, $b = 0.b_1b_2b_3, \ldots$, $c = 0.c_1c_2c_3, \ldots$, by randomly sampling digits $\{a_i, b_i, c_i\} = \{0, 1, 2\}$ (that is, we select a random permutation of $\{0, 1, 2\}$ to be the i-th digit of the three numbers). Let x' be the smallest number, y' the second smallest, and z' the largest number of a, b, c. It is easy to see, that $x' \in [0, \frac{1}{3}]$, $y' \in [\frac{1}{3}, \frac{2}{3}]$ and $z' \in [\frac{2}{3}, 1]$. Also by construction $x' + y' + z' = \frac{3}{2}$. Setting $x = \frac{2}{3}x', y = \frac{2}{3}y', z = \frac{2}{3}z'$ yields the desired result. $\square$

We remark that for implementing an algorithm that samples from this distribution, a different construction is more suitable. For example, one may start by selecting $x \in [0, \frac{2}{9}]$ randomly, and then use a case distinction as illustrated in Figure 5.2 to select y randomly in a suitable subset of $[\frac{2}{9}, \frac{4}{9}]$. Finally, we may set $z = 1 - x - y$. It is easy to verify that this also achieves the claimed properties.

For our proof we will need to slightly generalize this distribution to an arbitrary number of elements.

Lemma 5.16. *For any $d \geq 2$, there is a probability distribution that selects $a_1, \dots, a_d$, such that $\sum_{i=1}^d a_i = 1$ and a_i is uniformly distributed in $[\frac{2(i-1)}{d^2}, \frac{2i}{d^2}]$. For any i, j such that $|i-j| \geq 3$, the random variables corresponding to a_i and a_j are independent.*

Proof For $d = 2$ we can just choose a_1 uniformly in $[0, \frac{1}{2}]$ and set $a_2 = 1 - a_1$. The case $d = 3$ follows from Lemma 5.15. In general, note that the sum of the expectations of the a_i is $\sum_{i=1}^d \frac{2i-1}{d^2} = 1$. Hence we can partition $1, \dots, d$ into groups of two or three and apply the above with appropriate scaling and shifting.

More precisely, if d is odd, we choose x, y, z according to Lemma 5.15 and set $a_1 = \frac{9x}{d^2}$, $a_2 = \frac{9y}{d^2}$, and $a_3 = \frac{9z}{d^2}$. Then the remaining number of indices is even, and we group them into pairs; for indices i and $i+1$ we choose a_i uniformly in $[\frac{2(i-1)}{d^2}, \frac{2i}{d^2}]$ and set $a_{i+1} := \frac{4i}{d^2} - a_i$. $\square$

Theorem 5.17. *Let x be a fractional vertex cover in a d-partite hypergraph with given d-partition. There is a randomized linear-time algorithm that computes an integral solution $\bar{x}$ of expected cost $\mathbb{E}[\sum_{v \in V} c_v \cdot \bar{x}_v] \leq \frac{d}{2} \sum_{v \in V} c_v \cdot x_v$.*

Proof Let $V_1, \dots, V_d$ be the given d-partition of our hypergraph $(V, \mathcal{B})$, so $|P \cap V_i| = 1$ for all $i = 1, \dots, d$ and every hyperedge $P \in \mathcal{B}$. We write $l(v) = i$ if $v \in V_i$ and call V_i a *layer* of the given hypergraph.

Now consider the following randomized algorithm, which is also illustrated in Figure 5.3: Choose a random permutation $\sigma : \{1, \dots, d\} \to \{1, \dots, d\}$ and choose random numbers a_i uniformly distributed in $\left[\frac{2(\sigma(i)-1)}{d^2}, \frac{2\sigma(i)}{d^2}\right]$ for $i = 1, \dots, d$ such that $\sum_{i=1}^d a_i = 1$, as constructed in Lemma 5.16. Then, for all $v \in V$, set $\bar{x}_v := 1$ if $x_v \geq a_{l(v)}$ and $\bar{x}_v := 0$ if $x_v < a_{l(v)}$.

To show that $\bar{x}$ is a feasible solution, observe that any hyperedge $P \in \mathcal{B}$ has $\sum_{v \in P} x_v \geq 1 = \sum_{i=1}^d a_i \geq \sum_{v \in P} a_{l(v)}$ and hence $x_v \geq a_{l(v)}$ for some $v \in P$.

It is also easy to see that the probability that $\bar{x}_v$ is set to 1 is exactly $\min\{1, \frac{d}{2} x_v\}$. Indeed, if $x_v \geq \frac{2}{d}$, we surely set $\bar{x}_v = 1$. Otherwise, $x_v \in \left[\frac{2(j-1)}{d^2}, \frac{2j}{d^2}\right]$ for some $j \in \{1, \dots, d\}$; then we set $\bar{x}_v = 1$ if and only if $\sigma(l(v)) < j$ or $(\sigma(l(v)) = j$ and $a_{l(v)} \leq x_v)$, which happens with probability $\frac{j-1}{d} + \frac{1}{d}(x_v - \frac{2(j-1)}{d^2})\frac{d^2}{2} = \frac{d}{2} x_v$.

Hence, the expected cost $\mathbb{E}[\sum_{v \in V} c_v \cdot \bar{x}_v]$ is at most $\frac{d}{2} \sum_{v \in V} c_v \cdot x_v$. $\square$

Now we derandomize this algorithm and show how to implement it in polynomial time.

Theorem 5.18. *Let x be a fractional vertex cover in a d-partite hypergraph with given d-partition. There is a deterministic algorithm that computes an integral solution $\bar{x}$ of cost $\sum_{v \in V} c_v \cdot \bar{x}_v \leq \frac{d}{2} \sum_{v \in V} c_v \cdot x_v$ in time $\mathcal{O}(n^3)$.*

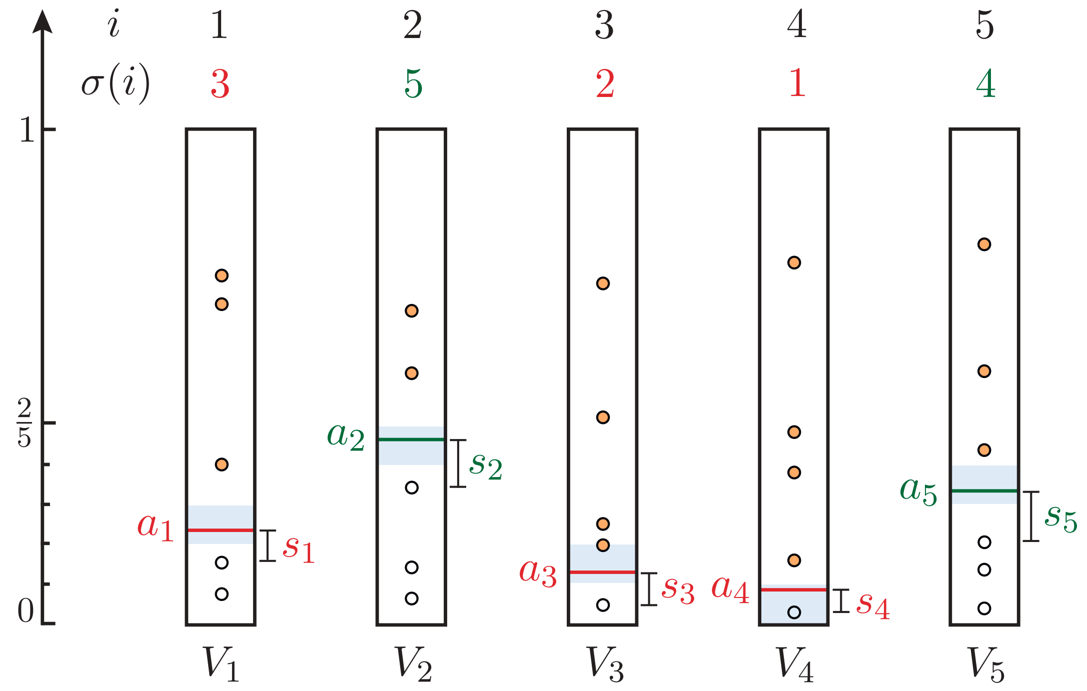


Figure 5.3: A sketch of thresholds $a_1, \dots, a_5$ chosen by our randomized algorithm in Theorem 5.17 for the case $d = 5$. The circles represent vertices in the hypergraph, drawn by their position in the partition and the value of their corresponding variable in the LP. Suppose the permutation $(\sigma(1), \dots, \sigma(5)) = (3, 5, 2, 1, 4)$ is chosen. Then the thresholds a_i are randomly chosen in the light blue intervals $\left[\frac{2(\sigma(i)-1)}{d^2}, \frac{2\sigma(i)}{d^2}\right]$; moreover, the thresholds a_1, a_3, a_4 are chosen independently of the thresholds a_2, a_5, as indicated by their color. The points above the thresholds are filled; these variables are rounded up to 1, while the empty circles represent variables that are rounded down to 0. Finally, the figure also shows "slack" values $s_1, \dots, s_5$, telling how much each threshold could be lowered without changing the solution returned by our algorithm. These will play a key role to improve the approximation guarantee in Theorem 5.19. Figure reprinted from our publication [DHV20].

Proof For a fixed value $\sigma(i) = j$ and a random choice of $a_i \in \left[\frac{2(j-1)}{d^2}, \frac{2j}{d^2}\right]$ we have the expected cost

$$\mathbb{E}\Big[\bar{x}_v \mid \sigma(i) = j\Big] = \begin{cases} 0, & \text{if } x_v < \frac{2(j-1)}{d^2} \\ x_v - \frac{2(j-1)}{d^2} \cdot \frac{d^2}{2}, & \text{if } x_v \in \left[\frac{2(j-1)}{d^2}, \frac{2j}{d^2}\right] \\ 1 & \text{if } x_v > \frac{2j}{d^2} \end{cases}$$

Let $\rho(i,j) := \sum_{v \in V_i} c_v \cdot \mathbb{E}\Big[\bar{x}_v \mid \sigma(i) = j\Big]$ be the total expected cost of layer i if we assign $\sigma(i) = j$ in the random permutation. We compute a permutation σ

that minimizes the total expected cost $\sum_{i=1}^{d} \rho(i, \sigma(i))$; this is a minimum-cost perfect matching problem in a complete bipartite graph with $d + d$ vertices. Hence this step can be implemented with a running time of $\mathcal{O}(d^3)$ [EK72; Tom71].

Therefore, we may now assume that the permutation σ is fixed. The probability distribution described in Lemma 5.16 chooses the values a_i for $i \in \{1, \dots, d\}$ independently for groups of two or three layers, with fixed sum $S_I := \sum_{i \in I} a_i$ for each such group I. Setting $a'_i = \max\{x_v : v \in V_i,\ x_v < a_i\}$, we see that the result in group I depends only on the numbers a'_i $(i \in I)$ and that there are less than n^3 possibilities. Among all choices of the a'_i $(i \in I)$ with $\sum_{i \in I} a'_i < S_I$, we can thus choose an optimum one (with minimum $\sum_{i \in I} \sum_{v \in V_i : x_v > a'_i} c_v$) in $\mathcal{O}(n^3)$ time. $\square$

It is easy to improve the running time in Theorem 5.18 to $\mathcal{O}(d^3 + n^2/d^2)$, but this is not important since already for time-cost tradeoff instances solving the LP dominates the overall running time of our approximation algorithm.

Since the vertex cover LP can be solved only approximately (Proposition 5.8), this would only yield an approximation ratio of $\frac{d}{2} + \epsilon$ for the time-cost tradeoff problem (unless d is fixed). In order to obtain a true $\frac{d}{2}$-approximation algorithm (and thus prove Theorem 5.1), we need a slightly stronger bound, which we derive next. Figure 5.1 shows that the integrality gap is at least $\frac{d}{2} - O(\frac{d}{n})$, In the following we will match this lower bound. Again, we first describe an improved randomized algorithm and then derandomize it.

Theorem 5.19. *Let $d \geq 4$. Let x be a fractional vertex cover in a d-partite hypergraph with given d-partition. There is a randomized linear-time algorithm that computes an integral solution $\bar{x}$ of expected cost $\sum_{v \in V} c_v \cdot \bar{x}_v \leq (\frac{d}{2} - \frac{d}{64n}) \sum_{v \in V} c_v \cdot x_v$.*

Proof First we choose the permutation σ and thresholds $a_1, \dots, a_d$ with $\sum_{i=1}^{d} a_i = 1$ randomly as above such that the thresholds are independent except within groups of two or three. For $i \in \{1, \dots, d\}$ denote the *slack* of level i by $s_i := \min\{\frac{1}{d}, a_i, a_i - \max\{x_v : v \in V_i,\ x_v < a_i\}\}$. Lowering the threshold a_i by less than s_i would yield the same solution $\bar{x}$. The reason for cutting off the slack at $\frac{1}{d}$ will become clear only below.

Next we randomly select one level $\lambda \in \{1, \dots, d\}$. Let Λ be the corresponding group (cf. Lemma 5.16), i.e., $\lambda \in \Lambda \subseteq \{1, \dots, d\}$, $|\Lambda| \leq 3$, and a_i is independent of a_λ whenever $i \notin \Lambda$. Now raise the threshold a_λ to $a'_\lambda = a_\lambda + \sum_{i \notin \Lambda} s_i$. Set $a'_i = a_i$ for $i \in \{1, \dots, d\} \setminus \{\lambda\}$.

As before, for all $v \in V$, set $\bar{x}_v := 1$ if $x_v \geq a'_{l(v)}$ and $\bar{x}_v := 0$ if $x_v < a'_{l(v)}$. We first observe that $\bar{x}$ is feasible. Indeed, if there were any hyperedge $P \in \mathcal{B}$ with $x_v < a'_{l(v)}$ for all $v \in P$, we would get $1 \leq \sum_{v \in P} x_v < \sum_{v \in P : l(v) \notin \Lambda} (a_{l(v)} -$

$s_{l(v)}) + \sum_{v\in P:l(v)\in\Lambda} a'_\lambda \le \sum_{i\notin\Lambda}(a_i - s_i) + \sum_{i\in\Lambda\setminus\{\lambda\}} a_i + a'_\lambda = \sum_{i=1}^d a_i = 1$, a contradiction.

We now bound the expected cost of $\bar{x}$. Let $v \in V$. With probability $\frac{d-1}{d}$ we have $l(v) \neq \lambda$ and, conditioned on this, an expectation $\mathbb{E}\big[\bar{x}_v \mid \lambda \neq l(v)\big] = \frac{d}{2}\min\{x_v, \frac{2}{d}\} \le \frac{d}{2}x_v$ as before. Now we condition on $l(v) = \lambda$ and in addition, for any S with $0 \le S \le \frac{d-2}{d}$, on $\sum_{i\notin\Lambda} s_i = S$; note that a_λ is independent of S. The probability that $\bar{x}_v$ is set to 1 is $\frac{d}{2}\max\{0, \min\{x_v - S, \frac{2}{d}\}\} \le \frac{x_v}{\frac{2}{d}+S} \le \frac{d}{2}(1-S)x_v$ in this case. In the last inequality we used $S \le \frac{d-2}{d}$, and this was the reason to cut off the slacks. In total we have for all $v \in V$:

$$\begin{aligned}
\mathbb{E}[\bar{x}_v] &= \frac{d-1}{d}\cdot\mathbb{E}[\bar{x}_v \mid \lambda \neq l(v)] \\
&\quad + \frac{1}{d}\cdot\int_0^{\frac{d-2}{d}} \mathbb{P}\Bigg[\sum_{i\notin\Lambda} s_i = S \mid \lambda = l(v)\Bigg]\cdot\mathbb{E}\Bigg[\bar{x}_v \mid \lambda = l(v), \sum_{i\notin\Lambda} s_i = S\Bigg]\,\mathrm{d}S \\
&\le \frac{d-1}{d}\cdot\frac{d}{2}x_v + \frac{1}{d}\cdot\int_0^{\frac{d-2}{d}} \mathbb{P}\Bigg[\sum_{i\notin\Lambda} s_i = S \mid \lambda = l(v)\Bigg]\cdot\frac{d}{2}(1-S)x_v\,\mathrm{d}S \\
&\le \frac{d}{2}\Bigg(1 - \frac{1}{d}\int_0^{\frac{d-2}{d}} \mathbb{P}\Bigg[\sum_{i\notin\Lambda} s_i = S \mid \lambda = l(v)\Bigg]\cdot S\,\mathrm{d}S\Bigg)\cdot x_v \\
&= \frac{d}{2}\Bigg(1 - \frac{1}{d}\cdot\mathbb{E}\Big[S \mid \lambda = l(v)\Big]\Bigg)x_v.
\end{aligned}$$

Let $\Lambda[v]$ be the the set Λ in the event $\lambda = l(v)$. We estimate

$$\mathbb{E}\Big[S \mid \lambda = l(v)\Big] = \sum_{i\notin\Lambda(v)} \mathbb{E}[s_i] \ge \sum_{i\notin\Lambda(v)} \frac{1}{d(n_i+1)} \ge \frac{(d-3)^2}{d(n+d)} \ge \frac{d}{32n}.$$

Here $n_i = |V_i|$, and the first inequality holds because $\mathbb{E}[s_i]$ is maximal if $\{x_v : v \in V_i\} = \{\frac{2j}{d(n_i+1)} : j = 1, \dots, n_i\}$. We conclude $\mathbb{E}\Big[\sum_{v\in V} c_v \cdot \bar{x}_v\Big] \le (\frac{d}{2} - \frac{d}{64n})\sum_{v\in V} c_v \cdot x_v$. □

Let us now derandomize this algorithm. This is easier than before because we can afford to lose a little again.

Theorem 5.20. *Let $d \ge 4$. Let x be a fractional vertex cover in a d-partite hypergraph with given d-partition. There is a deterministic algorithm that computes an integral solution $\bar{x}$ of cost $\sum_{v\in V} c_v \cdot \bar{x}_v \le (\frac{d}{2} - \frac{d}{65n})\sum_{v\in V} c_v \cdot x_v$ in runtime $\mathcal{O}(n^3 d)$.*

Proof We will argue that we can approximately compute the best possible choice for values a_i where $i \in \{1, \dots, d\}$, such that $\sum_{j=1}^d a_j \le 1$ and $\sum_{j=1}^d \sum_{v\in V_j, x_v \ge a_j} c_v$ is minimized. As our randomized algorithm computes a solution of this form, the optimum solution to this problem cannot exceed the expected cost of the randomized solution.

Fix some $\epsilon' > 0$. Let $\psi = \frac{\epsilon'}{n}(\frac{d}{2} - \frac{d}{64n})\sum_{v\in V} c_v \cdot x_v$. By scaling the costs we can assume that $\psi \geq 1$. We define rounded costs $c'_v = \left\lfloor \frac{c_v}{\psi} \right\rfloor$ for $v \in V$. Now, iterating increasingly on values $i = 1, \dots, d$ and for every $\alpha \in \{0, \dots, \left\lceil \frac{2n}{\epsilon'} \right\rceil\}$ we store the solution of objective $\sum_{j=1}^{i} \sum_{v\in V_j, x_v \geq a_j} c'_v$ at most α minimizing $\sum_{j=1}^{i} a_j$. With dynamic programming, this can be done in time $\mathcal{O}(dn^2\epsilon'^{-1})$. At the end we look up the cheapest computed solution with $\sum_{j=1}^{d} a_j \leq 1$.

It remains to analyze the cost of this solution. Let $c^* = (\frac{d}{2} - \frac{d}{64n})\sum_{v\in V} c_v \cdot x_v$ be the upper bound on the expected cost of the solution by our randomized algorithm. Let COST' denote the cost of the solution computed by our dynamic program with respect to the cost function $c'_v \cdot \psi$ for $v \in V$, i.e. COST' $= \sum_{j=1}^{d} \sum_{v\in V_j, x_v\geq a_j} \psi c'_v$. As the process of rounding only decreased costs, we have COST' $\leq c^*$. Note, that for all $v \in V$ we have

$$c_v - \psi c'_v = c_v - \psi \left\lfloor \frac{c_v}{\psi} \right\rfloor \leq c_v - \psi \left(\frac{c_v}{\psi} - 1 \right) = \psi \leq \frac{\epsilon'}{n} c^*.$$

Let COST denote the cost of the solution computed by our dynamic program with respect to the original cost function c_v for $v \in V$, i.e. COST $= \sum_{j=1}^{d} \sum_{v\in V_j, x_v \geq a_j} c_v$.

By the above computation we have COST $\leq$ COST' $+ n \cdot \frac{\epsilon'}{n} c^* \leq (1+\epsilon')c^*$. Putting everything together we obtain a solution $\bar{x}_v$ of cost $\sum_{v\in V} c_v \cdot \bar{x}_v \leq (1+\epsilon')(\frac{d}{2} - \frac{d}{64n})\sum_{v\in V} c_v \cdot x_v$. By letting $\epsilon' = \frac{1}{65\cdot 32n} = \frac{1}{2080n}$ we obtain a solution of cost at most $(\frac{d}{2} - \frac{d}{65n})\sum_{v\in V} c_v \cdot x_v$.

□

As explained above, together with Propositions 5.6 and 5.8 (with $\epsilon = \frac{1}{65n}$), Theorem 5.20 implies Theorem 5.1.

5.5 Inapproximability

Gutswami, Sachdeva and Saket [GSS00] proved that approximating the vertex cover problem in d-partite hypergraphs with a better ratio than $\frac{d}{2}$ is NP-hard under the Unique Games Conjecture. We show that even for the special case of time-cost tradeoff instances, the problem is hard to approximate by a factor of $\frac{d+2}{4}$.

Let us briefly sketch a technique of [GSS00] and explain why it does not serve our purpose. Let $d \geq k \geq 2$ be integers. One can reduce the vertex cover problem in k-uniform hypergraphs, i.e., for hypergraphs $H = (U, F)$ such that $|e| = k$ for all $e \in F$, to the d-partite case. The idea is to take d disjoint copies of the vertex set U as the vertex set of a new hypergraph G.

For every hyperedge $e \in F$, the hypergraph G contains all hyperedges e' that contain exactly one copy of every vertex in e and at most one vertex of any of the d copies of U. Clearly this hypergraph G is d-partite. It is easy to see that any optimal solution in G must contain either no or at least $d-k+1$ of the copies of a vertex and there is always a vertex cover of size dOPT, where OPT denotes the size of an optimum vertex cover in H. By a result of Khot and Regev [KR08], the vertex cover problem in k-uniform hypergraphs is NP-hard to approximate with a factor of $k-\epsilon$ under the Unique Games Conjecture. Therefore, for any $d \geq 4$, by letting $k = \lceil \frac{d+1}{2} \rceil$, it is easy to see that we obtain a d-partite hypergraph vertex-cover instance, which does not admit a $\frac{d}{4}$-approximation. However, G does certainly not represent a time-cost tradeoff instance.

In this and the next section, we will show Theorem 5.2, which is our main inapproximability result. Insetad of starting from k-uniform hypergraphs, we devise a reduction from the vertex deletion problem in acyclic digraphs, which Svensson [Sve12] called DVD. Let k be a positive integer; then DVD(k) is defined as follows: given an acyclic digraph, compute a minimum-cardinality set of vertices whose deletion destroys all paths with k vertices.

We remark that an undirected version of this problem has been called *k-path vertex cover*[Bre+11] or *vertex cover P_k* [TZ11]. Both the DVD(k) and the k-path vertex cover problem are interesting on their own and have direct applications in chip design or network construction [Ach+12; PRS94].

The DVD(k) problem is easily seen to admit a k-approximation algorithm:

Lemma 5.21. *For all $k \geq 1$,* DVD(k) *admits a k-approximation algorithm.*

Proof Find a maximal set of vertex-disjoint paths, each with k vertices, and take the set of all their vertices. □

Svensson proved that anything better than this simple approximation algorithm would solve the unique games problem:

Theorem 5.22 ([Sve12])**.** *Assuming the Unique Games Conjecture, for any integer $r \geq 2$ and arbitrary constant $\epsilon > 0$ the following problem is NP-hard: Given a directly acyclic graph $G = (V, E)$, distinguish between the following cases:*

- *Completeness: There are disjoint subsets $V_1, \ldots, V_r \subset V$ satisfying $|V_i| \geq \frac{1-\epsilon}{r}|V|$ and every subgraph induced by all but one of these subsets has no path of $\geq r$ vertices.*
- *Soundness: Every induced subgraph of $\epsilon|V|$ vertices has a path of $\geq |V|^{1-\epsilon}$ vertices.*

This implies the following Lemma.

Lemma 5.23. *Let $k \in \mathbb{N}$ with $k \geq 2$ and $\rho < k$ be constants. Let OPT denote the size of an optimum solution for a given* DVD(k) *instance. Assuming the Unique Games Conjecture it is NP-hard to compute a number $l \in \mathbb{R}_+$ such that $l \leq OPT \leq \rho l$.*

Proof Assume by contradiction that we can compute a $\rho = k - \gamma$ approximation for some $\gamma > 0$. Take an NP-hard instance of Theorem 5.22 with $r = k$ and ϵ small enough, such that $|V|^{1-\epsilon} \geq k$. (We can assume that $|V|$ has at least an arbitrary minimum size as smaller instances can be decided by enumeration). Also, choose ϵ small enough, such that $(1-\epsilon)/(\frac{1}{r}+\epsilon) \geq r - \gamma$. This is possible as $\lim_{\epsilon \to 0}(1-\epsilon)/(\frac{1}{r}+\epsilon) = r$.

If we are in the Soundness case, every subgraph $G[Y]$ with $|Y| \leq \epsilon|V|$ has a path of $|V|^{1-\epsilon} \geq k$ vertices. Therefore, the optimum solution for our vertex deletion problem on the same underlying graph has cost OPT $\geq (1-\epsilon)|V|$.

For the Completeness case, we can find disjoint sets $V_1, \ldots, V_r \subset V$ such that $G[(V_1 \cup \ldots \cup V_r) \setminus V_i]$ has no path of $r = k$ vertices and $|V_i| \geq \frac{1-\epsilon}{r}|V|$ for all i. Note that for $V_\epsilon := V \setminus (V_1 \cup \ldots, V_r)$ we have

$$|V_\epsilon| = |V| - \sum_{i=1}^{r} |V_i| \leq |V| - \sum_{i=1}^{r} \frac{1-\epsilon}{r}|V| = \epsilon|V|.$$

Therefore, the set $X = V_1 \cup V_\epsilon$ satisfies $|X| \leq \frac{1-\epsilon}{r}|V| + \epsilon|V|$ and $G[V \setminus X] = G[(V_1 \cup \ldots \cup V_r) \setminus V_1]$ has no path of $r = k$ vertices. This means that there is a solution to the vertex deletion problem of cost OPT $\leq (\frac{1}{r}+\epsilon)|V|$.

However as $(1-\epsilon)/(\frac{1}{r}+\epsilon) \geq r - \gamma = k - \gamma$ we can distinguish Completeness and Soundness. □

This Lemma is the starting point of our proof. Svensson [Sve12] already observed that DVD(k) can be regarded as a special case of the time-cost tradeoff problem. Note that this does not imply Theorem 5.2 because the hard instances of DVD(k) constructed in the proof of Theorem 5.23 have unbounded depth even for fixed k. Recall that the *depth* of an acyclic digraph is the number of vertices in a longest path. The following is a variant (and slight strengthening) of Svensson's observation.

Lemma 5.24. *Any instance of* DVD(k) *(for any k) can be transformed in linear time to an equivalent instance of the time-cost tradeoff problem, with the same depth and the same optimum value.*

Proof Let $G = (V, E)$ be an instance of DVD(k), an acyclic digraph, say of depth d. Let $l(v) \in \{1, \ldots, d\}$ for $v \in V$ such that $l(v) < l(w)$ for all

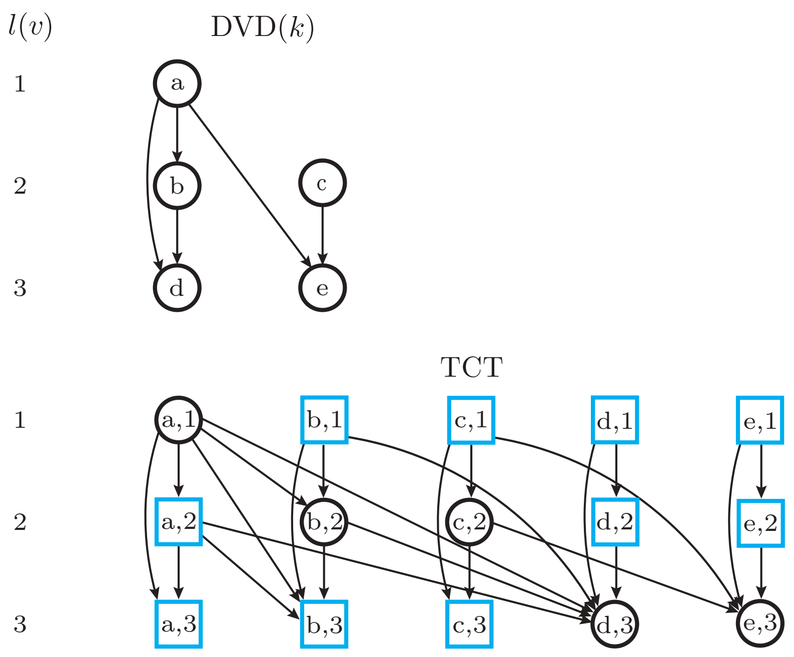


Figure 5.4: The transformation of Lemma 5.24. An instance of DVD(k) is transformed into an equivalent instance of the time-cost tradeoff problem. Jobs with fixed execution time are depicted as blue squares. Figure reprinted from our publication [DHV20].

$(v, w) \in E$. Let $J := \{(v, i) : v \in V, i \in \{1, \dots, d\}\}$ be the set of jobs of our time-cost tradeoff instance. Job (v, i) must precede job (w, j) if ($v = w$ and $i < j$) or ($(v, w) \in E$ and $l(v) \leq i < j$). Let $\prec$ be the transitive closure of these precedence constraints. For $v \in V$, the job $(v, l(v))$ is called *variable* and has a fast execution time 0 at cost 1 and a slow execution time $d + 1$ at cost 0. All other jobs are *fixed*; they have a fixed execution time d at cost 0. The deadline is $d^2 + k - 1$. A sketch of this construction is given in Figure 5.4.

We claim that any set of variable jobs whose acceleration constitutes a feasible solution of this time-cost tradeoff instance corresponds to a set of vertices whose deletion destroys all paths in G with k vertices, and vice versa. Indeed, the total delay of a chain in the time-cost tradeoff instance is at most $(d - 1)(d + 1)$ unless the chain contains a job in each level and contains no variable job that is accelerated, in which case the total delay is $d^2 + j$, where j is the number of variable jobs in the chain. These chains with total delay $d^2 + j$ correspond to the paths with j vertices in G. □

Therefore a hardness result for DVD(k) for bounded depth instances transfers to a hardness result for the time-cost tradeoff problem with bounded depth. We will show the following strengthening of Theorem 5.23:

Theorem 5.25. *Let $k, d \in \mathbb{N}$ with $2 \leq k \leq d$ and $\rho < \frac{k(d+1-k)}{d}$ be constants. Let OPT denote the size of an optimum solution for a given* DVD(k) *instance. Assuming the Unique Games Conjecture it is NP-hard to compute a number $l \in \mathbb{R}_+$ such that $l \leq OPT \leq \rho l$.*

It is easy to see that Theorem 5.25 and Lemma 5.24 imply Theorem 5.2. Indeed, let $d \in \mathbb{N}$ with $d \geq 2$ and $\rho < \frac{d+2}{4}$, and suppose that a ρ-approximation algorithm $\mathcal{A}$ exists for time-cost tradeoff instances of depth d. Let $k := \lceil \frac{d+1}{2} \rceil$ and consider an instance of DVD(k) with depth d. Transform this instance to an equivalent time-cost tradeoff instance by Lemma 5.24 and apply algorithm $\mathcal{A}$. This constitutes a ρ-approximation algorithm for DVD(k) with depth d. Since $\rho < \frac{d+2}{4} \leq \frac{k(d+1-k)}{d}$, Theorem 5.25 then implies that the Unique Games Conjecture is false or P = NP.[1]

It remains to prove Theorem 5.25, which will be the subject of the next section.

5.6 Reducing Vertex Deletion to Constant Depth

In this section we prove Theorem 5.25. The idea is to reduce the depth of a digraph by transforming it to another digraph with small depth but related vertex deletion number. Let $k, d \in \mathbb{N}$ with $2 \leq k \leq d$, and let G be a digraph. We construct an acyclic digraph G^d of depth at most d by taking the tensor product with the acyclic tournament on d vertices: $G^d = (V^d, E^d)$, where $V^d = V \times \{1, \ldots, d\}$ and $E^d = \{((v,i),(w,j)) : (v,w) \in E \text{ and } i < j\}$. It is obvious that G^d has depth d. An example of this construction is depicted in Figure 5.5. Here is our key lemma:

Lemma 5.26. *Let G be an acyclic directed graph and $k, d \in \mathbb{N}$ with $2 \leq k \leq d$. If we denote by $OPT(G,k)$ the minimum number of vertices of G hitting all paths with k vertices, then*

$$(d+1-k) \cdot OPT(G,k) \;\leq\; OPT(G^d,k) \;\leq\; d \cdot OPT(G,k). \tag{5.5}$$

Lemma 5.26, together with Theorem 5.23, immediately implies Theorem 5.25: assuming a ρ-approximation algorithm for DVD(k) instances with depth d, with $\rho < \frac{k(d+1-k)}{d}$, we can compute OPT($G,k$) up to a factor less than k for

[1]In fact, this proof shows that the threshold in Theorem 5.2 can be taken $\frac{1}{4d}$ larger for odd d; e.g., there is no ρ-approximation algorithm for $\rho < \frac{4}{3}$ for $d = 3$.

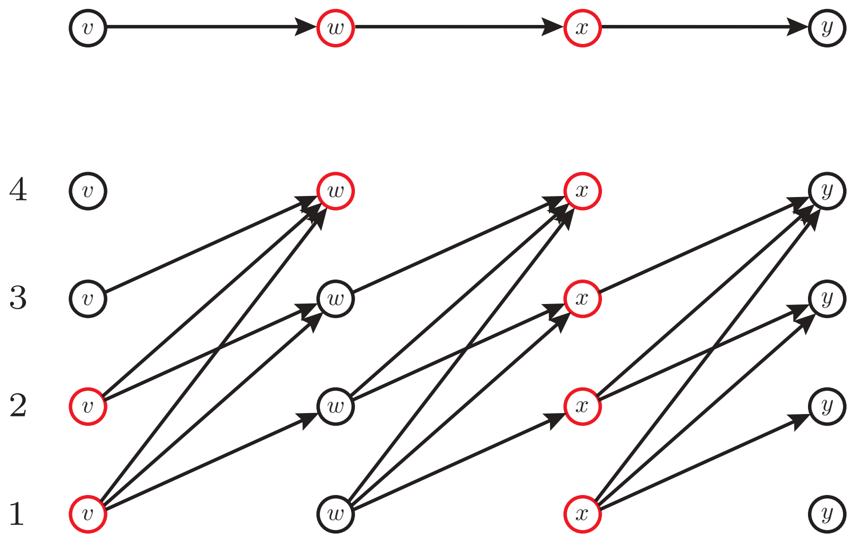


Figure 5.5: A directed path P_4 and the graph tensor product with the acyclic tournament on 4 vertices. The colored vertices show a solution to the vertex deletion problems with $k = 2$. Figure reprinted from our publication [DHV20].

any digraph G. By Theorem 5.23, this would contradict the Unique Games Conjecture or P $\neq$ NP.

Before we prove Lemma 5.26, let us give two examples which show that the bounds in (5.5) are sharp for all d and k, for infinitely many acyclic digraphs.

For the lower bound, consider the acyclic tournament D_n on the vertices $1, \ldots, n$. Obviously, $\mathrm{OPT}(D_n, k) = n - k + 1$. Moreover, $\mathrm{OPT}(D_n^d, k) \leq (d+1-k)(n-k+1)$ because $\{(i,j) : i = 1, \ldots, n-k+1,\ j = 1, \ldots, d+1-k\}$ is a feasible solution for $\mathrm{DVD}(D_n^d, k)$.

For the upper bound, consider the directed path P_n on the vertices $1, \ldots, n$, where $n = (r+1)k - 1$ for some $r \in \mathbb{N}$. Obviously $\mathrm{OPT}(P_n, k) = r$ because $\{k, 2k, \ldots, rk\}$ is a feasible solution. To show $\mathrm{OPT}(P_n^d, k) \geq rd$, we find rd vertex-disjoint paths in P_n^d, each with k vertices: for $i = 1, \ldots, r$ and $j = 1, \ldots, d$, the vertex set of the $(di - d + j)$-th path arises from $\{(ki, j), (ki+1, j+1), \ldots, (ki+k-1, j+k-1)\}$ by replacing $(s, d+t)$ by $(s-k+t, t)$ for all $s, t \geq 1$. See Figure 5.6.

We remark that the left inequality in (5.5) holds also for general (not necessarily acyclic) digraphs. However, for general digraphs it may be that $\mathrm{OPT}(G^d, k) > d \cdot \mathrm{OPT}(G, k)$.

Finally, we prove Lemma 5.26.

Proof (Lemma 5.26) Let G be an acyclic digraph. The upper bound of (5.5) is trivial: for any set $W \subseteq V$ that hits all k-vertex paths in G we can take $X := W \times \{1, \ldots, d\}$ to obtain a solution to the $\mathrm{DVD}(k)$ instance G^d.

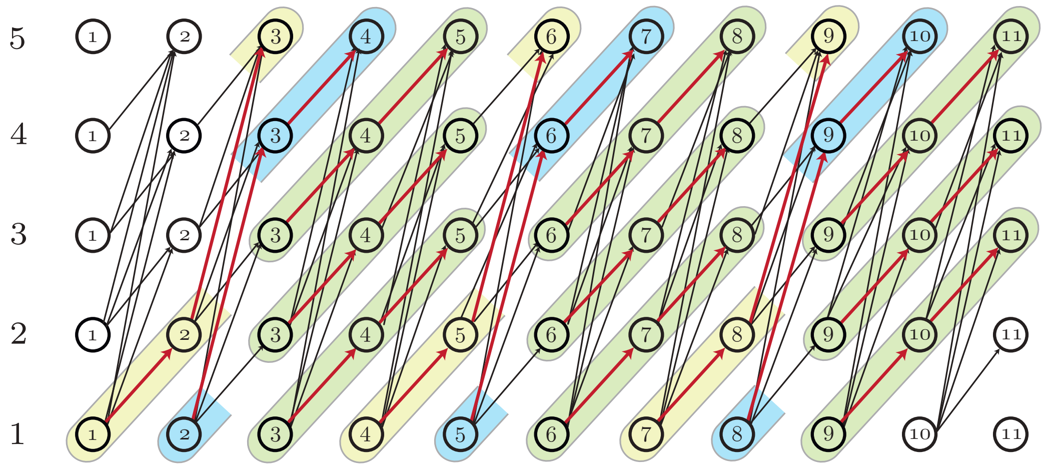


Figure 5.6: Construction of rd vertex-disjoint paths, each with k vertices, in $P^d_{(r+1)k-1}$ for $r = 3$, $d = 5$, and $k = 3$. The edge sets corresponding to paths are highlighted in red. Figure reprinted from our publication [DHV20].

To show the lower bound, we fix a minimal solution X to the DVD(k) instance G^d. Let Q be a path in G^d with at most k vertices. We write $\text{start}(Q) = i$ if Q begins in a vertex (v, i). We define $\mathcal{Q}$ as the set of paths in G^d with exactly k vertices. For $Q \in \mathcal{Q}$ let $\text{lasthit}(Q)$ denote the last vertex of Q that belongs to X. For $x \in X$ we define

$$\varphi(x) := \max\{\text{start}(Q) : Q \in \mathcal{Q}, \text{lasthit}(Q) = x\}.$$

Note that this is well-defined due to the minimality of X, and $1 \leq \varphi(x) \leq d + 1 - k$ for all $x \in X$.

We will show that for $j = 1, \ldots, d + 1 - k$,

$$S_j := \{v \in V : (v, i) \in X \text{ and } \varphi((v, i)) = j \text{ for some } i \in \{1, \ldots, d\}\}$$

hits all k-vertex paths in G. This shows the lower bound in (5.5) because then $\text{OPT}(G, k) \leq \min_{j=1}^{d+1-k} |S_j| \leq \frac{|X|}{d+1-k}$.

Let P be a path in G with k vertices $v_1, \ldots, v_k$ in this order. Consider d "diagonal" copies $D_1, \ldots, D_d$ of (suffixes of) P in G^d: the path D_i consists of the vertices $(v_s, s + i - k), \ldots, (v_k, i)$, where $s = \max\{1, k + 1 - i\}$. Note that the paths $D_1, \ldots, D_{k-1}$ have fewer than k vertices.

We show that for each $j = 1, \ldots, d + 1 - k$, at least one of these diagonal paths contains a vertex $x \in X$ with $\varphi(x) = j$. This implies that $S_j \cap P \neq \varnothing$ and concludes the proof.

First, D_d contains a vertex in $x \in X$ with $\varphi(x) = d + 1 - k$, namely $\text{lasthit}(D_d)$. Now we show for $i = 1, \ldots, d - 1$ and $j = 1, \ldots, d - k$:

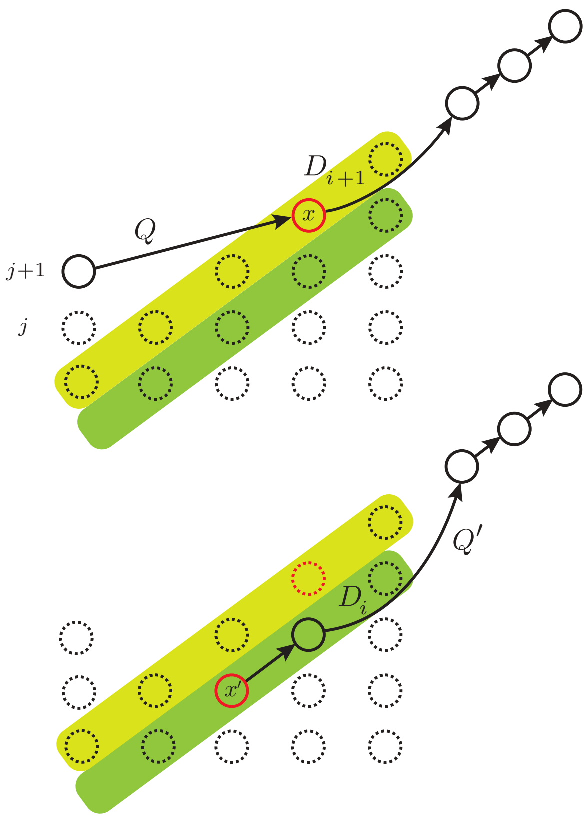


Figure 5.7: A visualization of the proof idea of the central Claim in the proof of Lemma 5.26. The Claim asserts that if D_{i+1} contains a vertex $x \in X$ with $\varphi(x) = j+1$, then D_i contains a vertex $x' \in X$ with $\varphi(x') \geq j$. The upper diagonal D_{i+1} is colored in light green, the lower diagonal D_i is depicted in dark green. We start by selecting a path Q with $\text{lasthit}(Q) \in D_{i+1}$ and $\text{start}(Q) = j+1$. This path is depicted on the left; the vertex $x = \text{lasthit}(Q)$ is highlighted in red. We construct a path Q' (shown on the right) such that $x' = \text{lasthit}(Q') \in D_i$ and $\text{start}(Q') = \text{start}(Q) - 1$. This path Q' results from appending the end of path Q to an appropriate subpath of the next lower diagonal D_i. Figure reprinted from our publication [DHV20].

Claim: If D_{i+1} contains a vertex $x \in X$ with $\varphi(x) = j+1$, then D_i contains a vertex $x' \in X$ with $\varphi(x') \geq j$.

This Claim implies the theorem because D_1 consists of a single vertex $(v_k, 1)$, and if it belongs to X, then $\varphi((v_k, 1)) = 1$.

To prove the Claim (see Figure 5.7 for an illustration), let $x = (v_h, l(x)) \in X \cap D_{i+1}$ and $\varphi(x) \geq j+1$, and let x be the last such vertex on D_{i+1}. We have $\varphi(x) \geq \text{start}(D_{i+1})$ for otherwise we have $\text{start}(D_{i+1}) > 1$, so D_{i+1} contains k vertices and we should have chosen $x = \text{lasthit}(D_{i+1})$; note that $\varphi(\text{lasthit}(D_{i+1})) \geq \text{start}(D_{i+1})$.

Let $Q \in \mathcal{Q}$ be a path attaining the maximum in the definition of $\varphi(x)$. So $\text{start}(Q) = \varphi(x)$ and $\text{lasthit}(Q) = x$. Suppose x is the p-th vertex of Q; note that

$$p \;\le\; 1 + l(x) - \varphi(x) \tag{5.6}$$

because Q starts on level $\varphi(x)$, rises at least one level with every vertex, and reaches level $l(x)$ at its p-th vertex.

Now consider the following path Q'. It begins with part of the diagonal D_i, namely $(v_{h+1-p}, l(x) - p), \ldots, (v_h, l(x) - 1)$, and continues with the $k - p$ vertices from the part of Q after x. Note that by (5.6)

$$l(x) - p \ge \varphi(x) - 1 \ge \max\{j, \text{start}(D_{i+1}) - 1\} \ge \max\{1, \text{start}(D_i)\},$$

so Q' is well-defined.

The second part of Q' does not contain any vertex from X because $\text{lasthit}(Q) = x$. Hence $x' := \text{lasthit}(Q')$ is in the diagonal part of Q', i.e., in D_i. By definition, $\varphi(x') \ge \text{start}(Q') = l(x) - p \ge j$. □

5.7 Variants of the Time-Cost Tradeoff Problem

Instead of obeying a strict delay bound in the time-cost tradeoff problem, one can introduce penalties $\alpha, \beta \ge 0$ and to look for a solution which minimizes the weighted sum of penalizing the longest path by α and the total size by β. This problem formulation can arise for example in global technology mapping. More formally, consider the following problem definition.

Problem 5: (α, β)-delay penalty time-cost tradeoff problem

Input: An acyclic graph $G = (V, E)$, edge delays $d_e \in \mathbb{R}_{\ge 0}$ for $e \in E$.
For every $v \in V(G)$ a set of alternative vertex delays $\mathcal{A}_v = \{(d_{v,1}, c_{v,1}), \ldots, (d_{v,v_e})\}$.

Task: Compute a map $\pi : V \to \mathbb{N}$ such that for every $v \in V(G) : \pi(v) \in \{1, \ldots, k_v\}$. The cost we want to minimize is $\alpha \max_{P \in \mathcal{P}} \left(\sum_{v \in V(P)} d_{v,\pi(v)} + \sum_{e \in E(P)} d_e\right) + \beta \sum_{v \in V(G)} c_{v,\pi(v)}$. Here, $\mathcal{P}$ denotes the set of inclusion-wise maximal paths in G.

On the positive side, we will present an approximation algorithm for the problem.

Theorem 5.27. *For every $\epsilon > 0$ there is a polynomial $2 + \epsilon$ approximation for the (α, β)-delay penalty time-cost tradeoff problem.*

Proof We will use the following result by Skutella [Sku98]: For every $\mu \in (0, 1)$ there is a bicriteria approximation algorithm A_μ for the regular time-cost tradeoff problem that returns a solution which exceeds the deadline by at most $\frac{1}{\mu}$ and the cost of the optimum solution by at most $\frac{1}{1-\mu}$.

Note that the original proof by Skutella used a slightly different problem formulation in which we choose different realization for edges instead of vertices. However, the simple proof of the bicriteria algorithm can easily be adapted for the vertex variant. Indeed we simply have to solve the linear relaxation of the problem and round up all vertex variables of lp value at least $1 - \mu$ and otherwise round down. This clearly increases the cost at most by a factor of $\frac{1}{1-\mu}$ and the delay of any vertex and thereby any path by at most $\frac{1}{\mu}$.

Let $D^\downarrow$ be the delay of the solution where every vertex is set to the slowest available variant, and $D^\uparrow$ the delay of the solution where every vertex is fast. Note that $\log(D^\downarrow/D^\uparrow)$ is polynomial in the size of the input. Similarly let $C^\downarrow$ be the cheapest and $C^\uparrow$ the most expensive solution.

Our algorithm will try all possible deadlines $D_h = D^\uparrow \cdot (1 + \epsilon)^h$, $h \in \mathbb{N}$ such that $D_h \leq (1 + \epsilon)D^\downarrow$ and return the best encountered solution. And for every D_h iterate over all possible costs $C_j = C^\downarrow \cdot (1 + \epsilon)^j$, $j \in \mathbb{N}$ such that $C_j \leq (1 + \epsilon)C^\uparrow$.

Note that we can enumerate these pairs in $\mathcal{O}(\log(D^\downarrow/D^\uparrow) \log(C^\uparrow/C^\downarrow)/\epsilon^2)$.

Assume that the optimum solution to our problem is given by $\mathrm{OPT} = \alpha D^\star + \beta C^\star$. Here $D^\star$ is the deadline and $C^\star$ the total cost. Eventually we will try some deadline budget pair (D_i, C_j) such that $D_i \in [D^\star, (1 + \epsilon)D^\star]$ and $C_j \in [C^\star, (1 + \epsilon)C^\star]$. For ease of notation, we write $\widehat{D} = D_i, \widehat{C} = C_j$.

Consider the instance of the TCT problem with deadline $\widehat{D}$. As $\widehat{D} \geq D^\star$ we know that $\mathrm{OPT}_{\mathrm{TCT}} \leq C^\star \leq \widehat{C}$. If we can find a solution to the (α, β)-delay penalty time-cost tradeoff problem with cost $C_A \leq 2(\alpha\widehat{D} + \beta\widehat{C})$ we are done as $(\alpha\widehat{D} + \beta\widehat{C}) \leq (\alpha(1 + \epsilon)D^\star + \beta(1 + \epsilon)C^\star) = (1 + \epsilon)\mathrm{OPT}$.

We further simplify notation by $\widehat{\alpha} := \alpha\widehat{D}, \widehat{\beta} := \beta\widehat{C}$. Let $\widehat{\gamma} = \sqrt{\widehat{\alpha} \cdot \widehat{\beta}}$ be the geometric mean of $\widehat{\alpha}$ and $\widehat{\beta}$. By basic properties of the geometric mean $\min(\widehat{\alpha}, \widehat{\beta}) \leq \widehat{\gamma} \leq \widehat{\alpha} + \widehat{\beta} \leq \max(\widehat{\alpha}, \widehat{\beta})$.

We define
$$\mu = \frac{\widehat{\alpha}}{\widehat{\alpha} + \widehat{\gamma}}.$$

Therefore we have
$$1 - \mu = \frac{\widehat{\gamma}}{\widehat{\alpha} + \widehat{\gamma}}.$$

Obviously, $0 < \mu < 1$ and therefore we may apply the bicriteria algorithm by Skutella. This yields a solution that may exceed the deadline $\widehat{D}$ by at most $\frac{1}{\mu}$ and the optimum cost $\mathrm{OPT}_{\mathrm{TCT}}$ by at most $\frac{1}{1-\mu}$. This corresponds to a solution of total cost:

$$\begin{aligned}
C_A =& \alpha\frac{1}{\mu}\widehat{D} + \beta\frac{1}{1-\mu}\mathrm{OPT}_{\mathrm{TCT}} \leq \widehat{\alpha}\frac{1}{\mu} + \beta\frac{1}{1-\mu}\widehat{C} = \widehat{\alpha}\frac{1}{\mu} + \widehat{\beta}\frac{1}{1-\mu} = \\
=& \widehat{\alpha} + \widehat{\gamma} + \widehat{\beta}\frac{\widehat{\alpha}+\widehat{\gamma}}{\widehat{\gamma}} = \frac{(\widehat{\alpha}+\widehat{\gamma})\big(\widehat{\beta}+\widehat{\gamma}\big)}{\widehat{\gamma}} = \frac{\Big(\widehat{\alpha}+\sqrt{\widehat{\alpha}\widehat{\beta}}\Big)\Big(\widehat{\beta}+\sqrt{\widehat{\alpha}\widehat{\beta}}\Big)}{\sqrt{\widehat{\alpha}\widehat{\beta}}} \\
=& \left(\frac{\sqrt{\widehat{\alpha}\widehat{\beta}}}{\widehat{\beta}}+1\right)\Big(\widehat{\beta}+\sqrt{\widehat{\alpha}\widehat{\beta}}\Big) = \sqrt{\widehat{\alpha}\widehat{\beta}} + \widehat{\alpha} + \widehat{\beta} + \sqrt{\widehat{\alpha}\widehat{\beta}} = \widehat{\alpha} + \widehat{\beta} + 2\sqrt{\widehat{\alpha}\widehat{\beta}}
\end{aligned}$$

We are now able to bound the approximation ratio

$$\frac{C_A}{\alpha\widehat{D}+\beta\widehat{C}} \leq \frac{\widehat{\alpha}+\widehat{\beta}+2\sqrt{\widehat{\alpha}\widehat{\beta}}}{\widehat{\alpha}+\widehat{\beta}} = 1 + \frac{2\sqrt{\widehat{\alpha}\widehat{\beta}}}{\widehat{\alpha}+\widehat{\beta}} \leq 2.$$

Here the last inequality is the inequality of arithmetic and geometric means $\frac{x+y}{2} \geq \sqrt{xy}$ for $x, y \geq 0$. □

Now we want to show a lower bound for the approximability of the problem.

Theorem 5.28. *There is some $\rho > 0$ such that there is no $1+\rho$ approximation for the $(1,1)$-delay penalty time-cost tradeoff problem unless P=NP.*

Proof We closely follow a hardness proof for the discrete time-cost tradeoff problem by Deĭneko et al.[DG01]. The reduction goes from the vertex cover problem in cubic graphs which is APX hard as shown by Alimonti et al.[AK00].

Assume that $G = (V, E)$ is some 3-regular undirected graph; our task is to find a vertex cover in G. We may assume without loss of generality that G is connected as we can solve the vertex cover problem for the connected components separately. We can further assume that $|V(G)| \geq 5$ as we can solve small instances by enumeration. By applying the theorem of Brooks [Bro41], we see that G has a three-coloring which can be computed in polynomial time. Let $V = X\dot{\cup}Y\dot{\cup}Z$ be a partition of the vertex set induced by this three-coloring.

We construct an instance G' of the restricted discrete time-cost tradeoff problem as follows: We set $V(G') = X \cup Y' \cup Z$, where

$$Y' = \{y^1, y^2, y^3 : y \in Y\} = Y^1 \mathbin{\dot{\cup}} Y^2 \mathbin{\dot{\cup}} Y^3.$$

The vertices $x \in X$ have two alternatives $\mathcal{A}_x = \{(2,0),(0,1)\}$. Similarly, we introduce alternatives $\mathcal{A}_z = \{(2,0),(0,1)\}$ for all $z \in Z$ and for vertices $y^2 \in Y^2$ we also add alternatives $\mathcal{A}_{y^2} = \{(2,0),(0,1)\}$. Vertices $y^1, y^3 \in Y^1, Y^3$ have a fixed delay of 0. Now we add edges $e = (y^1, y^2)$ and edges $e = (y^2, y^3)$, both with fixed delay 1.

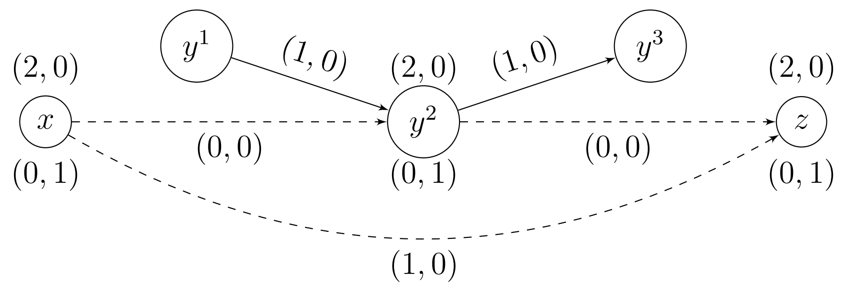


Figure 5.8: An excerpt of the constructed graph G'. Dashed edges are inserted if the corresponding edge is in $E(G)$. The descriptions indicate available delay cost pairs.

For every edge $\{x, y\} \in E(G)$ we add an edge (x, y^2) with delay 0. Similarly we add an edge (y^2, z) with delay 0 for every edge $\{y, z\} \in E(G)$. For every edge $\{x, z\} \in E(G)$ we add an edge (x, z) with delay 1 to G'. This completes the construction of G', in summary we have:

$$\begin{aligned} E(G') = \{(y^1, y^2), (y^2, y^3) : y \in Y\} \cup \{(x, y^2) : \{x, y\} \in E(G)\} \cup \\ \cup \{(y^2, z) : \{y, z\} \in E(G)\} \cup \{(x, z) : \{x, z\} \in E(G)\}. \end{aligned}$$

An illustration of the constructed graph is given in Figure 5.8.

Assume that we have a vertex cover W of size k in G. We claim that we find a feasible solution for the instance of the time-cost tradeoff problem arising from G', where $D = 4$ with the cost k. We set all vertices of the form $E_W = \{x : x \in W \cap X\} \cup \{y^2 : y \in W \cap Y\} \cup \{z : z \in W \cap Z\}$ to the fast alternative. Clearly, the cost of this solution will be k.

Assume that there is a path P in G' with longer delay than 4. There are only 4 possibilities for such a path:

1. A path of the form $(x, y^2), (y^2, y^3)$, where both x and y^2 use the slow alternative. In this case neither x nor y are in W but there is an edge $\{x, y\} \in E(G)$. Therefore W was not a vertex cover, contradicting our assumption.

2. The other three cases are analogous.

This shows that every vertex cover of size k yields a feasible solution of the time-cost tradeoff instance G' with deadline 4 of cost k.

We will now describe our instance of the $(1, 1)$-delay penalty time-cost tradeoff problem that corresponds to this vertex cover problem. First note that there is a simple polynomial 2-approximation for the vertex cover problem. Therefore we can compute some $\rho \in \mathbb{N}$ such that the optimum solution has cost $\mathrm{OPT} \in [\frac{\rho}{2}, \rho]$.

We change the edge costs such that adding a vertex into the cover incurs a cost of $\frac{3}{4\rho}$ instead of 1. Therefore we may assume OPT $\in [\frac{3}{8}, \frac{3}{4}]$ for the vertex cover instance.

By this transformation we may assume that every solution to the $(1,1)$-DPTCT problem respects the deadline, as otherwise the cost is $C_A \geq \alpha \cdot 5 = 5 > 4 + \frac{3}{4}$ which is the cost of our 2-approximation that has a deadline of 4.

Assume that we have a $\lambda > 1$ approximation for the $(1,1)$-DPTCT problem. We see that:

$$\begin{aligned} \frac{C_A}{\text{OPT}} &= \frac{4+\varphi\cdot\gamma}{4+\gamma} \leq \lambda \\ (\Leftrightarrow)\ \varphi &\leq \frac{\lambda(4+\gamma)-4}{\gamma}. \end{aligned}$$

As we have
$$\frac{\partial}{\partial\gamma}\frac{\lambda(4+\gamma)-4}{\gamma} = \frac{4-4\lambda}{\gamma^2} < 0,$$
we can conclude that the function is maximal if $\gamma = \frac{3}{8}$. Therefore,

$$\varphi \leq \frac{1}{3}(35\lambda - 32).$$

As $\lim_{\lambda\to 1}\frac{1}{3}(35\lambda-32) = 1$, this would imply that we may approximate the vertex cover problem in cubic graphs arbitrary well which implies P=NP. □

Note that by scaling all costs and delays we may always assume that $\alpha = \beta = 1$. If there is a further bound on the range of possible costs and delays better approximation algorithms may be possible.

5.8 The Power Recovery Problem

In practice, we observe that in approximative solutions for the time-cost tradeoff problem a substantial number of vertices that are accelerated by the algorithm can be discarded from the final solution without destroying feasibility. Of course, one could do this greedily or use the reverse order in which the vertices were accelerated, but this does not approximate the best possible choice of edges to be deleted from the solution.

Indeed, there is a simple $\frac{d}{2}$ approximation algorithm for this problem as we will now show.

Theorem 5.29. *Let $\Big(G=(V,E), (c_v)_{v\in V}, (d_v^1, d_v^2)_{v\in V}\Big)$ be an instance of the discrete time-cost tradeoff problem and $Y \subseteq V$ a feasible solution. Then there* *is a polynomial time algorithm that finds a set $X \subseteq Y$ such that $Y \backslash X$ is still feasible and $\frac{d}{2} c(X) \geq c(X^*)$, where X^* denotes the optimum solution.*

Proof As before we partition the set of vertices V into sets $V_1, \dots, V_d$ by setting $V_i := \{v \in V : \text{depth}(w) = i - 1\}$ where $\text{depth}(v)$ denotes the (unweighted) maximum length of any path ending in v. If we can show that for any pair V_i, V_j we can find the set $Z \subseteq (V_i \cup V_j)$ such that $\mathcal{U}_{Y \backslash Z} = \varnothing$ and $c(Z)$ is maximal, we are done by simply returning the best of those sets.

To show this note that any path $P \in \mathcal{P}$ can contain at most one vertex from V_i and one from V_j. Let us call these two vertices $v_i \in V_i$ and $v_j \in V_j$ (if they exist). Now it may happen that we may not add v_i and v_j simultaneously to Z. In this case we call v_i and v_j *conflicting*. The problem is now equivalent to finding a maximum set (w.r.t. the weight function c(e)) of pairwise non-conflicting vertices $Z \subseteq V_i \cup V_j$.

This is equivalent to finding a maximum weight independent set in the undirected bipartite graph with vertex set $V_i \cup V_j$ and edge set $\{\{v_i, v_j\} : v_i \text{ and } v_j \text{ are conflicting}\}$. This clearly can be done in polynomial time by solving a flow problem. □

The inapproximability of this problem is still open. At first glance the problem looks as difficult as the time-cost tradeoff problem, but our inapproximability proof cannot easily be adapted. If we assume that we have a 2 approximation for the recovery problem, it is still not easy to separate completeness and soundness. After our transformation, which reduces the problem to constant depth, approximately $\frac{1}{2}|V|$ vertices can be always set to the slow alternative in both cases (soundness and completeness). Analyzing the inapproximability is an interesting problem for future research.

Chapter 6

Gate Sizing

In the previous chapters, we considered the problem of threshold voltage optimization. For gates used on a chip not only threshold voltages can be adjusted, but also different implementations with varying transistor areas are available. Compared to V_t optimization, one major difficulty occurs when transistor areas are modified. Due to capacitance changes, also gates in the neighborhood are affected by resizing operations. Consider a gate $g \in \mathcal{G}$ and its predecessors $V_{pred}(g)$ in the timing graph. This situation is illustrated in Figure 6.1. When we increase the size of g, also the downstream capacitance of predecessor gates will increase. Using Elmore delay, we observe that this incurs an additional delay at these gates. This chapter is based on [Dab+18a], which is joint work with Nicolai Hähnle, Stephan Held, and Ulrike Schorr.

Our main contributions in this chapter are as follows. First, we present a new runtime analysis for the resource sharing formulation for gate sizing by Schorr [Sch15]. Thereby we resolve small inaccuracies in the proof of Schorr [Sch15], in particular concerning the outer binary search for the optimum power budget. By separating all needed assumptions from the proof, we allow our analysis to be easily applied to different delay models and objective functions by checking a list of prerequisites. Then, we extend the analysis of Schorr [Sch15] between the projected subgradient method and the resource sharing algorithm for gate sizing. We point out that an additional power constraint significantly improves the subgradient method. In earlier work by Langkau [Lan00] and Szegedy [Sze05] a separate power constraint was omitted in the problem formulation.

Finally, we present a practical implementation of the resource sharing algorithm for gate sizing with heuristic modifications. Previously, Flach et al. computed the best solutions on many instances of the ISPD 2013 benchmark set [Fla+14]. This algorithm was adapted for the use on industrial instances by Reimann et al. [RSR16a]. We compare our new implementation with their state-of-the-art algorithm on the same instances used in their publication. On all designs our algorithm obtains similar or better power savings while drastically reducing the runtime. The overall runtime is now fast enough to allow the algorithm to be used in practice. Due to the large power savings, it was quickly enabled in the default design flow by our industrial partner IBM.

6.1 Previous Work

Due to the strong interdependency of multiple gates, the gate sizing problem seems very difficult. However, while the number of possible V_t levels is usually a small constant (2-3), up to 10 or more sizes can be available for every gate. This justifies the relaxation to consider fractional gate sizes $x \in [l, u]$ for gates $g \in \mathcal{G}$. We will additionally assume that the delay functions are convexifiable. It is well known that for RC delay models this is the case after a variable transformation [FD85].

For solving the gate sizing problem, a large variety of approaches was proposed. Some examples are linear programming [CK05], network flows [RD13], delay or slew budgeting [Ngu+03; Hel09], sensitivity-based heuristics [Hu+12; Kah+13], interior point methods [Sap+93; Boy+05], or Lagrangian relaxation [CCW99; TS02; WDZ07; OBH12; Fla+14]. A survey can be found in [HH16].

Under our assumptions the gate sizing problem can be solved close to optimality, e.g. by the projected subgradient method applied to the Lagrangian dual function [CCW99], or with interior point methods [Sap+93; Boy+05]. However, for large instance sizes interior point methods become impracticable as each iteration has a super-quadratic running time [TS02; BJ08]. In contrast, each iteration of the projected subgradient method solves a Lagrangian subproblem in near-linear time [CW01], but without good bounds on the number of subgradient steps.

Langkau presented a gate sizing algorithm based on the projected subgradient method [Lan00]. This approach was later extended by Szegedy [Sze05]. In both cases the idea is to solve the Lagrangian subproblem by a local refinement step. It was already observed by Chu and Wong [CW01] that such a local greedy solves the Lagrangian subproblem for RC delay models. The convergence rate bounds were further improved by both Langkau and Szegedy [Lan00; Sze05]. However, in their approach no separate power constraint was added to the problem formulation. In this chapter we will give evidence that adding a separate power budget and the corresponding constraints to the optimization problem will lead to superior results for both feasible and infeasible designs.

Schorr [Sch15] formulated the gate sizing problem in the resource sharing framework and compared it to the subgradient method. Overall, she observed improved convergence for the resource sharing based approach and divergence for the subgradient method. We rule out some inaccuracies in the runtime

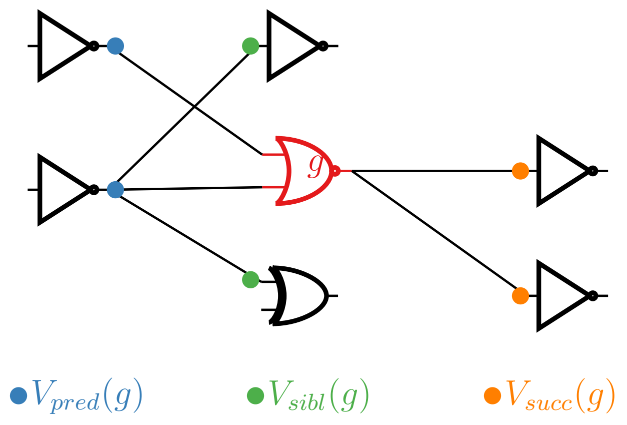


Figure 6.1: A gate g and the timing subgraph in its region. Sizing up gate g will increase the downstream capacitance of its predecessor gates in $V_{pred}(g)$ and increase their corresponding delays. If slews are also considered, even the sibling gates can be affected by this change. Figure reprinted from our publication [Dab+18a] (© 2018 IEEE).

analysis and give a proof that handles the general case in which the optimum power consumption is not known in advance.

6.2 Gate Sizing as a Resource Sharing Problem

We use the resource sharing formulation for gate sizing by Schorr [Sch15]. We denote the set of gates with $\mathcal{G}$ and the set of feasible cell vectors with $X = (X_g)_{g\in\mathcal{G}}$, where X_g is the set of alternative cell types for gate $g \in \mathcal{G}$. For simplicity, we assume that $\min X_g = l$ and $\max X_g = u$ for all gates $g \in \mathcal{G}$. A solution $x \in X$ specifies a cell type $x_g \in X_g$ for every gate $g \in \mathcal{G}$.

Furthermore, we assume that timing constraints are modeled by a timing graph D and delay functions $delay_e : X \to \mathbb{R}_{\geq 0}$ for all edges $e \in E$ that specify the delay $delay_e(x)$ of e given a solution x. More details can be found in Chapter 2.1.

We assume that for each gate $g \in \mathcal{G}$ there is a function $power_g : X_g \to \mathbb{R}_{\geq 0}$ specifying the power consumption for choosing a specific cell type for g. The total power consumption is simply the sum of the gate power values. The cell selection problem can then be formulated as follows:

$$\text{minimize} \quad power(x) := \sum_{g\in\mathcal{G}} power_g(x_g) \tag{6.1}$$

$$\begin{aligned} \text{subject to} \quad a_v + \mathit{delay}_e(x) &\leq a_w && \forall\ e = (v,w) \in E(D) \\ a_v &\geq 0 && \forall\ v \in P_{\text{inp}} \\ a_v &\leq T && \forall\ v \in P_{\text{out}} \\ x &\in X, \end{aligned}$$

where $\mathit{power}_g(x_g)$ denotes the power consumption of cell type x_g, a_v the arrival time at $v \in V$, and T the desired clock cycle time. We make the simplifying assumption that all signals start at time 0 and all required arrival times equal a unique clock cycle time T.

An equivalent formulation forces each path delay to be bounded by T. To this end, let $\mathcal{P}$ denote the set of (inclusion-wise) maximal paths in D, i.e. the set of paths between a signal start and end point. Then (6.1) is equivalent to

$$\text{minimize} \qquad \mathit{power}(x) := \sum_{g \in \mathcal{G}} \mathit{power}_g(x_g) \tag{6.2}$$

$$\begin{aligned} \text{subject to} \quad \sum_{e \in E(P)} \mathit{delay}_e(x) &\leq T && \forall\ P \in \mathcal{P} \\ x &\in X, \end{aligned}$$

where $E(P) \subseteq E$ denotes the set of timing graph edges in path P. At a first glance, the path formulation (6.2) appears inferior due to the possibly exponential number of paths and, thus, constraints. However, as we saw in Theorem 3.4, even an exponential amount of path resources can be handled in polynomial time in the resource sharing framework.

We use the resource sharing formulation for gate sizing by Schorr [Sch15]. Namely, we use the timing path resources explained in Section 3.3.3. In addition, we add a single power resource. In total, the set of resources is thus given by $\mathcal{R} = \mathcal{P} \cup \{\mathit{power}\}$. To define the power resource usage, we will assume that some power budget $B \in \mathbb{R}$ is known. It can be computed by an outer binary search on the best possible value. This leads to the following feasibility problem.

$$\begin{aligned} \mathit{power}(x) &\leq B, \\ \sum_{e \in E(P)} \mathit{delay}_e(x) &\leq T \quad \forall\ P \in \mathcal{P}, \\ x &\in X. \end{aligned} \tag{6.3}$$

Due to the interdependency of the gates, we only add a single customer $\mathcal{C} = \{C\}$. Usage functions of power and path resources are defined by the respective power consumption and path delay of a given solution. The downside of this fomulation is that the oracle problem will require to find a solution for all gates simultaneously.

An oracle function for the gate customer computes for given resource weights

$\omega \in \mathbb{R}^{\mathcal{R}}$ feasible sizes x for all gates such that the weighted resource usage

$$\omega_{power}\frac{power(x)}{B} + \sum_{P\in\mathcal{P}} \omega_P \frac{\sum_{e\in E(P)} delay_e(x)}{T} \tag{6.4}$$

is minimized up to a factor of $\sigma > 1$. With implicit edge weights

$$\omega_e := \sum_{P\in\mathcal{P}:\ e\in E(P)} \omega_P, \tag{6.5}$$

the sum (6.4) can be rewritten as

$$\omega_{power}\frac{power(x)}{B} + \sum_{e\in E} \omega_e \frac{delay_e(x)}{T}. \tag{6.6}$$

Note that the weights $(\omega_e)_{e\in E}$ fulfill the flow conservation rule $\sum_{e\in\delta^-(v)} \omega_e = \sum_{e\in\delta^+(v)} \omega_e$ because they are derived from the path weights. As an interesting side effect, the Karush-Kuhn-Tucker (KKT) conditions are fulfilled by $(\omega_e)_{e\in E}$ without requiring an extra projection step as in the Lagrangian relaxation based algorithm described in [CCW99]. However, the resource sharing algorithm for gate sizing does not depend on the KKT conditions.

To satisfy all prerequisites for the resource sharing algorithm, we define a set of technical constraints.

A1 $X = [l, u]^{\mathcal{G}}$, where $l, u \in \mathbb{R}_{>0}$ and $1 \le l \le u$, where $u_{\max} := \max\{u_g : g \in \mathcal{G}\}$ is a small constant compared to the netlist size.

A2 For all $e \in E$, the function $delay_e(x)$ is convex and for all $g \in \mathcal{G}$ $power_g(x)$ is convex and nondecreasing.

A3 $power(u)/power(l) \le \hat{U}$,

A4 the gradient $\nabla power(x) = \nabla \sum_{g\in\mathcal{G}} power_g(x_g)$ is Lipschitz continuous with bound K_P,

A5 the gradient $\nabla \sum_{e\in E} delay_e(x)$ is Lipschitz continuous with bound K_D, and

A6 $\min_{x\in X, e\in E} delay_e(x) \ge d_{\min} > 0$ ($d_{min} \in \mathbb{R}$),

where $\hat{U}, K_P$ and K_D are technology-specific constants independent of the netlist. Note that A3 and A4 hold for prevalent linear power functions with $K_P = \max_{g\in\mathcal{G}, x\in X_g} power_g(x)$ and $\hat{U} \le u_{\max}$. Schorr [Sch15] shows that A5 holds for the RC-delay model if the fan-in and fan-out of each gate is bounded by a constant. As we only use the gate sizing algorithm for buffered netlists, this assumption is met for our application. Finally, A6 usually holds because a gate of interest will at least drive the input pin capacitance of another gate.

By leveraging Theorem 3.2 one can show the following theorem.

Theorem 6.1 ([Dab+18a]). *Assuming A1–A6, $\epsilon > 0$, and that the gate sizing problem has a feasible solution, we can compute a gate sizing solution that minimizes the optimum power up to a factor of $(1+\epsilon)$ and violates any delay constraint by at most a factor of $(1+\epsilon)$ in time $\mathcal{O}\Big(u_{\max}^2 \hat{K} \cdot T_{grad} \cdot \Lambda \cdot \hat{U} \cdot \epsilon^{-3} \log|\mathcal{P}| \log \frac{\log \hat{U}}{\epsilon}\Big)$.*

We will now show how to derive it. By setting $\lambda_{power} = \omega_{power}/B$ and $\lambda_e = \omega_e/T$, we can rewrite (6.4) as a Lagrange function similar to [CCW99; TS08] with weighted power:

$$L(\lambda, x) := \lambda_{power} power(x) + \sum_{e\in E} \lambda_e delay_e(x). \tag{6.7}$$

For the RC-delay model, the Lagrange function can be minimized in polynomial time with a greedy algorithm as proposed in Chu and Wong [CW01]. Under certain assumptions, a solution $x \in X$ with $|(x_i^* - x_i)/x_i^*| \leq \epsilon$ for all $i = 1, \dots, n$ can be computed in $\mathcal{O}(n \log(1/\epsilon))$ time for $\epsilon > 0$. However, in our context we are interested in an approximation guarantee on the *value* of (6.7), whose scale depends on the exponentially growing weights.

Theorem 6.2. *Assume that A1–A6 hold and let T_{grad} be the time that is needed to compute the gradient $\nabla L(\lambda, x)$, which we also assume to dominate the time it takes for changing x in gradient direction. Let further $\hat{K} := \max\Big(\frac{K_P}{power(l)}, \frac{K_D}{d_{\min}}\Big)$. Then there exists an oracle for the gate customer that computes for $\omega \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ and $\sigma > 1$ a solution $x \in X$ in time $\mathcal{O}\Big(T_{grad}\hat{K}\frac{u_{\max}^2}{\sigma-1}\Big)$ such that the weighted resource usage (6.6) of the gate customer is minimized up to a factor of σ.*

Proof To simplify notation, we consider the minimization of the transformed weighted resource usage $L(\lambda, x)$ in (6.7) instead of (6.6). For fixed resource weights λ its gradient $\nabla L(\lambda, x)$ is Lipschitz continuous in x by A4 and A5:

$$\begin{aligned} lip(\lambda) := \max_{x,y\in X} \frac{\Big\|\nabla L(\lambda, x) - \nabla L(\lambda, y)\Big\|_\infty}{\|x-y\|_\infty} \\ \leq \lambda_{power} K_P + \max_{e\in E} \lambda_e K_D. \end{aligned}$$

We apply the well-known conditional gradient method of Frank and Wolfe [FW56] to $L(\lambda, x)$. Starting with an initial solution $x^{(0)} \in X$, in each iteration k of this descent method a minimizer $s := \arg\min_{y\in X}\langle y, \nabla L(\lambda, x^{(k)})\rangle$ of the linear approximation at $x^{(k)}$ is computed and a step from $x^{(k)}$ towards s is performed: $x^{(k+1)} := x^{(k)} + \theta^{(k)}(s - x^{(k)})$ with step size $\theta^{(k)} = \frac{2}{k+2}$. The linear minimization subproblem can be solved in linear time: For each entry s_g ($g \in \mathcal{G}$)

we set

$$s_g = \begin{cases} l_g & \text{if } \nabla L(\lambda, x^{(k)})_g > 0 \\ u_g & \text{if } \nabla L(\lambda, x^{(k)})_g < 0 \\ x_g^{(k)} & \text{otherwise.} \end{cases}$$

Let $diam_X := \max_{x,y \in X} \|x - y\|_\infty \leq u_{\max}$ be the diameter of X that is bounded by u_{max} by A1, and let $opt(\lambda) = \min_{x \in X} L(\lambda, x)$ be the minimum resource consumption for weights λ. The convergence analysis of the conditional gradient (see for example Jaggi [Jag13]) yields that after $k \geq 1$ iterations

$$L(\lambda, x^{(k)}) - opt(\lambda) \leq 2 \frac{lip(\lambda)}{k+2} diam_X^2.$$

It follows that a solution x with $L(\lambda, x) - opt(\lambda) \leq \sigma - 1$ can be computed in $\mathcal{O}\left(\frac{lip(\lambda) u_{\max}^2}{\sigma - 1}\right)$ iterations.

Now let lb_{opt} be a lower bound on $opt(\lambda)$. We run the conditional gradient method up to accuracy $(\sigma - 1) \cdot lb_{opt}$ such that $L(\lambda, x) \leq opt(\lambda) + (\sigma - 1) \cdot lb_{opt} \leq \sigma \cdot opt(\lambda)$ as desired.

It remains to find a good lower bound lb_{opt} to prove the desired total running time. In particular, we are interested in a running time that is independent of the weights λ (and hence independent of ω). We can bound

$$lb_{opt} \geq \max \left\{ \lambda_{power} \cdot power(l), \max_{e \in E} \lambda_e \cdot d_{\min} \right\}.$$

This implies a bound for $lip(\lambda)/lb_{opt}$:

$$\begin{aligned} lip(\lambda)/lb_{opt} &= \frac{\lambda_{power} K_P + \max_{e \in E} \lambda_e K_D}{lb_{opt}} \\ &\leq \frac{\lambda_{power} K_P}{\lambda_{power} power(l)} + \frac{\max_{e \in E} \lambda_e K_D}{\max_{e \in E} \lambda_e d_{\min}} \\ &= \frac{K_P}{power(l)} + \frac{K_D}{d_{\min}} \leq 2\hat{K}. \end{aligned}$$

Thus, It takes $\mathcal{O}\left(\frac{lip(\lambda) u_{\max}^2}{(\sigma-1) \cdot lb_{opt}}\right) = \mathcal{O}\left(\hat{K} \frac{u_{\max}^2}{\sigma - 1}\right)$ iterations and $\mathcal{O}\left(T_{grad} \hat{K} \frac{u_{\max}^2}{\sigma - 1}\right)$ time to achieve an σ-approximate solution. □

The running time depends on the problem width Λ, which is defined as the maximum ratio by which a single customer can overuse a resource in any solution compared to the optimum:

$$\Lambda := \max\left\{1, \sup\left\{\frac{(\mathrm{usg}_c(x_c))_r}{\lambda^*} : r \in \mathcal{R}, c \in \mathcal{C}, x_c \in X_c\right\}\right\}. \tag{6.8}$$

We will use Theorem 3.2 to analyze the runtime of the resource sharing algorithm for gate sizing. As it was defined several chapters ago, we repeat its statement.

Theorem 3.2 (Müller, Radke, and Vygen [MRV11], Lemma 7)**.** *Let* $0 < \delta, \delta' < 1$. *Given an instance of the min-max resource sharing problem with* $\lambda^* \leq 1$, *we can compute a* $\left(\sigma(1+\delta) + \frac{\delta'}{\lambda^*}\right)$*-approximate solution using* $\mathcal{O}(\Lambda(\delta\delta')^{-1}\sigma \log|\mathcal{R}|)$ *calls to an* σ*-approximate oracle function.*

Algorithm 6.1 is the resource sharing algorithm from [MRV11] with one minor modification. The original algorithm in [MRV11] has a mechanism to repeat oracle calls for a single customer if the usage of a resource exceeds one. This mechanism is important if many customers are present. In our case, we have only a single customer and can simplify the algorithm by setting the iteration count I to the worst case number of iterations from [MRV11] right away, i.e. choosing

$$I = \mathcal{O}(\Lambda(\delta\delta')^{-1}\sigma \log|\mathcal{R}|).$$

and $\gamma = \frac{\delta}{3\sigma}$ as in [MRV11].

As $|\mathcal{R}| = |\mathcal{P}| + 1$, we obtain the following guarantee for the continuous relaxation of the gate sizing problem:

Lemma 6.3. *Assuming A1–A6, given a power budget* $B \in \mathbb{R}_{\geq 0}$, *and* $0 < \epsilon < 1$, *we can decide whether* $\lambda^* \leq (1+\epsilon)$ *or* $\lambda^* > 1$ *using Algorithm 6.1 in time* $\mathcal{O}\left(u_{\max}^2 \hat{K} \cdot T_{grad} \cdot \Lambda \cdot \hat{U} \cdot \epsilon^{-3} \log|\mathcal{P}|\right)$.

Proof By A3, we have $\lambda^* \geq power(l)/power(u) \geq \hat{U}^{-1}$. We apply Theorem 3.2 with $\delta = \epsilon/4, \sigma = 1 + \delta$ and $\delta' = \delta/\hat{U}$. If $\lambda^* \leq 1$, we obtain a solution with maximum resource usage at most $\left((1+\epsilon/4)^2 + \epsilon/4\right) \leq (1+\epsilon)$. Otherwise, if the maximum resource usage of the solution is greater than $(1+\epsilon)$, we can conclude $\lambda^* > 1$. By our choice of σ, δ, and δ', Theorem 3.2, and Theorem 6.2, the running time is $\mathcal{O}\left(\frac{u_{\max}^2 \hat{K} \cdot T_{grad}}{\epsilon} \cdot \Lambda \cdot \frac{\hat{U}}{\epsilon^2} \log|\mathcal{P}|\right)$. $\square$

Finally, we apply binary search on B to minimize the total power.

Theorem 6.4. *Assuming A1–A6,* $\epsilon > 0$, *and that the gate sizing problem (6.2) has a feasible solution, we can compute a gate sizing solution that minimizes the optimum power up to a factor of* $(1+\epsilon)$ *and violates any delay constraint by at most a factor of* $(1+\epsilon)$ *in time* $\mathcal{O}\left(u_{\max}^2 \hat{K} \cdot T_{grad} \cdot \Lambda \cdot \hat{U} \cdot \epsilon^{-3} \log|\mathcal{P}| \log\frac{\log \hat{U}}{\epsilon}\right)$.

Proof Let $\epsilon' = \epsilon/4$. We perform binary search among the budgets $B_i := \mathit{power}(l) \cdot (1+\epsilon')^i$ for $i = 0, \ldots, i_{\max}$, where $i_{\max} = \min\{i \in \mathbb{N}_0 : \mathit{power}(l) \cdot (1+\epsilon')^i \geq \mathit{power}(u)\} = \mathcal{O}(\frac{\log \hat{U}}{\epsilon'})$. For every tested budget B_i, we call Algorithm 6.1 with accuracy ϵ' to decide whether B_i is feasible up to a factor $(1+\epsilon')$, i.e. the maximum resource usage and λ^* do not exceed $(1+\epsilon')$ or infeasible, i.e. the maximum resource usage exceeds $(1+\epsilon')$ and, thus, $\lambda^* > 1$.

As the instance is feasible the binary search will identify a locally smallest index $i_0 \in \mathbb{N}_0$ such that we find a solution with maximum usage $1+\epsilon'$ for the instance with budget B_{i_0}.

Let $B^\star$ be the optimum budget. For all $B_i \geq B^\star$ by Lemma 6.3, Algorithm 6.1 returns a solution in which the maximum timing violation is $1+\epsilon' \leq 1+\epsilon$. Therefore, $B_{i_0} \leq (1+\epsilon') \cdot B^\star$ and the total power consumption is bounded by $(1+\epsilon/4)B_{i_0} \leq (1+\epsilon/4)^2 B^\star \leq (1+\epsilon)B^\star$. The binary search examines $\mathcal{O}(\log \frac{\log \hat{U}}{\epsilon})$ budgets, and by Lemma 6.3 the total running time is $\mathcal{O}\Big(u_{\max}^2 \hat{K} \cdot T_{grad} \cdot \Lambda \cdot \hat{U} \cdot \epsilon^{-3} \log |\mathcal{P}| \log \frac{\log \hat{U}}{\epsilon}\Big)$. □

Similarly, we could perform a search on T to minimize the feasible cycle time, or two searches for minimizing the cycle time and then also the power.

If $\Lambda\hat{U} \geq |E|$ holds, we can use a different running time analysis by [Häh15] and obtain the following alternative result for Lemma 6.3.

Lemma 6.5. *Assuming A1–A6 and given a power budget* $\mathit{budget}_{power} \in \mathbb{R} \geq 0$, $0 < \epsilon < 1$, *we can decide whether* $\lambda^* \leq (1+\epsilon)$ *or* $\lambda^* > 1$ *using Algorithm 6.1 in time* $\mathcal{O}\Big(u_{\max}^2 \hat{K} \cdot T_{grad} \cdot |E| \epsilon^{-3} \log |\mathcal{P}|\Big)$.

Combined with binary search, this yields a total running time of

$$\mathcal{O}\left(u_{\max}^2 \hat{K} \cdot T_{grad} \cdot |E| \epsilon^{-3} \log |\mathcal{P}| \log \frac{\log \hat{U}}{\epsilon}\right).$$

The cardinality $|\mathcal{P}|$ appears only logarithmically, thus as a rough estimate we can derive a linear bound

$$\log |\mathcal{P}| \leq \log 2^{|V|} = \mathcal{O}(|V|).$$

Moreover, under the assumptions of Lemma 6.3, the running time is provably polynomial. Filtering out the technology-dependent constants $u_{\max}, \hat{K}, \hat{U}$ and Λ the running time in Theorem 6.4 is essentially

$$\mathcal{O}\left(\frac{T_{grad}}{\epsilon^3} |V| \log \frac{1}{\epsilon}\right).$$

For the special case of gate sizing, the resource sharing algorithm can be simplified to Algorithm 6.1. As there is only a single customer, we do not have to solve certain customers multiple time per phase but only maintain a global phase counter.

Algorithm 6.1: Resource sharing algorithm for gate sizing

```
Input: An instance of the gate sizing problem, a power budget $B$,
       SizingOracle($\omega$) for the gate customer, $\gamma > 0$, $I \in \mathbb{N}$.
Output: Convex combination of gate sizes $x \in \mathrm{conv}(X)$.
1  $x \leftarrow 0,\ \Xi \leftarrow 0$;
2  $y_e \leftarrow 0$ for all $e \in E$, $y_{power} \leftarrow 0$;
3  for $i = 1, \dots, I$ do
4      $\omega_{power} \leftarrow e^{\gamma \cdot y_{power}}$;
5      $\omega_E \leftarrow$EdgeWeights($y_E, \gamma$);                (Algorithm 3.2)
6      $x' \leftarrow$ SizingOracle($\omega$);
7      $\xi \leftarrow \min\left\{\frac{B}{power(x')}, \frac{T}{||delay(x')||_\infty}\right\}$;
8      if $\xi \geq 1$ then return $x'$;
9      $y_e \leftarrow y_e + \xi\frac{delay_e(x')}{T}$ for all $e \in E$;
10     $y_{power} \leftarrow y_{power} + \xi\frac{power(x')}{B}$;
11     $x \leftarrow x + \xi \cdot x'$;
12     $\Xi \leftarrow \Xi + \xi$;
13 return $\frac{1}{\Xi}x$;
```

Algorithm reprinted from our publication [Dab+18a] (© 2018 IEEE).

The weights ω_{power} and ω_e in Equation 6.6 can be seen as Lagrange multipliers. In fact, Schorr observed that heuristic modifications to the Lagrangian multipliers in the subgradient method often lead to similar update rules as used in the resource sharing algorithm [Sch15].

In practice, optimizing weighted pin slacks instead of timing edge delays often significantly improves the convergence speed. A detailed evaluation of this was given by Schorr [Sch15]. Therefore, we optimize the following heuristic objective when solving the discrete gate sizing problem in practice.

$$\omega_{power}\frac{power(x'_g)}{B} - \sum_{v \in Nb(g)} \omega_v \frac{\mathrm{slack}_{x'}(v)}{T}. \tag{6.9}$$

In the heuristic objective function we make use of pin weights ω_v for $v \in V(D)$. They can be obtained by

$$\omega_v := \omega_{\mathcal{P}_{[P_{\mathrm{inp}},v]}} \cdot \omega_{\mathcal{P}_{[v,P_{\mathrm{out}}]}} = \max\left\{\sum_{e\in\delta^+(v)} \omega_e, \sum_{e\in\delta^-(v)} \omega_e\right\}, \tag{6.10}$$

where $\omega_{\mathcal{P}_{[P_{\mathrm{inp}},v]}} \cdot \omega_{\mathcal{P}_{[v,P_{\mathrm{out}}]}}$ is defined in the proof of Lemma 3.4.

Algorithm 6.2 summarizes the heuristic local search oracle in the resource

Algorithm 6.2: Local search oracle with pin weights

```
  Input: A timing graph $D$. Cumulative usages $y_e$ for $e \in E(D)$. $\gamma > 0$.
  Output: A new cell selection $x'_g \in X$.
1 $x' \leftarrow x$ for $g \in \mathcal{G}$ do
2     Update slacks for all $v \in Nb(g)$ with global slew and delay
        propagation.
3     Choose $x'_g$ s.t. (6.9) is minimized with slew and delay propagation
        restricted to the region around $g$.
4 return x'
```

Algorithm reprinted from our publication [Dab+18a] (© 2018 IEEE).

sharing algorithm which we use in practice.

6.3 Comparison to the Projected Subgradient Method

We also implemented the projected subgradient method (core implementation by [Sch15]) to allow comparison with the resource sharing algorithm. It is built around the same oracle (Algorithm 6.2) which enables us to directly compare the different weight update schemes. In this particular comparison, the oracle locally minimizes the original objective from (6.6) instead of (6.9) for both the subgradient method and the resource sharing algorithm as required by theory.

To get a fair comparison between the two methods, we omitted heuristic modifications of the subgradient method except for those necessary to implement a discrete cell selection oracle.

For the subgradient method, it is difficult to determine initial multipliers and step lengths that work well for a broad range of instances, despite many improvements [TS02; WDZ07]. We simply initialize each multiplier $(\lambda_e)_{e \in E(G)}$ with a small percentage of the absolute negative slack of e. The Lagrange multipliers are updated by the local edge slack as in [CCW99] and projected to the nonnegative flow space with the heuristic from [TS02]. Schorr conducted experiments in which she projected the multipliers exactly by solving the arising quadratic minimum cost flow problems. However, she observed that the results and convergence behavior did not improve significantly [Sch15].

We implemented a variant of the subrgradient method suggested by Jiang et al. [JJC99], which solves the feasibility problem (6.3) and maintains a power multiplier λ_P. This has the advantage that on infeasible designs, where edge delay multipliers grow to infinity, the objective is not dominated by the delay multipliers.

We tested several step size rules and found best results when using the step size $1/\sqrt{i}$ in the i-th subgradient iteration, which guarantees convergence.

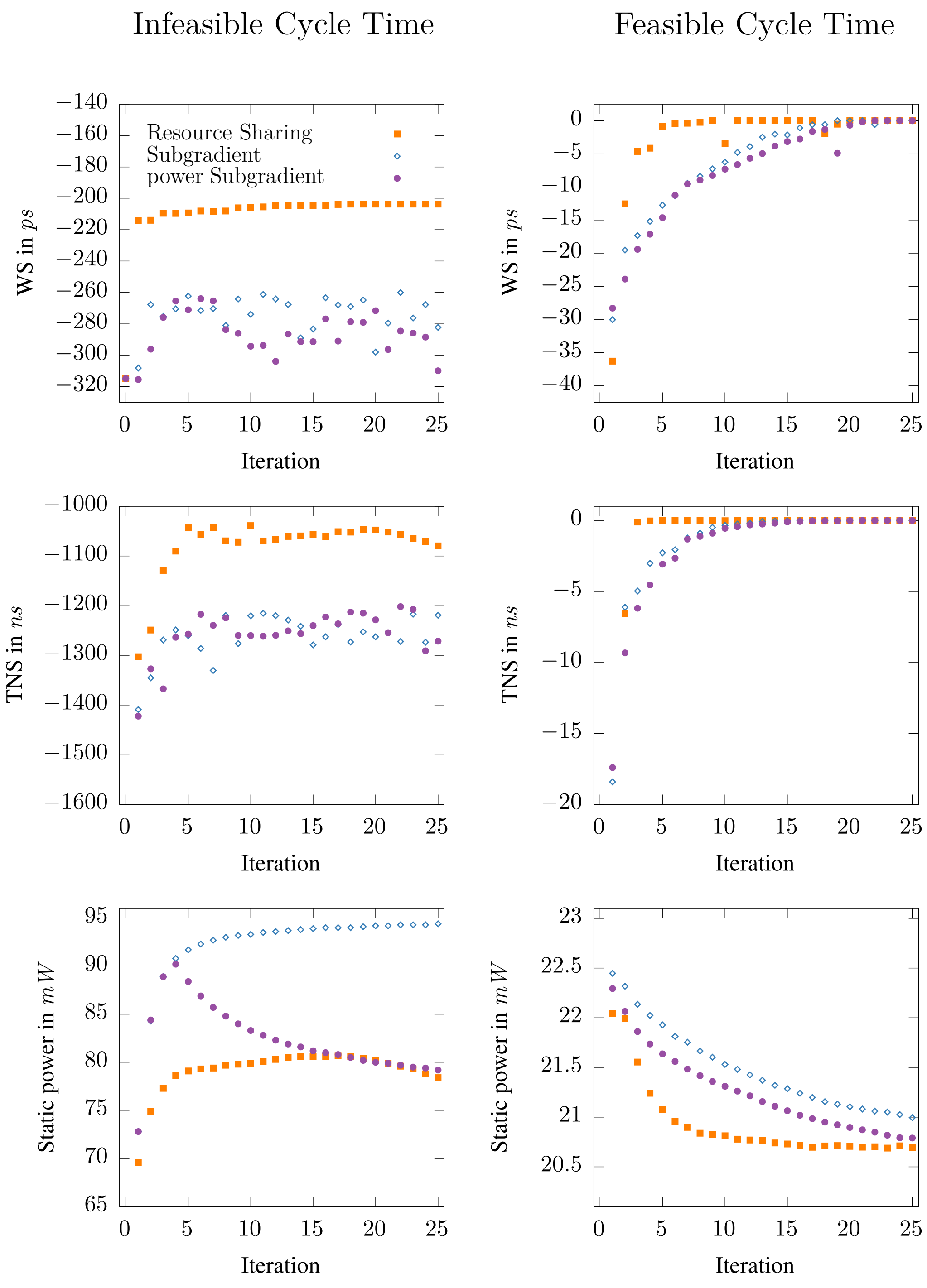


Figure 6.2: Convergence of resource sharing algorithm vs. subgradient method. Figure reprinted from our publication [Dab+18a] (© 2018 IEEE).

On the left side of Figure 6.2, we compare our new resource sharing algorithm (squares) with the regular projected subgradient method [CCW99] (diamonds) and with the power-constrained subgradient method [JJC99] (circles) on an industrial instance with infeasible timing constraints. On the right side, we compare the same methods on the same instance but with a relaxed feasible cycle time. The figures show the development of the worst slack WS, the total negative slack TNS and static power consumption after each of 25 iterations.

On the infeasible instance, both subgradient methods show bouncing WS and TNS. For the regular variant, the power consumption increases as multipliers go to infinity. The power-constrained subgradient method lets the power consumption decrease from iteration 5 on, as the power multiplier grows, too. The resource sharing algorithm shows a stable convergence in all metrics, yielding significantly better WS and TNS.
On the feasible instance, all methods converge to feasible solutions. Again, the resource sharing algorithm exhibits a much faster convergence and a better power consumption after 25 iterations.

Compared to the experiments by [Sch15] we see the importance of adding a separate power resource. Both for feasible and infeasible designs the subgradient method performs significantly better when this additional constraint is added.

6.3.1 Results on industrial instances

The experiments on industrial 22nm instances were conducted on a heterogeneous cluster with Intel Xeon CPUs with clock frequencies between 2.6 and 3.5 GHz. For each instance, all experiments were carried out on the same server. We ran our algorithm with six threads. We compared our approach (*RS*) with the Lagrangian relaxation (*LR*) algorithm by [RSR16a], which runs sequentially and which was integrated into the IBM design environment by the authors of [RSR16a].

We ran physical design with the same instances as [RSR16a]. As [RSR16a], we used the result of the current IBM design flow just before detailed routing as input to our cell selection experiments. Thus, the input to cell selection differs slightly to the input in [RSR16a], which, in turn, differs to the one in [RSR16a], all caused by minor changes of the flow. However, within our new experiments the input to our *RS* algorithm and to the *LR* algorithm coincide.

On these instances the primary purpose of cell selection is to reduce the power consumption while maintaining the timing metrics. Instead of optimizing the power budget B via binary search, we initialize B as 80% of the initial power consumption. Then, after each iteration of Algorithm 6.1, we increment (decrement) B by the factor 1.15 (1.15^{-1}) if the TTNS decreased (increased) by more than 5% compared to the previous iteration. On top of that we use a further heuristic modification. In lines 4,5 of Algorithm 6.1, we replace γ by γ/i. This pays respect to the fact that we aim to find a good integral solution

INSTANCE	$\lvert\mathcal{G}\rvert$	Λ^t	Λ^p	Flow	WS	TNS	TTNS	v_{slew}	v_{load}	P_{static}	P_{total}	ΔP_{total}	t_{wall}
					[*ps*]	[*ns*]	[*ns*]			[μW]	[μW]		[*h:m:s*]
ibm16uP_01	99k	16	10	Industrial	-69.5	-101.4	-602.6	11	0	81.7	95.1		
				LR [RSR16a]	-69.6	-94.7	-583.4	5	1	65.7	79.0	-16.9%	11:25:03
				RS	-69.4	-103.2	-582.1	3	0	65.7	79.0	-16.9%	57:16
ibm16uP_02	10k	11	14	Industrial	-156.9	-1.9	-10.0	0	5	1.2	2.5		
				LR [RSR16a]	-156.9	-1.9	-10.0	0	3	1.2	2.4	-2.1%	1:47:53
				RS	-156.7	-1.9	-10.2	0	0	1.2	2.4	-1.4%	24:06
ibm16uP_03	9k	8	7	Industrial	7.0	-0.0	-0.0	0	2	2.7	52.5		
				LR [RSR16a]	7.0	-0.0	-0.0	0	2	2.7	52.4	-0.1%	1:19:29
				RS	7.0	-0.0	-0.0	0	2	2.7	52.7	+0.5%	22:13
ibm16uP_04	7k	14	9	Industrial	-11.2	-0.7	-0.7	0	0	1.6	2.9		
				LR [RSR16a]	-11.2	-0.7	-0.7	0	0	1.6	2.9	+0.7%	58:32
				RS	-11.1	-0.7	-0.7	0	0	1.6	2.9	-0.5%	15:22
ibm16uP_05	16k	9	4	Industrial	-76.6	-36.6	-64.0	91	2	20.3	67.4		
				LR [RSR16a]	-76.5	-37.1	-64.5	40	2	18.0	64.8	-3.8%	53:13
				RS	-76.3	-36.6	-64.0	42	1	16.7	63.2	-6.3%	20:19
ibm16uP_06	77k	13	6	Industrial	-108.9	-15.9	-25.6	20	381	35.7	147.6		
				LR [RSR16a]	-108.9	-14.6	-24.5	14	381	33.5	145.3	-1.5%	3:13:37
				RS	-108.9	-13.7	-21.0	10	381	33.9	145.8	-1.3%	38:13
ibm16uP_07	72k	14	9	Industrial	-33.9	-38.6	-231.6	9	4	60.8	73.2		
				LR [RSR16a]	-33.9	-38.7	-235.0	2	2	53.2	65.6	-10.4%	7:55:57
				RS	-33.3	-38.2	-221.3	3	5	53.2	65.6	-10.3%	47:45
ibm16uP_08	18k	11	4	Industrial	-72.6	-35.1	-176.4	64	4	16.8	85.6		
				LR [RSR16a]	-72.6	-35.0	-176.3	40	4	16.7	85.4	-0.3%	2:20:00
				RS	-72.6	-35.4	-175.3	40	2	12.0	79.7	-6.9%	17:19
ibm16uP_09	18k	11	6	Industrial	-23.2	-8.8	-36.2	3	1	14.5	47.6		
				LR [RSR16a]	-22.8	-8.7	-37.0	1	0	12.3	45.2	-5.0%	2:06:49
				RS	-22.6	-9.3	-36.4	1	1	12.4	45.4	-4.7%	17:47
ibm16uP_10	126k	14	6	Industrial	-43.8	-76.0	-342.6	67	7	91.6	397.1		
				LR [RSR16a]	-41.0	-84.2	-401.8	48	7	74.7	371.5	-6.5%	9:05:31
				RS	-38.9	-78.8	-346.9	31	2	65.1	365.3	-8.0%	1:22:58
ibm16uP_11	25k	16	5	Industrial	-140.7	-167.2	-886.7	19	27	39.7	61.6		
				LR [RSR16a]	-140.4	-164.1	-881.8	20	28	36.7	58.7	-4.7%	2:27:04
				RS	-140.4	-164.2	-842.2	20	21	34.5	56.5	-8.4%	19:30
ibm16uP_12	18k	16	4	Industrial	-417.8	-342.0	-696.1	12	3	5.1	25.4		
				LR [RSR16a]	-417.8	-333.7	-680.6	12	3	4.8	25.0	-1.8%	2:46:08
				RS	-417.7	-333.6	-675.6	12	0	5.3	25.5	+0.2%	18:36
ibm16uP_13	20k	11	4	Industrial	-47.6	-20.8	-103.4	1	2	19.6	80.2		
				LR [RSR16a]	-47.4	-20.3	-103.0	0	2	18.2	78.6	-2.0%	1:21:09
				RS	-47.3	-20.1	-101.0	0	2	15.4	75.8	-5.5%	21:35
ibm16uP_14	13k	7	2	Industrial	-54.8	-5.1	-9.2	1	4	8.2	17.9		
				LR [RSR16a]	-54.8	-5.1	-9.2	1	4	8.2	17.9	-0.1%	39:34
				RS	-54.3	-5.1	-9.2	1	4	8.2	17.9	+0.0%	13:04

Table 6.1: Results on IBM 22 nm server instances. Table reprinted from our publication [Dab+18a] (© 2018 IEEE).

and further improves the experimental convergence speed.

In every iteration of the algorithm we invoke the local search oracle exactly once. In global routing one usually chooses γ depending on the amount of iterations. [MRV11] obtain good solutions with $\gamma = \frac{125}{\#\text{iterations}}$. As we only perform four iterations we use a large value of $\gamma = 80$.

Table 6.1 shows the results. The instance names and their sizes are given in the first two columns. For each instance, we computed upper bounds Λ^t and Λ^p for the maximum usage of a timing or power resource by any feasible solution, respectively. To this end, we compute a minimum power solution $\bar{l}$ satisfying all capacity constraints but ignoring delay constraints. For Λ^t, we then assert at each sink pin of a net the highest possible pin capacity to get upper bounds for the delays. Then Λ^t is the quotient of the maximum path delay and T. Furthermore, $\Lambda^p = power(\bar{u})/power(\bar{l})$, where $\bar{u}$ is a solution using lowest V_t levels and largest sizes everywhere. It follows that $\Lambda \leq \max\{\Lambda^t, \Lambda^p\}$.

For every instance three solutions are analyzed. The solution that is given by the industrial flow is named *Industrial*, the solution computed by the Lagrangian relaxation algorithm of [RSR16a] is named *LR* and the solution computed by our resource sharing algorithm is called *RS*. For each of those runs we measured the worst slack (WS), the total negative slack (TNS) and the true total negative slack (TTNS) which were introduced in Section 7.2. The electrical violations are shown in terms of the number of slew violations (v_{slew}) and load violations (v_{load}). For the power consumption we distinguish between the static power usage P_{static} and the total power usage $P_{\text{total}} = P_{\text{static}} + P_{\text{dynamic}}$. The testbed contains a high variety in terms of the leakage/dynamic power ratio. ΔP_{total} denotes the change in total power. Running times t_{wall} are given in the last column.

The timing metrics are measured just before detailed routing. All numbers refer to results after subsequent placement legalization. The power reductions and running times that we measured for *LR* are comparable to those in [RSR16a].

Throughout the testbed our algorithm obtains a comparable or a better total power reduction. On instance ibm16uP_08 we are able to reduce the power by 6.9% while the reference algorithm only reduces the total power by 0.3%. The running time is greatly reduced. On the largest instance ibm16uP_10 with 126k gates the resource sharing approach takes about 83 minutes. The reference algorithm does not only take more than 6 times the running time but also greatly worsens the TNS and TTNS while it obtains a worse power reduction.

Chapter 7

BonnRouteBuffer

We have now seen how to solve time-cost tradeoff and gate sizing problems. The global interconnect optimization problem is much more general. It consists of computing buffered routes for all nets. While time-cost tradeoff and gate sizing mostly affect power consumption and timing, the interconnect optimization step impacts the chip performance, routability, power consumption, netlength and various other resouces. Parts of this chapter have been previously published in [Dab+19]. The results are joint work with Stephan Held, Bento Natura and Daniel Rotter.

BonnRouteBuffer has been in development for many years. Rotter [Rot17] described the resource sharing formulation for buffering in his dissertation. He also gave a first implementation to solve it. A significant part of this code was written by him between 2014 and 2017. Next to Rotter, several more people have been involved. The used arrival time customers were initially implemented by Traub [Tra15] and later refined by Rotter and Scheifele. The multilabel algorithm, which finds better solutions for critical nets at the cost of higher runtime, was implemented by Natura under the supervision of Rotter [Nat17]. At the end of 2017, Rotter already obtained promising results but the implementation was not ready for industrial use. On some instances many electrical slew violations were left or the solution consumed a lot of additional power. The path search implementation was not entirely correct which could lead to unsatisfactory routes. Due to an overly simplistic placement model, the legalization step after buffer insertion could lead to large timing degradations. Finally, the runtime was inadequate for applications in an industrial design flow.

In this chapter, we present a refined implementation of BonnRoute-Buffer. We start by modifying the theoretic model of Rotter in two ways. First, we use timing path resources introduced by Hähnle [Häh15]. This greatly simplifies the reproducibility. Previously, arrival time customers were used which need several non-trivial adjustments to the outer resource sharing algorithm and the arrival time intervals to perform well [Tra15; Hel+17]. Our experiments indicate an equal solution quality with the new simplified model. Second, we remove an inaccuracy in Rotters model by explicitly adding further placement constraints. Instead of simple tile area capacities, we analyze the particular shape of the buffers and derive restrictions that allow us to prevent large displacements in the legalization step.

In addition to the model changes, we present a refined implementation with various enhancements and fixes. We also devise a design flow which includes an initial gate sizing step and a global wire based gate sizing. As a result our refined implementation now decisively outperforms a state-of-the-art design flow in almost all metrics including netlength, power, congestion and timing. We describe a set of speedup techniques that allow us to obtain the new results in up to 70% less runtime.

The new implementation is now the primary buffering algorithm used by our industrial partner IBM. Thereby, demonstrating how a mathematical sound formulation can outperform elaborate heuristics.

The remainder of this chapter is structured as follows. After discussing previous work in Section 7.1, we present our problem formulation in Section 7.2. The algorithm for the oracle problem is discussed in Section 7.3. Our practical improvements of the previous implementation are listed in Section 7.4. We compare our speedups to the implementation of Rotter in Section 7.5. Our new optimization flow is presented in Section 7.6. Finally, our practical results when using the new implementation in an industrial design flow are given in Section 7.7.

7.1 Previous Work

Modern chips exhibit metal stacks with many different pitches and various options to select wire widths and spacings. The corresponding signal speeds may differ by up to a factor of 10 or more. The variance in the optimum repeater spacing is even higher. Consequently, buffering cannot be done without specifying the wire structure.

Furthermore, all nets compete for the limited routing resources, particularly on high layers, and for the limited placement space, especially around placement blockages and buffer bays. For the overall chip performance, it is essential to balance these resources between all the interconnects.

Most approaches insert repeaters on a net-by-net basis with fixed sink delay bounds and limited interaction with global routing, e.g. buffering a given topology [Van90; LCT96; LZS12].

Rotter proposed a resource sharing formulation for global interconnect optimization, balancing global timing, routing, placement, and power constraints [Rot17]. The core of the algorithm is an oracle function that, given a net and Lagrangean resource prices for routing, timing, placement, and power, computes a buffered global route approximately minimizing the total cost. This subproblem is the *cost-based buffered Steiner tree problem*. If this step can be solved with an approximation guarantee, the resource sharing algorithm leads to a provably good global solution in a polynomial number of oracle calls. For a constant fanout, this can be achieved by combining ideas from [Alb+02; HL03]. However, for general fanout, it is unlikely that algorithms with a practical performance guarantee exist. As many nets only have a small number of terminals, exponential algorithms may be used to solve these instances optimally. A notable approach has been given by Rockel [Roc18] who explains how the Dijkstra-Steiner [HSV17] algorithm by Hougardy, Silvanus and Vygen, which was initially formulated for the shortest Steiner tree problem, can be extended to solve the cost-based buffered Steiner tree problem.

The algorithm of Rotter solves the oracle problem by first computing a geometric 2D topology. Then, it embeds the topology into the 3D global routing graph, also selecting wire width and spacing w.r.t. a linearized model of delay, power, and repeater space. Finally, it inserts repeaters, recycling the global routes as much as possible and possibly cloning them to save repeaters. For individual instances, where this approach fails, Rotter performs a more elaborate embedding inspired by ideas of [HL03] that embeds and buffers simultaneously. It was implemented by Natura [Nat17] under the supervision of Rotter [Rot17].

To reduce the problem complexity, most previous algorithms for buffer insertion separate interconnect optimization into 1) topology generation and 2) repeater insertion.

Many algorithms for topology generation fall into two categories: Prim-Dijkstra heuristics [Alp+95; Alp+18] and bicriteria approximation [HR13; CY19]. Sometimes topology enumeration is combined with a fast exact Steiner vertex embedding [HR18] or even buffering [HL03].

Maßberg [Maß15] considered the problem of finding positions for Steiner nodes in a fixed topology such that length is minimized while delay bounds are obeyed. He described a polynomial algorithm that applies dynamic programming to scaled instances. Rockel [HR18] observed that a linear program formulation of this problem is the dual of a min-cost flow problem. In practice this new approach is significantly faster.

Bartoschek, Held, Maßberg, Rautenbach and Vygen [Bar+10] proposed a new delay model for the repeater tree construction problem that includes bifucation penalties and thus leads to topologies with reduced depth.

Algorithms for repeater insertion and wire width optimization have their foundation in dynamic programming by [Van90] with improvements by [LCT96; HL03; Li+08; LZS12] to name a few. Recently, the pFOM was proposed as an

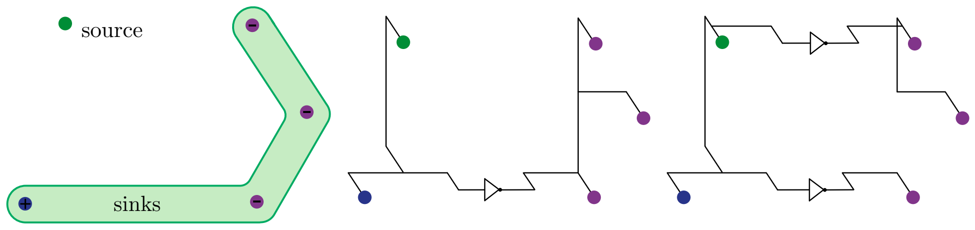


Figure 7.1: An important subproblem during buffering: Given polarities (blue=positive and magenta=negative) and electrical properties we need to compute a buffered 3D Steiner tree in a grid graph. There are two possible solutions on the right.

optimization metric [Hu+18].

Bartoschek, Held, Rautenbach and Vygen [Bar+09] presented an improved dynamic program called FastBuffering which allows topology changes to save inverters.

Timing-driven layer assignment has often been proposed as a standalone algorithm, e.g. by [Wei+13] or recently by [Liu+18] to transform a 2D into a 3D global routing.

The classic multicommodity flow formulation for global routing [SK87] has been extended to integrate net delay bounds and buffer insertion by [Alb+02], who presented a fully polynomial approximation scheme (FPTAS) for two-terminal nets. An extensive overview on previous work to solve the global routing problem was given in Chapter 3.1.

7.2 Problem Formulation

We will quickly review the most important notions from Chapter 3 to present the problem formulation. Let $\mathcal{N}$ be the set of nets of a placed circuit that are to be synthesized. As before, $\mathcal{I}$ is the chip image and $\square$ the chip area. Routing and placement congestion is modeled using a global routing graph G with $z_{\max}$ layers and a partition B of the placement area into bins.

Furthermore, there is a finite repeater library L of inverters and buffers and a set $\mathcal{T}$ of wire types. In conjunction with an axis-parallel line segment $I \subset \mathcal{I}$, a wire type $\tau \in \mathcal{T}$ determines a space consumption, as well as electrical resistance and capacitance of the (metal shape) pair (I, τ).

Each net $N \in \mathcal{N}$ consists of a source pin $s \in \square \times \{1, \dots, z_{\max}\}$ and a set of sink pins $T \subset \square \times \{1, \dots, z_{\max}\}$ with sink polarities $pol : T \to \{+, -\}$ and positions $\widehat{p} : T \cup \{s\} \to V(G)$.

Recall the following important definitions from Chapter 3.2. A *topology* for a net N is an arborescence A rooted in s, such that T is the set of leaves.

We require that the root has exactly one successor and all inner vertices have outdegree at most 2. A *Steiner tree* for a net N is a topology A together with a map into the chip image $p : V(A) \to \mathcal{I}$. p has to satisfy $p(s) = s, p(t) = t$ for the source s and any sink $t \in T$. We further require that the image $(p(v), p(w))$ for edges $(v, w) \in E(A)$ are either of length 0 or axis-parallel routing stick figures, i.e. they have to correspond to an edge in the global routing graph.

We call a tuple (A, p, φ, b) *buffered Steiner tree* for N if (A, p) is a Steiner tree for N. Each edge $e \in E(A)$ is assigned a wire type $\varphi(e) \in \mathcal{T}$. Finally, $b : V(A) \to L \cup \{\varnothing\}$ assigns Steiner vertices to a repeater, or to $\varnothing$ if no repeater should be placed. Here, wires are connecting to the centers of inserted repeaters disregarding the actual pin shapes. For any sink $t \in T$, the parity of inverting repeaters on the *s*-*t* path in A must match $pol(t)$. An instance with both polarities and two possible solutions is shown in Figure 7.1.

We assume that we are given functions that, given a global routing edge $e \in E(G)$ and a pair (I, τ) where $I \subset \mathcal{I}$ is an axis-parallel segment and $\tau \in \mathcal{T}$, returns the fraction of the global routing capacity of e that is used by (I, τ). Similarly, there is a function that for bin $\beta \in B$ and a pair (p, ι), where $p \in \square \times \{0\}$ is a placement position and $\iota \in L$, returns the fraction of the placement capacity of β that is used by ι, when placed at p.

Moreover, we are given functions cap and res that return the capacitance $\text{cap}(I, \tau)$ and resistance $\text{res}(I, \tau)$ of a certain wire (I, τ). Both are assumed to grow linearly in the length of the segment I.

Given switching frequencies $(\eta_N)_{N \in \mathcal{N}}$, power values $\text{power}(\iota, \eta)$ for every repeater $\iota \in L$ and switching frequency η, and a solution vector $(A_N, p_N, \varphi_N, b_N)_{N \in \mathcal{N}}$, we can compute the total power consumption with an appropriate scale factor $\textsc{c2p} \in \mathbb{R}$ as

$$\sum_{N \in \mathcal{N}} \left(\sum_{e \in E(A_N)} \eta_N \cdot \textsc{c2p} \cdot \text{cap}(p_N(e), \varphi_N(e)) + \sum_{v \in V(A)} \text{power}(b(v), \eta_N) \right). \tag{7.1}$$

Timing constraints are modeled by a timing graph D as described in Section 2, except that it reflects buffer insertion. The vertex set $V(D)$ consists of the sink pins in the unbuffered netlist plus all primary inputs. In order to use use sink pins consistently, we may consider a virtual buffer driving each primary input pin p. This is a standard procedure as also input pins have to be driven by some predecessor. Then, we would add the input to this buffer to $V(D)$, instead of p. There is an edge $(p, q) \in E(D)$ if p is an input of the gate that drives the net $N \in \mathcal{N}$ with $q \in N \setminus \{p\}$. The delay of an edge in the timing graph depends on the buffered global route for N and the source gate of N. The timing constraints are fulfilled if for every endpoint $v \in \{u \in V(D) : \text{outdegree}(u) = 0\}$, the maximum delay of a path in D ending in v is within a delay bound $\mathrm{rat}(v)$ that is part of the input.

For delays, we assume that gate delays are approximated by two non-decreasing functions depending on load capacitance and input slew, one for the delay, the other for the output slew. We model the interconnect timing by Elmore delay [Elm48].

The *timing-aware global routing and buffering problem* consists of computing a buffered Steiner tree for every net, such that no global routing edge and no placement bin is overused while meeting all timing constraints and minimizing the power consumption.

7.2.1 Customers and Resources for Interconnect Optimization

We now specify all the customers and resources that we use to model the global interconnect optimization problem. We roughly follow Rotter [Rot17].

For the global routing constraints, we follow [MRV11]. There is a customer for every $N \in \mathcal{N}$ and B_N is the set of buffered global routing solutions for N. There is a resource for every edge of the global routing graph $e \in E(G)$. The usage function $\mathrm{usg}_{N,e}$ specifies how much the wiring segments of a buffered global route $b(N)$ consume from e relative to its capacity.

Similarly, for placement constraints we add a resource for every placement bin $\beta \in B$. Again, the usage function $\mathrm{usg}_{N,\beta}$ specifies how much the repeaters from a buffered global route $b(N)$ consume from β relative to its capacity. The free area of β is given by its area minus the area of blockages, e.g. fixed macros. The resource usage of a repeater $\iota \in L$ corresponds to its area relative to the free area of the bin it is assigned to. Furthermore, the area of all non-repeating standard cells contributes to a fixed usage of the placement bins. Note that while we assign repeaters to a single bin, a non-repeating standard cell contributes fractionally to all bins it covers proportional to the overlap. This resource alone is not enough to avoid large displacements when legalizing. We further introduce an oracle $\zeta : L \times \mathcal{I} \to \{0, 1\}$. For a given buffer $l \in L$ and a position $p \in \mathcal{I}$ in the chip image, $\zeta(l, p) = 1$ if and only if l can be placed at position p. We will describe this oracle in detail in Section 7.4.3.

We consider standard cells as usage because this makes it easier to set a congestion target for the bins so that subsequent placement legalization succeeds without large movements when legalizing repeaters and non-repeaters simultaneously.

For the timing constraints, we follow the approach depicted in Section 3.3.3. We add a resource for each path P in the timing graph D to $\mathcal{R}$. Of course, we will only use these paths $\mathcal{P}$ implicitly, as $|\mathcal{P}|$ may be exponential in the size of the input.

In the following, we will assume that the delay of an edge in the timing graph only depends on our solution for the net that is modeled by that edge. Apart from slew effects which we will only model heuristically, this is mostly true in practice.

To model our objective of power minimization, we need to guess the optimum power consumption and add that as a constraint/resource $\{\text{power}\}$, which must not be exceeded by the power consumption in (7.1). Formally, we could conduct an optimum guess by binary search. However, we will simply make an initial guess on the optimum power and then slightly relax/tighten it after each iteration if the overall over usage increases/decreases. Netlength and via count are treated in a similar way in [MRV11].

Note that the resource sharing model is quite flexible and allows to add further resources and customers, e.g. one could have separate resources for static and dynamic power and the netlength or via count.

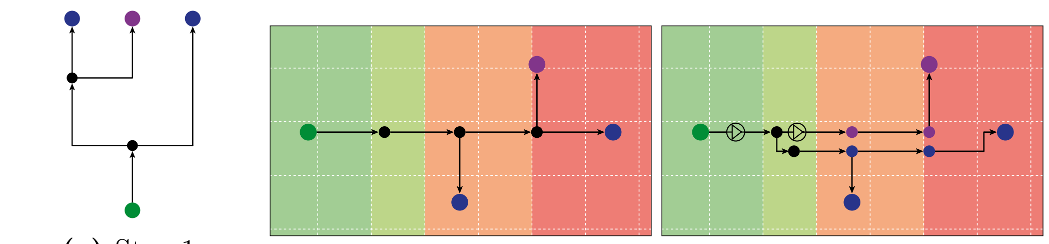
The resource sharing algorithm maintains prices price_r for all resources $r \in \mathcal{R} = E(G) \cup B \cup \mathcal{P} \cup \{\text{power}\}$.

For the timing paths $P \in \mathcal{P}$ the prices are only stored implicitly. As shown by Hähnle in Theorem 3.4 we can still compute edge prices for all edges in the timing graph $e \in E(D)$.

We still need to provide oracle functions for the net customers. For this, we have to solve the *cost-based buffered Steiner tree problem*. Here, for a net $N \in \mathcal{N}$, the problem is to compute a buffered global routing A approximately minimizing

$$\min \sum_{r \in \mathcal{R}} \text{price}_r \cdot \text{usg}_{N,r}(A). \tag{7.2}$$

We will present our oracle in Section 7.3. The resource sharing model usually



(a) Step 1: Computing a topology of small linear delay.

(b) Step 2: 2D projection of embedding the topology of Step 1 into the 3D global routing graph.

(c) Step 3: Buffering the route. Colors show placement congestion. The leftmost repeater is a buffer.

Figure 7.2: The three-stage routing & buffering process.

requires that all customer solutions can be computed independently and the resource prices guide the inter-dependencies. In our context, this assumption is violated because the slews at the sinks of a buffered Steiner tree influence the delays in the next stage. We use an approach by Rotter [Rot17] that mitigates this problem as follows. In the first iteration, we assume a default slew at all gate inputs. Whenever a new solution for a net has been computed, the slews at its sinks, i.e. the gate inputs of the next stage, are updated. To this end, it might seem favorable to process the nets in topological order of the timing graph. However, as this limits effective parallelization, we instead process the nets roughly by the size of their bounding-box.

7.3 An Oracle for the Cost-Based Buffered Steiner Tree Problem

Solving or even approximating the cost-based buffered Steiner tree problem is a notoriously hard problem in theory and in practice. Nevertheless, we need to tackle large instances with hundreds, thousands, or even millions of nets efficiently. To this end, we use the algorithm by Rotter [Rot17] which proceeds in three stages.

First, we compute a timing-aware Steiner topology/arborescence in the Manhattan plane (Section 7.3.1). Then, we embed this topology into the 3D global routing graph with respect to the objective (7.2), but using a linear model for placement, delay, and power usages (Section 7.3.2). Finally, we use dynamic programming for repeater insertion into the 3D Steiner tree, allowing to change the topology locally to save repeaters (Section 7.3.3). The three stages are visualized in Figure 7.2.

To estimate the effect of buffering in the first two stages, we customize the *linear delay routing oracle* of [Hel+17]. To this end, we linearize the delay, power, and placement usage functions. As proposed by [Bar+09] we compute, for each wire type $\tau \in \mathcal{T}$ and routing layer $l \in \{1, \dots, z_{\max}\}$ a fastest buffering of an infinite line on layer l with wire type τ, choosing the best uniform repeater type $\iota \in L$ and spacing repeaters uniformly. Similarly, we compute a most power efficient buffering allowing a 30% delay increase compared to the fastest buffering.

The fast buffered line determines a delay and placement usage per unit length, while we use the power-efficient buffered line to extract a static and dynamic power consumption per unit length. For a given $N \in \mathcal{N}$, we use the given switching frequency η_N to obtain a total power per unit length. The reason for using different buffered lines for delay and power is that in general different layer and wire trait pairs are best for delay minimization and power optimization. If the delay prices dominate the power price, the fastest layers and wire traits should be favored, and vice versa. It is straight forward to derive corresponding linear resource usage functions using the original resource capacities.

Unlike the first two stages, we consider Elmore delay, including slew propagation during buffer insertion, the discrete placement space usage and the primary power consumption function (7.1). However, we use the stationary slew of the fastest buffered line as an *estimated input slew* for newly inserted repeaters during buffer insertion.

As an alternative for Stage 2, we use a buffered embedding algorithm for simultaneous embedding and repeater insertion (Section 7.3.4) on difficult instances which was implemented by Natura [Nat17] under the supervision of Rotter [Rot17].

7.3.1 Stage 1: Topology Generation

For the topologies, we use a linear delay model. The previous implementation by Rotter [Rot17] proceeded as follows.

First, arrival times which were obtained by using arrival time customers were converted to heuristic delay bounds for all sinks. Then, the bicriteria algorithm of Held and Rotter was applied [HR13]. Given length bounds for each source-sink path and an $\epsilon > 0$, it computes a Steiner arborescence that exceeds these bounds by at most a factor $(1 + \epsilon)$ and whose length is within a factor $\left(2 + \left\lceil \log\left(\frac{2}{\epsilon}\right)\right\rceil\right) \cdot \frac{3}{2}$ of the length of a minimum Steiner tree. The length bounds are derived from the difference of the current arrival times at sinks and driver inputs, assuming the fastest layer and wire type combination for translating delay into distance. The value $\epsilon \in [0.1, 0.2]$ was chosen depending on the timing prices of the particular net.

Instead of relying on arrival times to guide the topology generation, it seems to be closer to theory if edge prices are directly optimized. To this end, we changed the topology generation to an implementation by Foos [Foo20] of the algorithm by Held and Khazraei [HK20]. As input, the new algorithm uses timing prices that were computed by the resource sharing algorithm. Currently, the edge weights are set heuristically to obtain similar netlengths as by the previous approach.

Furthermore, we employ a post-optimization recently proposed by [HR18]. The Steiner arborescence computed by the topology generation algorithm is partitioned into sub-arborescences with up to 9 leaves, which are restructured optimally via branch-&-bound.

7.3.2 Stage 2: Topology Embedding

In Stage 2, we embed the topology from Stage 1 into the global routing graph and also select a wire type for every edge. We use the method proposed by

[Hel+17], including their heuristic speed-up technique. However, we modify the cost function to reflect power and repeater space.

Formally, we embed the topology into the graph

$$G' := (V(G), \{\{x,y\}_\tau \mid \{x,y\} \in E(G),\, \tau \in \mathcal{T}\}),$$

that contains a copy of each edge for all wiretypes $\tau \in \mathcal{T}$.

As in [Hel+17], we traverse the topology in a bottom-up fashion (from sinks to source). For each Steiner vertex u, it simultaneously embeds the two outgoing edges (u,v) and (u,w) for which the endpoints are already embedded at $x_v, x_w \in V(G')$ and $X_u \subset V(G')$ is the set of vertices in G' covering u.

Now it computes shortest path labels for the x_v-X_u-path (similarly for the x_w-X_u-path) in G' w.r.t. the following cost for $e \in E(G)$ and $\tau \in \mathcal{T}$

$$c'(e,\tau) = c_{route}(e,\tau) + c_{place}(e,\tau) + c_{delay}(e,\tau) + c_{power}(e,\tau).$$

Here, $c_{route}(e,\tau) = \text{price}_e \cdot \text{usg}_e(\tau)$ is the routing congestion cost, $c_{delay}(e,\tau) = \text{price}_v \cdot \text{lin\_delay}(e,\tau)$, where price_v is the sum of delay prices to the sinks of the sub-arborescence rooted at v and $\text{lin\_usg}(e,\tau)$ is the linearized delay usage. Then, $c_{power}(e,\tau) = \text{price}_{power} \cdot \text{lin\_usg}_{power}(e,\tau)$ is the linearized power cost.

The most delicate modification concerns the approximation for the placement cost of repeaters that still need to be inserted. For an edge (e,τ), we estimate the average placement cost by its average linearized tile cost

$$c^{est}_{place}(e,\tau) = \frac{1}{2}\sum_{x\in e} \text{price}_{\beta_x} \cdot \text{lin\_usg}_{\beta_x}(e,\tau).$$

Finally, we set

$$c_{place}(e,\tau) = \alpha \cdot \min\Big(c^{est}_{place}(e,\tau), c_{route}(e,\tau)\Big), \tag{7.3}$$

where α is a constant parameter that weighs this rough approximation of the placement cost. Note that for some bins $\beta \in B$, we may have $\text{lin\_usg}_\beta \equiv \infty$. Therefore, we cut off $c_{place}(e,\tau)$ at $c_{route}(e,\tau)$ to avoid that the placement cost dominates the overall objective. For our experiments, we used $\alpha = 0.05$.

The position of u is given by the point in X_u where the two paths first meet. As in [Hel+17], we continue labeling a few further vertices, even after both path searches have reached X_u, to find a possibly better location of the Steiner vertex u estimating the upstream cost to its parent with a future cost estimate.

The embedded topology consists of global wires reaching from tile center to tile center. In particular, it does not directly connect to the given pins. This is too inaccurate for repeater insertion, unless a very small tile size is used. Thus, we post-process the routes before buffering. To achieve this, we first complete

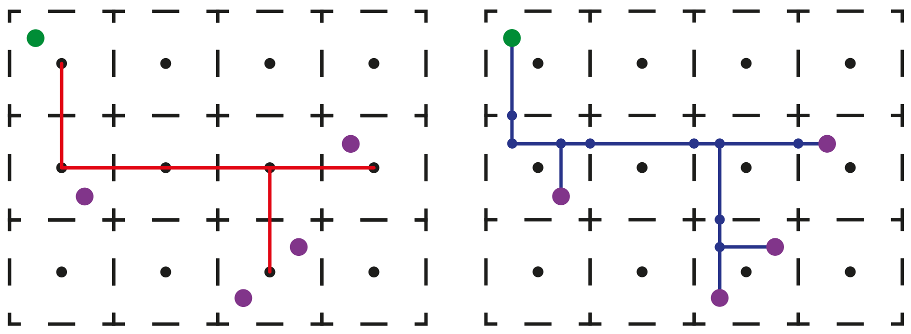


Figure 7.3: Before the global route is used for buffering, we connect to local pinshapes and postoptimize. Vertices that are considered for buffer placement are marked in blue.

the route by connecting all pins to the global route. Then, we use an algorithm of [Kie16] to move all Steiner-segments for minimum total wire length without introducing additional tile-crossings (see Figure 7.3). Thereby, segments are subdivided such that no segment crosses a tile border. Of course two segments may join at a tile border. Besides, we add additional nodes at the start and the end of each passed placement blockage.

7.3.3 Stage 3: Repeater Insertion

Finally, we need to determine where to insert repeaters into the global routes. We combine ideas of dynamic programming for cost minimization [LCT96] and the fast buffering algorithm by [Bar+09], which saves repeaters by postponing the insertion of a repeater when two branches of different polarity are merged, introducing parallel wires. An example of this local topology change is depicted in Figure 7.2(c), where cheaper repeater positions are found by deviating from the initial topology. However, the "fast" running time of [Bar+09] is heavily relying on a uniform routing and placement cost structure on each segment chain. As the cost structure of the routing and placement resources is inhomogeneous, we use dynamic programming with bounded effort similar to [LCT96] to insert repeaters along paths.

So-called *clusters* representing buffered sub-arborescences are propagated from the sinks to the source. A cluster may be partitioned into two subtrees, one with positive and one with negative polarity. This allows to temporarily propagate unmerged sinks of different polarity. Thereby, the input topology is modified to save inverters at the expense of routing. The resource sharing weights are useful to decide whether to perform such a restructuring. Formally, we use labels that represent a buffered subtree. With each label l we store

- cap(l): The load capacitance of the given label up to the next (repeater) sink.

- cost(l): The total routing, placement, power, and delay cost of the subtree solution represented by l.
- slewtarget(l): The maximum slew such that the estimated input slews at the next (repeater) sinks are met.
- inverters(l): The number of inverters inserted in the current subtree.
- sinks(l): A set of pins corresponding to original sinks of the net or input pins of inserted repeaters.
- node(l): The current node in the route graph.
- pos(l): The current position in the chip area.
- pol(l): The polarity '+' or '-' at the current position.

Now a cluster consists of a *label pair* $p = (l, l')$ where pol(l) = '+' and pol(l') = '-'. If sinks(l) $\neq \varnothing$ and sinks(l') $= \varnothing$, we have a label pair of 'ident type'. Otherwise, if sinks(l) $= \varnothing$ and sinks(l') $\neq \varnothing$, we have a label pair of 'invert type'. If both sinks(l) $\neq \varnothing$ and sinks(l') $\neq \varnothing$, the label pair is of 'parallel type', which corresponds to the aforementioned possibility to deviate from the input topology and to propagate a set of unmerged sinks with different polarities upstream. In this case, we will have node(l) = node(l').

For a cluster $p = (l, l')$, we can again define a cost by cost(p) = cost(l) + cost(l'), where we have cost(l) = 0 if sinks(l) $= \varnothing$. Analogously we can define cap(p) = cap(l) + cap(l').

At the net sinks the clusters are initialized canonically with a single label. The only non-canonical choice concerns the slewtarget, which is set to the slew of the last iteration. The algorithm uses three main operations. The most basic is the 'move step' where a cluster is moved along a topology edge. This is where multiple clusters are propagated. Two clusters at the same position can be combined in the 'merge step'. Finally, when we arrive at the source, we have a 'connect source' step.

The repeater insertion obeys further constraints such as capacitance and slew limits. We never place a repeater into a bin with zero placement capacity and lexicographically minimize 1) capacitance violations, 2) slew violations and 3) the resource sharing cost.

When inserting a repeater, we subdivide the global wire at the given position and insert stacked vias to reach the layers of input and output pin of the repeater. In general, the vias will not physically connect to the pin shapes, but the center of the repeater. Sometimes, two consecutive repeaters may be inserted at the same position, leaving subdivided global wires of length zero. The label capacitance is set to the input pin capacitance of the repeater and the slew target to the estimated input slew for the given layer and wire type. Furthermore, the label cost is increased by the resulting placement cost, power cost, and delay through the repeater.

In general, propagating all clusters would lead to an exponential number of labels. We avoid this by pruning the amount of labels at a given position. First, whenever a repeater is inserted into a cluster of parallel type, we enforce its resolution into either indent or invert type. Thus, no particular pruning is needed for clusters of the parallel type.

For non-parallel clusters, we prune by a relaxed dominance check. If we have two non-parallel clusters p, p' at the same position, with the same sink sets and polarity, and $\epsilon_1, \epsilon_2 \geq 0$, p almost dominates p' if $\text{cap}(p) \leq (1+\epsilon_1)\text{cap}(p')$, $\text{slewtarget}(p) \geq (1-\epsilon_1)\text{slewtarget}(p')$, and $\log(\text{cost}(p)) \leq (1+\epsilon_2)\log(\text{cost}(p'))$. We apply the logarithm to the cost, because it depends exponentially on the usage. If $\epsilon_1 = \epsilon_2 = 0$, we have proper dominance. We use $\epsilon_1 = 0.05$ and $\epsilon_2 = 0.1$, without sacrificing the quality significantly.

Furthermore, similar to [LCT96], for each non-parallel cluster type, we subdivide the range of feasible load capacitances into buckets, keeping only the cheapest solution per bucket.

The cheapest solution sometimes has many small repeaters, which are more sensitive to placement legalization in the very end. Thus, we perform one more heuristic pruning. First, we keep only solutions that have no electrical violations and are within a factor of 10 of the minimum cost cluster. Note that a factor of 10 is not much, as prices increase multiplicatively. Then, we remove solutions with more than $\lfloor 1.05n+1 \rfloor$ repeaters, where n is the minimum repeater count among all clusters (buffers are counted twice). Finally, we prune each bucket to a single solution. If this approach would lead to an empty bucket, we keep the cheapest solution in that bucket (i.e. if all solutions have electrical violations).

We found that using 15 buckets yields a good tradeoff between quality and runtime. Thereby, we obtain a guaranteed polynomial runtime for the buffering step.

Move Step

The move step is the most elementary operation in which we propagate clusters bottom-up along the global wires, which are subdivided at tile/bin borders after the post-processing shown in Figure 7.3. In our case, the tile size is well below the optimum repeater spacing, otherwise, a more frequent subdivision could be used. For a cluster $p = (l, l')$, we generate clusters at the next segment position as follows.

First, we generate a label that represents the possibility not to insert any repeater. The slew target of the label is reduced according to the slew degradation on the wire segment. When moving a parallel cluster, we duplicate the global route segment and increase the cost accordingly. If p is of parallel type, we also propagate labels for the option to resolve the parallel segment, i.e. inserting an inverter to equalize polarities. Then, we consider the possibility to insert a repeater for each non-empty label in p. The position of this repeater is not bound to the above segmentation, e.g. at a bin border. Instead, we move the repeaters as far as possible to the beginning of the segment without exceeding the current slew target using binary search. Here, we use the estimated input slew at the repeater input.

Merge Step

The merge step merges cluster candidates from two emanating branches. First, to bound the running time, we prune all except for the cheapest cluster inside each merge branch, adding to the cost a penalty term for inserting a cheapest repeater to drive the current capacitance. Then, we enumerate all possibilities to insert up to two repeaters into the two branch clusters. If a repeater is inserted into a cluster, we require that only one polarity remains and also enumerate all possibilities to drive the merged result with one or two repeaters.

The merging results in at most one parallel cluster, and sets of non-parallel clusters that are pruned as described before.

Connect Source Step

Once the cluster propagation reaches the source, we consider the possibility to insert a final repeater for polarity correction or capacitance reduction, and choose the cheapest resulting solution. As at this time the complete tree has been built, we can evaluate the objective function correctly with estimated input slews.

7.3.4 Stage 3 (Alternative): Buffered Embedding

For most nets, the 3 stage approach gives good results. However, the embedding in Section 7.3.2 has only a rough measure of the placement and delay cost. In particular, in the presence of large placement blockages, the subsequent repeater insertion might not be able to avoid placement congestion, substantial delays or even capacitance and slew violations.

Figure 7.4 shows an example where the embedding algorithm from Section 7.3.3 might have chosen the path in the middle picture due to slightly

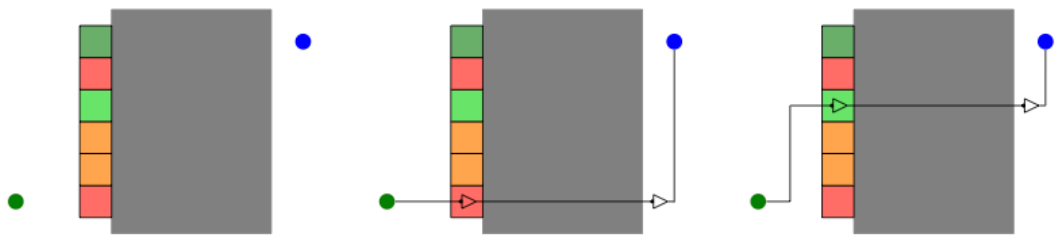


Figure 7.4: The buffered embedding considers placement congestion on tiles before routing over a blockage.

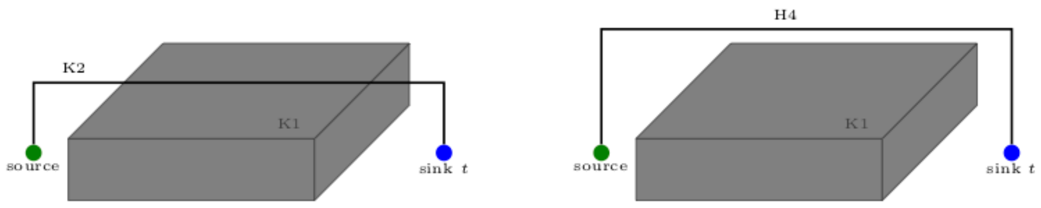


Figure 7.5: The buffered embedding (right) uses higher layers as lower layers would lead to slew violations.

lower routing prices, forcing a repeater insertion into the overfull bin at the left border of the blockage. If routing congestion permits, we would prefer the solution in the right where an under-occupied placement bin can be used. Also, the placement blockages might only be crossed on high layers to meet slew limits, while the algorithm in Section 7.3.2 might choose low layers as shown in Figure 7.5.

For such cases, we employ a more elaborate embedding algorithm that inserts repeaters during the embedding similar to the S-Tree algorithm [HL03]. It was implemented by Natura [Nat17] under the supervision of Rotter [Rot17]. In the following we explain how this multilabel algorithm works. At its core, there is a more runtime intensive embedding process, that already considers effects of placing repeaters while embedding the topology into the routing graph. We capture potential slew and capacitance violations during the embedding process to further facilitate the concluding repeater insertion step described in Section 7.3.3.

As in Section 7.3.2, we proceed bottom-up and embed sibling paths pairwise. Instead of the linearized cost function, we already insert repeaters during the shortest path searches using the method as in Section 7.3.3. This allows us to evaluate the objective function accurately.

As the embedding is still undetermined, we create labels at substantially more vertices in the global routing graph compared to buffering a fixed embedding as in Section 7.3.3. Thus, we reduce the effort for buffering substantially, as follows. We suppress parallel clusters and the binary search during the move step. Furthermore, we keep at most 5 clusters per vertex instead of 15 per cluster modes and vertex.

The time complexity of the buffered embedding algorithm allows only its selective usage. For each net, we first compute a solution with the embedding from Section 7.3.2 and repeater insertion from Section 7.3.3. If this violates slew limits by more than 5%, we embed it again with the buffered embedding algorithm. Finally, we re-buffer the solution by using the approach from Section 7.3.3.

7.4 Improvements

In this section, we list the major changes made to the code since its initial implementation by Rotter [Rot17].

We will distinguish between two types of code changes. On the one hand we have runtime improvements, which speed up some computations but do not change the result. On the other hand we have quality improvements which aim at improving the results.

7.4.1 Runtime Improvements

Unfortunately, the implementation by Rotter used a future cost function, which was not always a lower bound. While this still yields acceptable timing and congestion, it can lead to undesired behavior as zig-zag routes and is therefore out of the question. In all tests performed in this thesis, the incorrect future cost was disabled and only simple estimates (for example, a geometric lower bound), which are guaranteed to be correct, were used.

Dominated Label Cache

We maintain a cache. Every time a new label is created for inserting a repeater, for given current values $\text{round\_cap}(\text{cap}(l)), \text{round\_slew}(\text{slewtarget}(l))$, we check if it was dominated. Here the round_cap and round_slew divide the reasonable range for caps and slews into 128 buckets.

If a certain repeater is detected to be dominated exceedingly often (in 90% of the cases) and at least a minimum amount of total labels have been cached for this cap and slew bucket, we will exclude it from the labeling process.

In practice this allows us to dynamically exclude all repeaters which are not suitable for driving a given capacity. It will also remove repeaters from the labeling process, which are dominated by other choices in the library. In practice we did not observe any change in quality of the results, but obtain a large speedup.

Label Storage Data Type

The old code maintained a linear vector to store the labels at every vertex of the tree to be buffered. After every label insertion this vector was sorted to improve the performance of dominance checks.

To overcome the overhead of constant sorting, we keep the labels sorted by inserting them into a red–black tree [Bay72]. We use the implementation of the C++ STL, which needs $\mathcal{O}(\log n)$ runtime for an insertion operation. This is faster than the previous code both in theory and practice, especially since we allow more labels compared to the old implementation.

Geometric Future Cost Cache

For computing geometric future costs, one has to iterate over possible layers and take via delays into account. As via delays are relatively small, in most cases the lower bound consists of going to the topmost layer and traversing the distance with the fastest available wire type.

To do this lookup as fast as possible, we introduce a separate cache for every pair (l^1, l^2) of start and goal layers. If this pair is fixed, the optimum layer only depends on the geometric distance of the two points (ignoring vertical layers). Within the cache we occasionally sort the delay and via combinations such that branch mispredictions are minimized.

7.4.2 Quality Improvements

Compared to the implementation by Rotter, we incorporated multiple changes aiming at improved solution quality. Besides many fixes and small improvements, we will focus on fundamental changes.

Bounding the Number of Inverters

Previously, the number of inverters was not considered and only the solution cost was optimized. However, we noticed that in many cases the high prices would lead to extra inverters with little benefit. In fact, using an excessive number of gates usually leads to worse solutions, as legalization effects are not fully visible during buffer insertion. To prevent this, we now store the amount of inverters as a property in the labels. All steps are adapted in a way, such that the cheapest solution which does not use an excessive number of inverters is found.

Amount of Labels

The implementation by Rotter only kept 5 labels while propagating in the move step. One label was chosen to have a small capacitance, a different label was chosen with a capacitance that does not exceed the capacitance of an optimally buffered chain, the last 3 labels were selected as the cheapest ones with higher capacitance.

We greatly increase the number of labels kept by subdividing the set of feasible capacitances into 15 buckets. For each of the buckets we keep the cheapest label, which does not use an excessive amount of inverters. This comes at the cost of some runtime, but leads to better buffer solutions. By previous runtime optimizations (in particular the label storage data type), the overhead of this change is acceptable.

Placement Aware Route Computation

The previous implementation did not take placement into account at all while computing the routes to be buffered. On some designs, this could lead to a significant number of avoidable multilabel computations. The new code introduces a modified cost function described above in Equation 7.3. This leads to routes which roughly consider expected costs of inverter insertion and avoid long segments on blockages.

7.4.3 A placement feasibility oracle

Simply considering tile space usages is not sufficient to always obtain legalizable solutions. Consider the situation depicted in Figure 7.7. Here, we are unable to find a legal place for a larger inverter, which spans two circuit rows, due to the blockade structure. Of course such a situation has to be prevented.

To legally place a cell, two constraints have to be satisfied. The obvious one consists of avoiding overlaps with other cells or blockages. In addition, some cells have a *grid constraint.* It is given by a tuple $(a, b) \in \Gamma$, where Γ denotes the set of all grid constraints. Such a pair (a, b) indicates that the top left corner of the cell may only be placed into circuit row $i \in \mathbb{N}$ if

$$i \bmod a = b.$$

In other words, the remainder when dividing i by a has to be b. On our practical instances, there are only two types of grid constraints. Some cells can be placed into every row while others can be placed into every second row.

We will now describe our approach to prevent situations as the one in Figure 7.7. Consider a fixed grid constraint $(a, b) \in \Gamma$. By rounding all blockages to align to the grid induced by the pair (a, b) we are able to find solutions

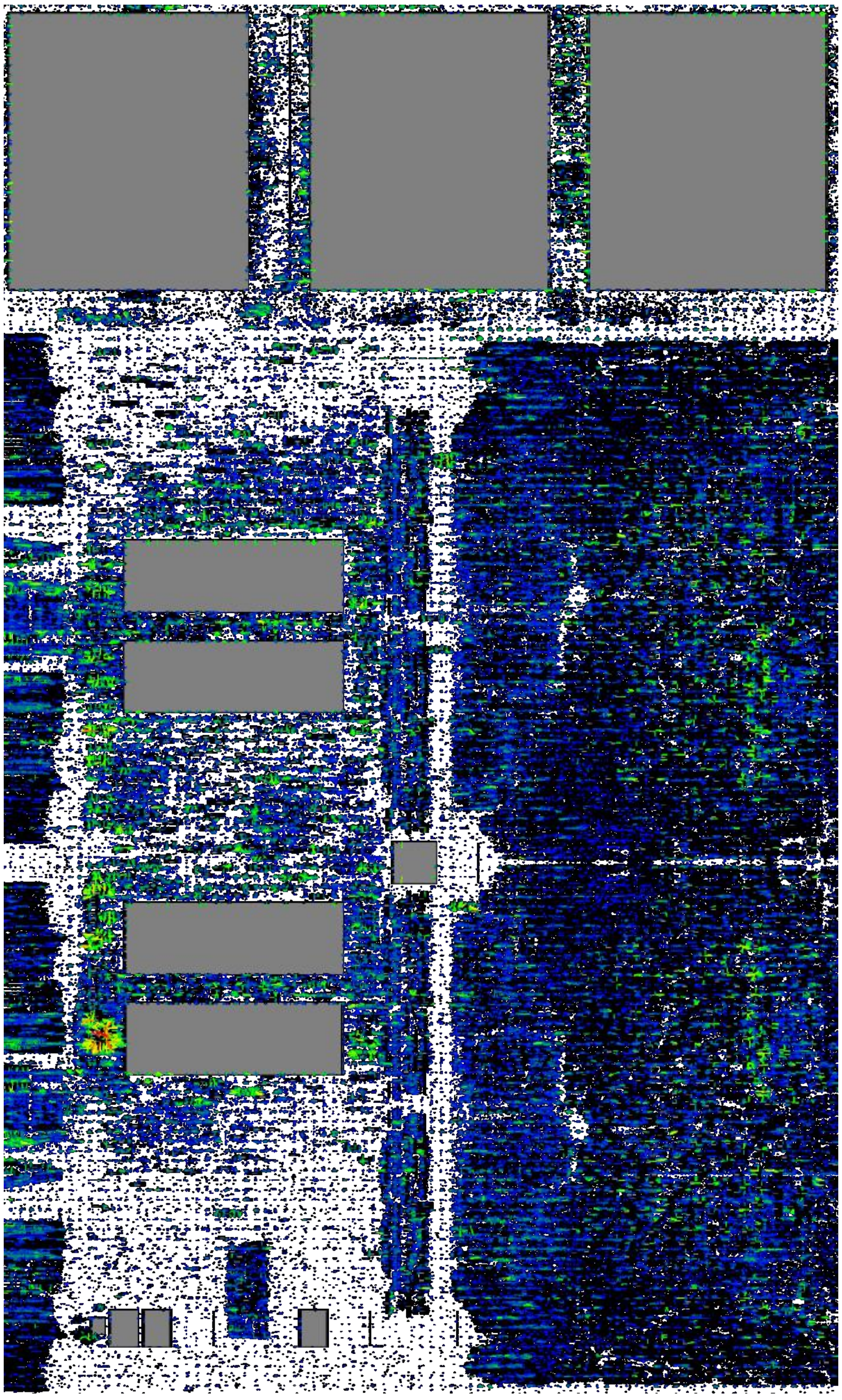


Figure 7.6: Cell movement when legalizing a large design after buffer insertion with an algorithm by Brenner [Bre12]. Lines and colors indicate cell movement. Green and blue corresponds to small movement. Orange and red indicated larger movements.

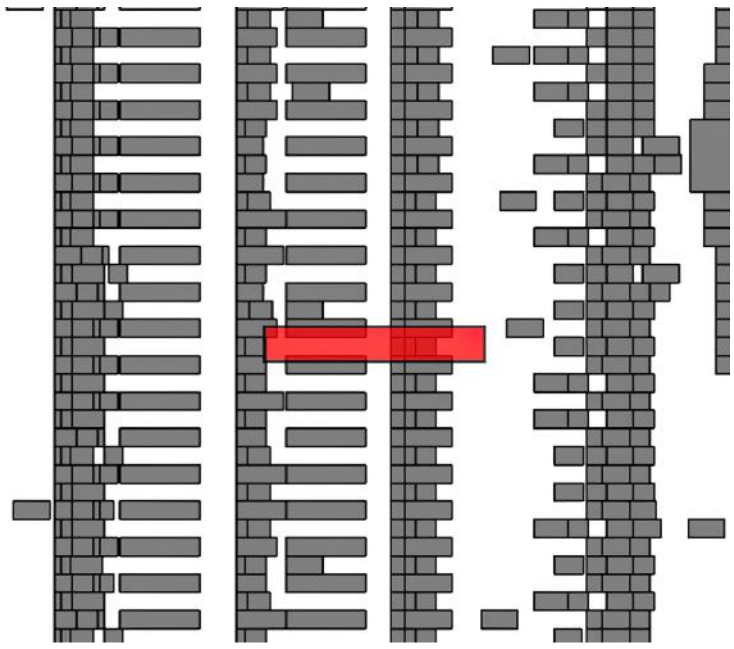


Figure 7.7: A situation that may occur if only area constraints are imposed on the buffering solution. The red cell is to be legalized. While there is a large amount of unused space, no feasible location exists due to the structure of the pre-placed cells.

which automatically satisfy the grid constraint. This operation is illustrated in Figure 7.8. To obtain a legalizable solution, we will first subdivide the chip area into a coarse set. In practice choosing an uniform grid of 256 horizontal and 256 vertical points yields a reasonable tradeoff between runtime, memory and accuracy. Let coarsify($\square$) denote this subdivision of $\square$ into 256^2 points. For every point $P \in$ coarsify($\square$) and grid constraint $\gamma \in \Gamma$, we will compute a value lw(P, γ) which is the maximum width of a cell with grid constraint γ that can be legalized in the neighborhood of P. We will now describe our approach to prevent situations like the one in Figure 7.7.

We roughly proceed as follows. In the first step of Algorithm 7.1 we will round all blockages to a given grid constraint. Then, we compute a stripe-decomposition of the remaining free space. This step is illustrated in Figure 7.9. Finally, we check all points $P \in$ coarsify($\square$) and store the maximum width of a stripe intersecting a rectangle around P. For fast intersection checks, we use a quad tree that contains the corresponding rectangles [FB74].

The old implementation only relied on area capacities. This could lead to very large movements when legalizing some cells. After our adjustments the designs can usually be legalized without any problems. Figure 7.6 shows an excerpt of a larger design when applying a legalization algorithm by Brenner [Bre12] to the output our buffering flow. The spacing map is also used in the

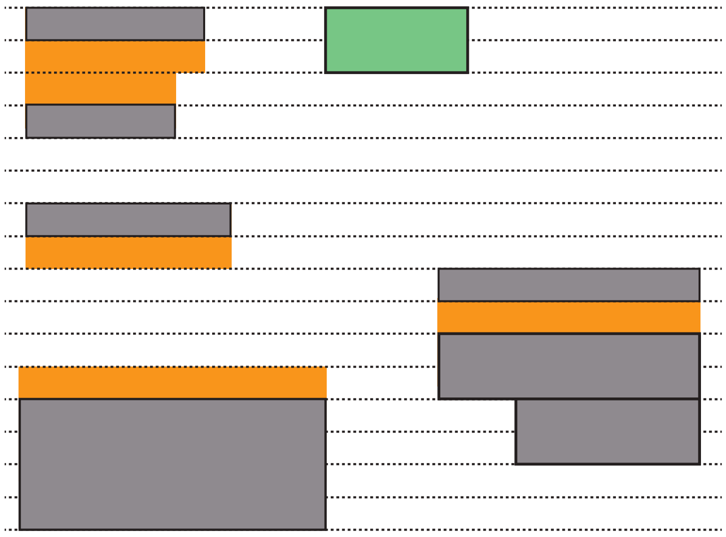


Figure 7.8: Consider the problem of finding a feasible placement position for the green cell. Assuming that the cell can be placed into every odd row, i.e. $\Gamma = \{(2,1)\}$, we can also simply look for an overlap avoiding position in a modified instance. The new instance is constructed by rounding all blockages to align with the respective grid constraint of our cell. The orange areas indicate new blockage outlines after this rounding step.

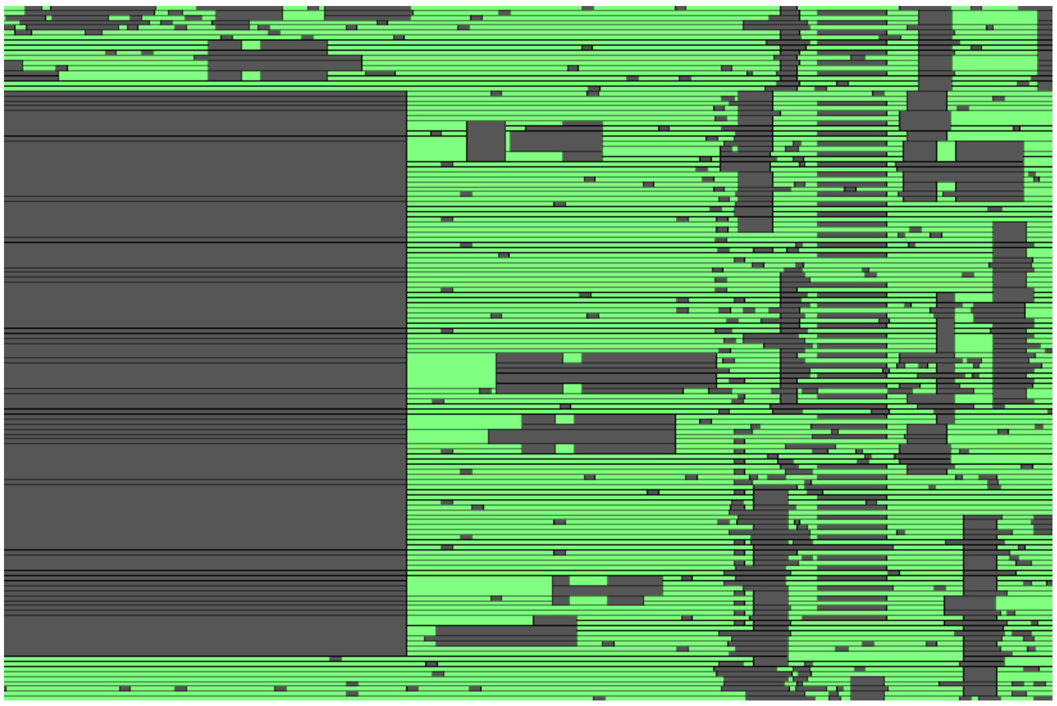


Figure 7.9: A decomposition of the available whitespace on the chip area into stripes.

Algorithm 7.1: Spacing Map Computation

```
   Input: A set of blockages and grid constraints $\Gamma$.
   Output: Local spacings $\mathrm{lw}(P, \rho)$, for all $\gamma \in \Gamma, P \in \mathrm{coarsify}(\square)$
 1 Collect all blockages BLOCKAGES, filling small gaps.
 2 for each grid constraint $\gamma = (a, b) \in \Gamma$ do
 3     Round all blockages to align with $\gamma$, obtain BLOCKAGES$'$
 4     Compute a horizontally maximal stripe-decomposition of the
         whitespace $\square \setminus$BLOCKAGES$'$
 5     Insert $\square \setminus$BLOCKAGES$'$ into a quad tree QTREE
 6     for each point $P$ in a coarse grid of the chip area do
 7         Let $R$ be a rectangle $[-L, L]^2 + P$ (minkowski sum)
 8         Look up all stripes $\mathcal{S}$ intersecting $R$ in QTREE
 9         Let $w$ be the maximum width of a stripe in $\mathcal{S}$
10         Store $\mathrm{lw}(P, \gamma) := w$
```

multilabel algorithm which computes the buffered embedding for difficult nets.

7.5 Comparison to Rotter

We evaluated our speedups of Section 7.4.1 by comparing the modified code with the original implementation by Rotter. Both code variants run without the infeasible future cost and only use geometric lower bounds. The results are presented in table 7.1. Compared to Rotter we achieve large speedups on all instances. The largest instance IBM_14NM_07 previously took more than 24h with 16 threads. Using a larger number of threads it can now be solved in only 3.5 hours.

The timing achieved with the new code is equivalent or better. Other metrics were omitted, as these did not significantly deviate between the two runs. On the larger testcases the step of embedding the topologies is a major bottleneck for both the old and the new code. This is due to the fact that buffering a given topology scales roughly linearly in the input tree, while the embedding step scales approximately quadratically in the side length of the chip.

All runs were performed on an identical machine with two Intel E5-2699 v4 CPUs, each of which offers 22 cores. Hyperthreading was disabled.

7.5.1 Comparison of Path Resources to Arrival Time Customers

As we saw in Section 3.3.2 and 3.3.3, there are two different approaches to model timing constraints in the resource sharing framework. While arrival time

Instance	Branch	#Gates	Ws [ps]	Tns [ns]	$t^{\text{wall}}_{\text{total}}$ [h:m:s]	
Ibm_14nm_01	Rotter	16k	-80.0	-22.1	7:49	
	New	16k	-70.0	-22.3	2:19	-70.4%
Ibm_14nm_02	Rotter	69k	-127.4	-178.3	59:38	
	New	69k	-122.6	-171.6	26:20	-55.8%
Ibm_14nm_03	Rotter	183k	-366.4	-947.0	3:00:38	
	New	182k	-365.5	-873.8	1:44:13	-42.3%
Ibm_14nm_04	Rotter	53k	-174.9	-231.5	46:27	
	New	53k	-178.3	-229.7	17:51	-61.6%
Ibm_14nm_05	Rotter	43k	-161.6	-34.8	21:56	
	New	42k	-162.9	-33.4	7:49	-64.4%
Ibm_14nm_06	Rotter	153k	-187.2	-342.6	1:30:59	
	New	153k	-181.0	-341.9	35:36	-60.9%
Ibm_14nm_07*	Rotter	1770k	-1330.2	-26575.1	11:28:20	
	New	1659k	-444.5	-10390.6	3:25:21	-70.1%

*With 44 threads, all other runs with 16 threads. Old runtime with 16 threads > 24h.

Table 7.1: Runtime comparison on 14nm microprocessor designs provided by IBM.

customers only introduce a polynomial amount of additional resources, timing path resources seem to convey more information and are easier to implement.

The initial implementation by Rotter [Rot17] relied on arrival time customers. For this, Rotter extended an implementation of the arrival time customers by Traub [Tra15]. The bicriteria algorithm used for topology generation required explicit arrival time values which are not available when using timing path resources.

We also integrated timing path resources into BonnRouteBuffer using an implementation by Foos [Foo20] of the topology generation algorithm by Held and Khazraei [HK20] with further improvements. Sink prices are given by the resource sharing costs of all timing path resources to a given sink. Edge costs were heuristically scaled by a uniform constant to obtain similar netlengths as for the bicriteria algorithm. This still leaves room for future improvement. A more sophisticated solution which dynamically chooses the netlength cost dependent on the individual nets would be superior.

To evaluate timing path resources, we used the most recent version of BonnRouteBuffer and switched between arrival time customers and timing path resources.

The results are shown in Table 7.2. We distinguish between three different flows. First, an industrial flow without BonnRouteBuffer is given as a reference (No Brb). Then, we compare the previous topology generation with arrival time customers to the new implementation using timing path resources and cost-based topologies. Timing, power, congestion and netlength are each measured at the end of overall flow.

The results do not show a clear winner and are very similar. However, we remark that our implementation of the timing path resources is significantly simpler than the arrival time customers which use sophisticated bounds and mechanisms to relax constraints. Therefore, the timing path resources should possibly be preferred. With some tuning it may be possible to outperform the initial arrival time customer based topology.

Instance	Run	Ws	Tns	Pstatic	ΔPstatic	Ptotal	ΔPtotal	Gates	Wace4	Netl
		[ps]	[ps]	[mW]		[mW]				
Ibm_7nm_01	No Brb	-295	-60378	3.6		116.7		119614	86.62	100%
	Brb Atc	-289	-59794	3.6	+0.6%	117.1	+0.3%	122424	85.63	97.7%
	Brb Tpr	-301	-60671	3.6	+0.6%	116.8	+0.1%	122134	85.36	97.1%
Ibm_7nm_02	No Brb	-40	-32052	16.9		92.6		44190	87.62	100%
	Brb Atc	-40	-30438	16.7	-1.3%	91.5	-1.2%	43244	87.81	99.0%
	Brb Tpr	-36	-30498	16.6	-1.9%	90.8	-1.9%	42756	87.63	98.8%
Ibm_7nm_03	No Brb	-107	-803058	36.0		124.7		116219	81.63	100%
	Brb Atc	-110	-782491	32.8	-8.9%	121.6	-2.5%	118601	82.77	98.3%
	Brb Tpr	-106	-776328	34.0	-5.6%	122.8	-1.5%	119499	83.23	98.2%
Ibm_7nm_04	No Brb	-114	-62496	17.3		65.0		60427	80.75	100%
	Brb Atc	-111	-56241	16.9	-2.3%	64.5	-0.6%	60399	84.24	98.7%
	Brb Tpr	-114	-57672	17.0	-1.8%	64.6	-0.5%	60617	84.03	98.8%
Ibm_7nm_05	No Brb	-83	-71505	25.1		41.5		142972	90.70	100%
	Brb Atc	-81	-55439	23.4	-6.4%	40.0	-3.8%	143016	91.30	97.5%
	Brb Tpr	-81	-55404	23.4	-6.5%	40.0	-3.8%	143527	90.99	97.5%
Ibm_7nm_06	No Brb	-42	-18024	10.8		150.2		55359	96.73	100%
	Brb Atc	-37	-14404	9.7	-10.4%	145.2	-3.4%	53611	94.39	94.4%
	Brb Tpr	-38	-14029	9.5	-12.5%	144.2	-4.0%	53009	94.38	94.6%
Ibm_7nm_07	No Brb	-52	-1290	8.4		158.1		110893	93.67	100%
	Brb Atc	-52	-1147	7.9	-6.1%	156.4	-1.1%	111374	92.56	97.3%
	Brb Tpr	-51	-1120	7.9	-5.4%	156.2	-1.2%	110664	93.22	97.0%
Ibm_7nm_08	No Brb	-62	-1361	8.3		192.3		120022	91.73	100%
	Brb Atc	-64	-1524	8.2	-1.7%	190.7	-0.8%	119211	91.60	96.9%
	Brb Tpr	-63	-1420	8.1	-3.2%	191.0	-0.6%	118514	91.57	97.1%

Table 7.2: Experimental results comparing arrival time customers to timing path resources on recent 7nm microprocessor designs.

7.6 Global Interconnect Optimization Flow

Our global interconnect optimization algorithm does not size the driver gates of the nets. However, reasonable initial gate sizes are important to avoid electrical violations and reduce the number of needed repeaters. To obtain good initial sizes, we run our algorithm twice.

First, in Steps 1-3 of Algorithm 7.2, we use a quick variant of our buffering algorithm using only 5 resource sharing iterations. Then, we perform global gate sizing [Dab+18a], followed by the main repeater insertion (step 4 of Algorithm 7.2) using 25 resource sharing iterations. Afterwards, we perform gate sizing [Dab+18a] again to reduce the total gate area and, thus, the cell movement during legalization.

Finally, in Steps 6-8 we convert the global routes into net-based layer assignments. The only reason for this phase is that the IBM design flow [Li+12], which is the basis for our experiments, relies heavily on net-based layer-assignments.

Algorithm 7.2: Overall buffering flow

```
  Input: An unbuffered netlist.
  Output: A buffered netlist with a global routing.
  Phase 1: Initial sizing
1 5 iterations routing&buffering+gate sizing on global wires.
2 Rip out all inverters.
3 Legalize placement, discard layer-assignment.
  Phase 2: Buffering
4 25 iterations of routing&buffering.
5 Gate sizing on global wires, then delete wires & legalize.
  Phase 3: PostOpt
6 Compute a net-based layer-assignment.
7 Fix electrical violations & legalize placement.
8 Final layer-assignment based gate-sizing.
```

7.7 Experimental results

The global interconnect optimization algorithm is implemented in C++. Our new flow in Section 7.6 was integrated into the IBM design flow that is based on [Li+12]. We replaced the existing global buffering step. The global buffering step in [Li+12] is followed by refining the signal delays by gate sizing, V_t-optimization, buffering single nets, layer assignment, moving single gates, local logic restructuring, etc., which we did not modify.

The existing global buffering algorithm in [Li+12] first computes a timing-aware layer assignment [Wei+13] and a timing-unaware global routing respecting the layer assignment[MRV11]. Then, it uses the global wires to guide the buffer insertion based on dynamic programming [LCT96; LZS12]. Finally, it calls the same global gate sizing algorithm [Dab+18a].

For placement bins, we set the capacity to 85% of the remaining space after subtracting placement blockages to facilitate placement legalization.

All runs were allowed 16 threads scheduled on a heterogeneous Intel Xeon cluster with clock frequencies between 2.6 and 2.9 GHz. The measured running times still show a tendency but are not entirely comparable. As input, we used ten current mircoprocessor units in 7nm technology.

We compare the existing flow and our new flow once at the end of the global buffering step and at the end of the overall flow, just before detailed routing. The results were provided by IBM and make use of a slightly older version of the code, which does not yet support timing path resources and still uses the old topology generation code.

For the final comparison, we use an industrial signoff timer with the RICE [RP94] delay model. The timing directly after buffering is measured using Elmore delays. Our results are presented in Table 7.3. The first two columns show the name of the instance and the type of the run, followed by the number of gates $|\mathcal{G}|$. The WS columns correspond to the slack of the most critical path. The total negative slack 'TNS' is given by the sum of all negative endpoint slacks, and the power and its difference to the reference in columns PWR and Δ PWR.

Congestion is measured using the ACE4 number [Wei+14]. For $0 < x \leq 100$, [Wei+14] define $\text{Ace}(x)$ to be the average congestion (in percent) of the x percent most congested edges in the global routing graph. The ACE4 metric is then defined by

$$\textsc{Ace4}(x) = \frac{1}{4}\Big(\textsc{Ace}(0.5) + \textsc{Ace}(1) + \textsc{Ace}(2) + \textsc{Ace}(5)\Big).$$

$\text{Ace4} < 93$ indicates routability while designs with $\text{Ace4} > 95$ can hardly be routed or cannot be routed at all.

The number of slew violations is given in the v_{SLW} columns and the load violations in the v_{LD} columns. Finally, columns labeled NETL show the global routing netlength in percentage of the reference run. It is measured with the identical timing-unaware global router. Columns labeled TIME report the running times for global buffering or the overall flow.

The reference algorithm inserts repeaters economically into a global routing that was minimizing the netlength. As we consider timing directly, the number of inserted repeaters, congestion, and the global routing netlength of our algorithm is worse directly after the buffering step. However, the default design flow has to work harder in the subsequent refinement. At the end of the flow, the global routing netlength is significantly improved by our approach, on average by 2.6%. Electrical violations are generally low. On P_02 our algorithm fails to resolve 109 load violations, which can mostly be fixed in the refinement. The slew violations on that instance are reduced from 1486 to 611. For this violation critical design, our multilabel algorithm was used in roughly 0.5% of the oracle calls.

For the routability at the end of the flow, our algorithm shows mostly

		After global buffering										**End of flow**									
INST.	RUN	$\lvert\mathcal{G}\rvert$	WS	TNS	PWR	ΔPWR	ACE4	v_{SLW}	v_{LD}	NETL	TIME	$\lvert\mathcal{G}\rvert$	WS	TNS	PWR	ΔPWR	ACE4	v_{SLW}	v_{LD}	NETL	TIME
			$[ps]$	$[ps]$	$[mW]$						$[h{:}m]$		$[ps]$	$[ps]$	$[mW]$						$[h{:}m]$
P_01	REF.	252k	-124	-19287	9.9		92.5	127	0	100.0%	0:29	221k	-5	-20	9.8		89.3	34	0	100.0%	6:35
	ALG. 7.2	268k	-8	-326	9.6	-2.9%	94.5	98	0	100.7%	4:16	225k	-2	-6	9.6	-1.8%	89.9	24	0	98.0%	10:06
P_02	REF.	109k	-241	-441679	35.3		93.5	1486	2	100.0%	0:12	103k	-87	-178101	37.3		86.8	138	7	100.0%	3:55
	ALG. 7.2	108k	-169	-490888	31.8	-10.1%	92.8	611	109	97.1%	1:15	104k	-84	-166529	37.2	-0.1%	84.5	119	21	98.0%	4:43
P_03	REF.	190k	-235	-532411	109.3		95.8	423	0	100.0%	0:24	202k	-49	-53351	127.9		89.6	103	0	100.0%	5:51
	ALG. 7.2	198k	-407	-498348	110.2	+0.9%	90.7	427	0	99.8%	2:28	202k	-59	-45562	126.8	-0.9%	88.6	135	0	98.9%	7:51
P_04	REF.	84k	-96	-58640	4.3		90.0	381	0	100.0%	0:09	78k	-24	-6374	4.2		82.2	116	0	100.0%	2:44
	ALG. 7.2	96k	-62	-16227	4.2	-2.1%	91.0	209	0	100.8%	1:18	80k	-24	-5748	4.2	-0.2%	84.4	97	0	99.6%	4:00
P_05	REF.	168k	-307	-1358333	33.8		94.9	435	0	100.0%	0:23	177k	-66	-301398	36.0		100.0	109	0	100.0%	5:07
	ALG. 7.2	170k	-205	-968919	31.8	-6.0%	96.2	407	0	100.3%	2:28	176k	-65	-247912	35.8	-0.5%	99.1	88	0	94.0%	6:44
P_06	REF.	18k	-213	-54509	1.2		93.0	201	1	100.0%	0:04	21k	-123	-29098	1.3		90.8	40	5	100.0%	3:00
	ALG. 7.2	20k	-211	-53114	1.2	+2.6%	90.9	162	1	102.7%	0:20	21k	-125	-29064	1.3	+0.0%	89.1	37	4	98.9%	3:18
P_07	REF.	150k	-196	-488710	31.3		93.5	513	1	100.0%	0:25	156k	-45	-119088	32.7		94.5	31	11	100.0%	5:18
	ALG. 7.2	149k	-147	-394686	30.2	-3.4%	93.0	250	4	105.4%	2:12	156k	-36	-88526	32.8	+0.3%	93.3	22	1	97.3%	6:15
P_08	REF.	125k	-236	-333837	31.3		92.7	467	2	100.0%	0:11	129k	-54	-23782	32.1		93.1	82	10	100.0%	5:11
	ALG. 7.2	127k	-152	-189023	29.8	-4.8%	94.6	183	0	101.4%	1:51	128k	-55	-20763	32.0	-0.5%	93.3	62	6	97.2%	7:04
P_09	REF.	268k	-347	-1178450	96.0		110.2	461	0	100.0%	0:47	263k	-98	-312661	100.6		100.1	144	0	100.0%	7:51
	ALG. 7.2	261k	-257	-897348	91.7	-4.5%	96.2	489	1	105.1%	4:55	265k	-85	-221758	100.2	-0.4%	96.3	144	1	95.0%	11:59
P_10	REF.	182k	-181	-404592	85.5		101.0	230	2	100.0%	0:21	162k	-77	-60628	93.4		91.0	13	2	100.0%	6:57
	ALG. 7.2	173k	-239	-366890	83.4	-2.4%	101.2	186	2	102.9%	1:58	160k	-72	-46405	91.3	-2.2%	90.6	13	3	96.7%	8:33

Table 7.3: Experimental results on recent 7nm microprocessor designs provided by IBM.

better results on routing critical instances (ACE4 ≥ 93). On P_09 the ACE4 is improved from 100.1 to 96.3, which brings the design from being unroutable to possibly being routable.

Despite inserting more gates and increasing the netlength, our algorithm improves the power consumption and the TNS directly after global buffering. The timing improvements are persistent at the end of the flow, on P_05, P_07, P_09 and P_10 drastically. On P_01 we can almost satisfy all timing constraints after buffering while the reference flow has a TNS of -19287 ps.

The running time of our global interconnect optimization increases compared to the reference flow. However, our algorithm scales well using more CPU cores, routing nets within each iteration in parallel. With 44 cores we still observed near-optimum scaling on large instances. Further speed-ups are still possible by refining our implementation. Finally, the refinement flow after global buffering has not been adjusted to the improved starting solution yet.

Overall, our industrial partner IBM was extremely satisfied with the improvements obtained by the algorithm. Especially the netlength savings were surprising as previously even a reduction by 1% was considered a major improvement. Usually, changes that are made early in the flow only have mild implications on the final results. However, as we demonstrated above our new buffering solutions significantly improves many metrics of the final result. As a consequence the new algorithm was quickly adapted to be the new default buffering tool.

Chapter 8

Summary

Resource sharing problems are ubiquitous in the chip design process. In this thesis, we considered three such problems in detail: The time-cost tradeoff problem, the gate sizing problem and the buffered, timing-constrained global routing problem. In each of these, timing constraints have to be met while minimizing the use of other resources like power, netlength or edge congestion.

For the time-cost tradeoff problem, we presented a primal-dual V_t optimization algorithm with a provable performance guarantee. Our implementation does not require a separable delay function. In practice, it achieved leakage reductions of up to 8% on netlists that were pre-optimized by one of the most successful algorithms for gate sizing and V_t assignment [RSR16a]. When applied after global routing, the savings were even larger. On top of the approximation guarantee, we computed a posteriori lower bounds which show that we solve several instances almost optimally.

Besides the practical algorithm, we also considered the theoretic time-cost tradeoff problem in directed acyclic graphs of bounded depth d. Previously, a simple d approximation was known [Dab+18b]. First, we presented a $\frac{d}{2}+(\lceil\frac{d}{2}\rceil-\frac{d}{2})\frac{1}{d}$ algorithm. We then note that the problem can be seen as a vertex cover problem in d-partite hypergraphs. This implies a $\frac{d}{2}+\epsilon$ approximation for any $\epsilon > 0$ by a result of Lovász [Lov75]. We improve this further, by presenting a randomized algorithm of guarantee slightly better than $\frac{d}{2}$. It also approximates the d-partite hypergraph vertex cover problem, where it best possible (up to $o(1)$ terms) under the Unique Games Conjecture and $\mathrm{P} \neq \mathrm{NP}$.

From a negative side, previously only APX-hardness was known for the special case of bounded depth. We improve this lower bound to $\frac{d+2}{4}$, again assuming the Unique Games Conjecture and $\mathrm{P} \neq \mathrm{NP}$. Thereby, we settle the approximability of the problem up to a factor of less than 2.

Next, we considered the gate sizing problem, which adds a further difficulty of interdependent delays. To solve it, we used the resource sharing formulation by Schorr [Sch15]. Our new analysis of the resource sharing algorithm applied to gate sizing rules out previous inaccuracies and can be easily adapted to further delay models. We presented a new practical implementation of the resource sharing algorithm for gate sizing with heuristic modifications and compared it to the state-of-the-art algorithm of Reimann et al. [RSR16a]. On all designs our algorithm obtains similar or better power savings while significantly reducing runtime up to a factor of 10.

Finally, we presented BonnRouteBuffer. Rotter explained how to heuristically solve the problem of computing buffered global routes for all nets with a single resource sharing formulation [Rot17]. We extended and simplified his model. First, we used timing path resources which were first described by Hähnle [Häh15] to allow an easier implementation. Our experiments indicate that this does not affect the overall solution quality. Next, we pointed out that simply using area based resources to attain placement feasibility is insufficient and adjusted the theoretic model accordingly.

Rotter presented a first proof-of-concept implementation of his algorithm [Rot17]. However, overall his practical results were not yet completely convincing. Next to improved placement constraints we applied several new ideas to obtain better solutions. As a result, the algorithm now decisively outperforms a state-of-the-art design flow in almost all metrics, including netlength, power, congestion and timing. We also drastically improve the performance, obtaining runtime reductions up to 70%.

All three described algorithms are now in use by our industrial partner IBM, they significantly reshaped the optimization methodology used in the design flow. While the previous algorithms mostly relied on local optimization, our new flow is driven by global problem formulations which take all major design objectives into account.

Bibliography

[Abr+11] H. Abrishami, J. Lou, J. Qin, J. Froessl, and M. Pedram. “Post sign-off leakage power optimization”. In: *Proceedings of the 48th Design Automation Conference*. DAC (San Diego, California, USA). 2011, pp. 453–458 (cit. on pp. 38, 39).

[Ach+12] H. Acharya, T. Choi, R. Bazzi, and M. Gouda. “The K-observer problem in computer networks”. In: *Networking Science* 1 (2012), pp. 15–22 (cit. on p. 76).

[AHK96] R. Aharoni, R. Holzman, and M. Krivelevich. “On a theorem of Lovász on covers in r-partite hypergraphs”. In: *Combinatorica* 16.2 (1996), pp. 149–174 (cit. on pp. 57, 59, 60, 63, 69).

[AK00] P. Alimonti and V. Kann. “Some APX-completeness results for cubic graphs”. In: *Theoretical Computer Science* 237.1-2 (2000), pp. 123–134 (cit. on pp. 59, 85).

[Alb+02] C. Albrecht, A. B. Khang, I. Mandoiu, and A. Zelikovsky. “Floorplan Evaluation with Timing-Driven Global Wireplanning, Pin Assignment and Buffer/Wire Sizing”. In: *Proceedings of the 7th Asia and South Pacific Design Automation Conference*. ASP-DAC (Bangalore, India). 2002, pp. 580–588 (cit. on pp. 20, 107, 108).

[Alp+18] C. Alpert, W.-K. Chow, K. Han, A. Kahng, Z. Li, D. Liu, and S. Venkatesh. “Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees”. In: *Proceedings of the 2018 International Symposium on Physical Design*. ISPD (Monterey, California, USA). 2018, pp. 10–17 (cit. on p. 107).

[Alp+95] C. Alpert, H. J.H., A. Kahng, and D. Karger. “Prim-Dijkstra tradeoffs for improved performance-driven routing tree design”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14.7 (1995), pp. 890–896 (cit. on p. 107).

[Aut+17] C. Auth, A. Aliyarukunju, M. Asoro, D. Bergstrom, V. Bhagwat, J. Birdsall, N. Bisnik, M. Buehler, V. Chikarmane, G. Ding, Q. Fu, H. Gomez, W. Han, D. Hanken, M. Haran, M. Hattendorf, R. Heussner, H. Hiramatsu, B. Ho, S. Jaloviar, I. Jin, S. Joshi,

S. Kirby, S. Kosaraju, H. Kothari, G. Leatherman, K. Lee, J. Leib, A. Madhavan, K. Marla, H. Meyer, T. Mule, C. Parker, S. Parthasarathy, C. Pelto, L. Pipes, I. Post, M. Prince, A. Rahman, S. Rajamani, A. Saha, J. D. Santos, M. Sharma, V. Sharma, J. Shin, P. Sinha, P. Smith, M. Sprinkle, A. Amour, C. Staus, R. Suri, D. Towner, A. Tripathi, A. Tura, C. Ward, and A. Yeoh. "A 10nm high performance and low-power CMOS technology featuring 3rd generation FinFET transistors, Self-Aligned Quad Patterning, contact over active gate and cobalt local interconnects". In: *Proceedings of the International Electron Devices Meeting* (San Francisco, California, USA). 2017, pp. 673–676 (cit. on p. 2).

[Bar+09] C. Bartoschek, S. Held, D. Rautenbach, and J. Vygen. "Fast Buffering for Optimizing Worst Slack and Resource Consumption in Repeater Trees". In: *Proceedings of the 2009 International Symposium on Physical Design*. ISPD (San Diego, California, USA). 2009, pp. 43–50 (cit. on pp. 35, 108, 112, 115).

[Bar+10] C. Bartoschek, S. Held, J. Maßberg, D. Rautenbach, and J. Vygen. "The repeater tree construction problem". In: *Information Processing Letters* 110.24 (2010), pp. 1079–1083 (cit. on pp. 33, 107).

[Bay72] R. Bayer. "Symmetric binary B-Trees: Data structure and maintenance algorithms". In: *Acta Informatica* 1.4 (1972), pp. 290–306 (cit. on p. 121).

[BE81] R. Bar-Yehuda and S. Even. "A linear-time approximation algorithm for the weighted vertex cover problem". In: *Theoretical Computer Science* 2.2 (1981), pp. 198–203 (cit. on pp. 37, 42, 59, 65).

[BJ08] S. Boyd and S. Joshi. "An efficient method for large-scale gate sizing". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 55.9 (2008), pp. 2760–2773 (cit. on p. 90).

[Boy+05] S. Boyd, S. Kim, D. Patil, and M. Horowitz. "Digital circuit optimization via geometric programming". In: *Operations Research* 53.6 (2005), pp. 899–932 (cit. on p. 90).

[Bre+11] B. Brešar, F. Kardoš, J. Katrenič, and G. Semanišin. "Minimum k-path vertex cover". In: *Discrete Applied Mathematics* 159.12 (2011), pp. 1189–1195 (cit. on p. 76).

[Bre12] U. Brenner. “VLSI legalization with minimum perturbation by iterative augmentation”. In: *Proceedings of the 2012 Conference on Design, Automation and Test in Europe.* DATE (Dresden, Germany). 2012, pp. 1385–1390 (cit. on pp. 123, 124).

[Bro41] R. Brooks. “On colouring the nodes of a network”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 37.2 (1941), pp. 194–197 (cit. on p. 85).

[CC91] R. C. Carden and C.-K. Cheng. “A global router using an efficient approximate multicommodity multiterminal flow algorithm”. In: *Proceedings of the 28th Design Automation Conference.* DAC (San Francisco, California, USA). 1991, pp. 316–321 (cit. on pp. 20, 21).

[CCW99] C. Chen, C. Chu, and D. Wong. “Fast and exact simultaneous gate and wire sizing by Lagrangian relaxation”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18.7 (1999), pp. 1014–1025 (cit. on pp. 90, 93, 94, 99, 100).

[Cha+10] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang. “NTHU-Route 2.0: A Robust Global Router for Modern Designs”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.12 (2010), pp. 1931–1944 (cit. on p. 20).

[Cho+09] M. Cho, K. Lu, K. Yuan, and D. Z. Pan. “BoxRouter 2.0: A Hybrid and Robust Global Router with Layer Assignment for Routability”. In: *ACM Trans. Des. Autom. Electron. Syst.* 14.2 (2009), Art. No. 32 (cit. on p. 20).

[Chu+08] J. Chuzhoy, A. Gupta, J. Naor, and A. Sinha. “On the approximability of some network design problems”. In: *Transactions on Algorithms* 4.2 (2008), Art. No. 23 (cit. on pp. 33, 35).

[CK05] D. Chinnery and K. Keutzer. “Linear Programming for Sizing, Vth and Vdd Assignment”. In: *Proceedings of the 2005 International Symposium on Low Power Electronics and Design.* ISLPED (San Diego, California, USA). 2005, pp. 149–154 (cit. on p. 90).

[CW01] C. Chu and D. Wong. “VLSI Circuit Performance Optimization by Geometric Programming”. In: *Annals of Operations Research* 105.1-4 (2001), pp. 37–60 (cit. on pp. 90, 94).

[CW03] J. Croix and D. Wong. “Blade and razor: cell and interconnect delay analysis using current-based models”. In: *Proceedings of the 40th Design Automation Conference.* DAC (Anaheim, California, USA). 2003, pp. 386–389 (cit. on p. 13).

[CY19] G. Chen and E. Young. “SALT: Provably Good Routing Topology by a Novel Steiner Shallow-Light Tree Algorithm”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019). Early Access (cit. on p. 107).

[Dab+18a] S. Daboul, N. Hähnle, S. Held, and U. Schorr. “Provably Fast and Near-Optimum Gate Sizing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.12 (2018). DOI: 10.1109/TCAD.2018.2801231, © 2018 IEEE, pp. 3163–3176 (cit. on pp. 29, 89, 91, 94, 98–100, 102, 129).

[Dab+18b] S. Daboul, S. Held, J. Vygen, and S. Wittke. “An approximation algorithm for threshold voltage optimization”. In: *ACM Transactions on Design Automation of Electronic Systems* 23.6 (2018). Art. No. 68, DOI: https://doi.org/10.1145/3232538 (cit. on pp. 37, 59, 133).

[Dab+19] S. Daboul, S. Held, B. Natura, and D. Rotter. “Global Interconnect Optimization”. In: *IEEE/ACM International Conference on Computer-Aided Design.* ICCAD (Westminster, Colorado, USA). DOI: 10.1109/ICCAD45719.2019.8942155. 2019, Art. No. 1C.3, pp. 1-8 (cit. on p. 105).

[Dab15] S. Daboul. “Algorithms for the gate sizing and Vt assignment problem”. Master’s thesis. University of Bonn, 2015 (cit. on p. 16).

[De+97] P. De, E. Dunne, J. Ghosh, and C. Wells. “Complexity of the discrete time-cost tradeoff problem for project networks”. In: *Operations Research* 45.2 (1997), pp. 302–306 (cit. on p. 58).

[DG01] V. Deĭneko and W. G.J. “Hardness of approximation of the discrete time-cost tradeoff problem”. In: *Operations Research Letters* 29.5 (2001), pp. 207–210 (cit. on pp. 59, 85).

[DHV20] S. Daboul, S. Held, and J. Vygen. “Approximating the discrete time-cost tradeoff problem with bounded depth”. arXiv 2011.02446. 2020 (cit. on pp. 57, 64, 70, 72, 78, 80–82).

[DS05] I. Dinur and S. Safra. “On the hardness of approximating minimum vertex cover”. In: *The Annals of Mathematics* 162.1 (2005), pp. 439–485 (cit. on p. 58).

[EF70] J. Edmonds and D. Fulkerson. “Bottleneck extrema”. In: *Journal of Combinatorial Theory* 8.3 (1970), pp. 299–306 (cit. on p. 61).

[EK72] J. Edmonds and R. M. Karp. “Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems”. In: *Journal of the ACM (JACM)* 19.2 (1972), pp. 248–264 (cit. on p. 73).

[Elm48] W. C. Elmore. “The transient response of damped linear networks with particular regard to wide-band amplifiers”. In: *Journal of Applied Physics* 19.1 (1948), pp. 55–64 (cit. on pp. 11, 110).

[FB74] R. A. Finkel and J. L. Bentley. “Quad trees a data structure for retrieval on composite keys”. In: *Acta Informatica* 4.1 (1974), pp. 1–9 (cit. on p. 124).

[FD85] J. Fishburn and A. Dunlop. “Tilos: a posynomial programming approach to transistor sizing”. In: ICCAD (Santa Clara, California, USA). 1985, pp. 326–328 (cit. on pp. 11, 13, 42, 90).

[Fla+14] G. Flach, T. Reimann, G. Posser, M. Johann, and R. Reis. “Effective Method for Simultaneous Gate Sizing and Vt Assignment Using Lagrangian Relaxation”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33.4 (2014), pp. 546–557 (cit. on pp. 38, 50, 89, 90).

[Foo20] J. Foos. “Algorithms for buffered global routing”. Master’s thesis. University of Bonn, 2020 (cit. on pp. 113, 127).

[FW56] M. Frank and P. Wolfe. “An algorithm for quadratic programming”. In: *Naval Research Logistics Quarterly* 3.1-2 (1956), pp. 95–110 (cit. on p. 94).

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. “The ellipsoid method and its consequences in combinatorial optimization”. In: *Combinatorica* 1 (1981), pp. 169–197 (cit. on p. 62).

[GSS00] V. Guruswami, S. Sachdeva, and R. Saket. “Inapproximability of minimum vertex cover on k-uniform k-partite hypergraphs”. In: *SIAM Journal on Discrete Mathematics* 29.1 (2000), pp. 36–58 (cit. on pp. 59, 60, 75).

[GW04] A. Grigoriev and G. Woeginger. “Project scheduling with irregular costs: complexity, approximability, and algorithms”. In: *Acta Informatica* 41.2 (2004), pp. 83–97 (cit. on p. 58).

[GW96] M. Goemans and D. Williamson. “The primal-dual method for approximation algorithms and its application to network design problems”. In: *Approximation Algorithms for NP-Hard Problems.* Ed. by D. Hochbaum. 1996, pp. 144–191 (cit. on p. 47).

[Häh15] N. Hähnle. *Time-cost tradeoff and Steiner tree packing with multiplicative weights.* Technical Report no. 1511115. Research Institute for Discrete Mathematics, University of Bonn, 2015 (cit. on pp. 5, 21, 29, 31, 97, 105, 134).

[Hel+17] S. Held, S. Müller, D. Rotter, R. Scheifele, V. Traub, and J. Vygen. “Global Routing with Timing Constraints”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.2 (2017), pp. 406–419 (cit. on pp. 21, 26–28, 105, 112, 114).

[Hel08] S. Held. “Timing closure in chip design”. Dissertation. University of Bonn, 2008 (cit. on p. 7).

[Hel09] S. Held. “Gate Sizing for Large Cell-Based Desings”. In: *Proceedings of the 2009 Conference on Design, Automation and Test in Europe*. DATE (Nice, France). 2009, pp. 827–832 (cit. on p. 90).

[HH16] S. Held and J. Hu. “Gate Sizing”. In: *Electronic Design Automation for Integrated Circuits Handbook, Second Edition.* Ed. by L. Scheffer, L. Lavagno, G. Martin, and I. Markov. 2016, Chapter 33 (cit. on pp. 12, 13, 90).

[HK20] S. Held and A. Khazraei. “An Improved Approximation Algorithm for the Uniform Cost-Distance Steiner Tree Problem”. In: *WAOA: International Workshop on Approximation and Online Algorithms* (Online). 2020 (cit. on pp. 33, 113, 127).

[HL03] M. Hrkić and J. Lillis. “Buffer tree synthesis with consideration of temporal locality, sink polarity requirements, solution cost, congestion, and blockages”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22.4 (2003), pp. 481–491 (cit. on pp. 107, 119).

[Hon+97] X.-L. Hong, T. Xue, J. Huang, C.-K. Cheng, and E. S. Kuh. “TIGER: an efficient timing-driven global router for gate array and standard cell layout design”. In: vol. 16. 11. 1997, pp. 1323–1331 (cit. on p. 20).

[HR13] S. Held and D. Rotter. “Shallow Light Steiner Arborescences with Vertex Delays”. In: *Proceedings of the 16th International Conference on Integer Programming and Combinatorial Optimization.* IPCO (Valparaíso, Chile). 2013, pp. 996–1025 (cit. on pp. 32, 107, 113).

[HR18] S. Held and B. Rockel. “Exact Algorithms for Delay-Bounded Steiner Arborescences”. In: *Proceedings of the 55th Design Automation Conference.* DAC (San Francisco, California, USA). 2018, Art. No. 44 (cit. on pp. 107, 113).

[HS19] N. Hähnle and P. Saccardi. “Global Routing on Rhomboidal Tiles”. In: *IEEE/ACM International Conference on Computer-Aided Design.* ICCAD (Westminster, Colorado, USA). 2019, Art. No. 10C.1 (cit. on pp. 20, 21).

[HSV17] S. Hougardy, J. Silvanus, and J. Vygen. “Faster min-max resource sharing in theory and practice”. In: *Mathematical Programming Computation* 9.7 (2017), pp. 135–202 (cit. on p. 107).

[Hu+12] J. Hu, A. Kahng, S. Kang, M.-C. Kim, and I. Markov. “Sensitivity-Guided Metaheuristics for Accurate Discrete Gate Sizing”. In: *Proceedings of the 2012 International Conference on Computer-Aided Design.* ICCAD (San Jose, California, USA). 2012, pp. 233–239 (cit. on pp. 38, 90).

[Hu+18] J. Hu, Y. Zhou, Y. Wei, S. Quay, L. Reddy, G. Téllez, and G.-J. Nam. “Interconnect Optimization Considering Multiple Critical Paths”. In: *Proceedings of the 2018 International Symposium on Physical Design.* ISPD (Monterey, California, USA). 2018, pp. 132–138 (cit. on p. 108).

[Hua+93] J. Huang, X.-L. Hong, C.-K. Cheng, and E. S. Kuh. “An efficient timing-driven global routing algorithm”. In: *Proceedings of the 30th Design Automation Conference.* DAC (Anaheim, California, USA). 1993, pp. 596–600 (cit. on p. 20).

[Isb58] J. Isbell. “A class of simple games”. In: *Duke Mathematical Journal* 25.3 (1958), pp. 423–439 (cit. on p. 61).

[Jag13] M. Jaggi. “Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization”. In: *Proceedings of the 30th International Conference on Machine Learning.* Ed. by S. Dasgupta and D. McAllester. Vol. 28. 1. 2013, pp. 427–435 (cit. on p. 95).

[JJC99] H. Jiang, J. Jou, and Y. Chang. “Noise-Constrained Performance Optimization, by Simultaneous Gate and Wire Sizing Based on Lagrangian Relaxation”. In: *Proceedings of the 36th Design Automation Conference.* DAC (New Orleans, Louisiana, USA). 1999, pp. 90–95 (cit. on pp. 99, 101).

[Kah+13] A. Kahng, S. Kang, H. Lee, I. Markov, and P. Thapar. “High-performance gate sizing with a signoff timer”. In: *Proceedings of the 2013 International Conference on Computer-Aided Design.* ICCAD (San Jose, California, USA). 2013, pp. 450–457 (cit. on pp. 14, 38, 90).

[Kho02] S. Khot. “On the power of unique 2-prover 1-round games”. In: (2002), pp. 767–775 (cit. on p. 4).

[Kie16] A. Kiefner. “Minimizing path lengths in rectilinear Steiner minimum trees with fixed topology”. In: *Operations Research Letters* 44.6 (2016), pp. 835–838 (cit. on p. 115).

[KK82] N. Karmarkar and R. Karp. “An efficient approximation scheme for the one-dimensional bin-packing problem”. In: *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*. FOCS (Chicago, Illinois, USA). 1982, pp. 312–320 (cit. on p. 63).

[Kon02] T. Kong. “A novel net weighting algorithm for timing-driven placement”. In: ICCAD (San Jose, California, USA). 2002, pp. 172–176 (cit. on p. 16).

[KR08] S. Khot and O. Regev. “Vertex cover might be hard to approximate to within 2-epsilon”. In: *Journal of Computer and System Sciences* 74.3 (2008), pp. 335–349 (cit. on pp. 4, 76).

[KRY95] S. Khuller, B. Raghavachari, and N. Young. “Balancing minimum spanning trees and shortest-path trees”. In: *Algorithmica* 14 (1995), pp. 305–321 (cit. on p. 33).

[KV11] B. H. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics)*. Springer, 5th ed. 2011. 660 pp. (cit. on pp. 7, 149).

[KW16] N. Klein and T. Wexler. “On the Approximability of DAG Edge Deletion”. Manuscript. 2016 (cit. on p. 66).

[KW59] J. Kelley and M. Walker. “Critical-Path Planning and Scheduling”. In: *Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference* (Boston, Massachusetts, USA). 1959, pp. 160–173 (cit. on pp. 3, 58).

[Lan00] K. Langkau. “Gate Sizing in VLSI Design”. Diploma thesis (in German). University of Bonn, 2000 (cit. on pp. 89, 90).

[LCT96] J. Lillis, C. Cheng, and L. T.T.Y. “Optimal Wire Sizing and Buffer Insertion for Low Power and a Generalized Delay Model”. In: *Journal of Solid-State Circuits* 31.3 (1996), pp. 436–447 (cit. on pp. 106, 107, 115, 117, 129).

[LF08] F. Liu and P. Feldmann. “MAISE: An Interconnect Simulation Engine for Timing and Noise Analysis”. In: *9th International Symposium on Quality of Electronic Design*. ISQED (San Jose, California, USA). 2008, pp. 621–626 (cit. on p. 50).

[LH10] Y. Liu and J. Hu. “A new algorithm for simultaneous gate sizing and threshold voltage assignment”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.2 (2010), pp. 223–234 (cit. on p. 39).

[Li+08] Z. Li, C. J. Alpert, S. Hu, T. Muhmud, S. T. Quay, and P. G. Villarrubia. “Fast interconnect synthesis with layer assignment”. In: *Proceedings of the 2008 International Symposium on Physical Design*. ISPD (Portland, Oregon, USA). 2008, pp. 71–77 (cit. on p. 107).

[Li+12] Z. Li, C. Alpert, G.-J. Nam, C. Sze, N. Viswanathan, and N. Zhou. “Guiding a physical design closure system to produce easier-to-route designs with more predictable timing”. In: *Proceedings of the 49th Design Automation Conference*. DAC (San Francisco, California, USA). 2012, pp. 465–470 (cit. on pp. 2, 17, 21, 129).

[Liu+18] D. Liu, B. Yu, S. Chowdhury, and D. Pan. “TILA-S: Timing-driven incremental layer assignment avoiding slew violations”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.1 (2018), pp. 231–244 (cit. on p. 108).

[Lov75] L. Lovász. “On minmax theorems of combinatorics”. Doctoral Thesis (in Hungarian). Bolyai Institute, József Attila University, 1975 (cit. on pp. 57, 59, 60, 69, 133).

[LZS12] Z. Li, Y. Zhou, and W. Shi. “$O(mn)$ Time Algorithm for Optimal Buffer Insertion of Nets With m Sinks”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.3 (2012), pp. 437–441 (cit. on pp. 106, 107, 129).

[Maß15] J. Maßberg. “The Rectilinear Steiner Tree Problem with Given Topology and Length Restrictions”. In: *Computing and Combinatorics*. 2015, pp. 445–456 (cit. on p. 107).

[MRV11] D. Müller, K. Radke, and J. Vygen. “Faster min-max resource sharing in theory and practice”. In: *Mathematical Programming Computation* 3.1 (2011), pp. 1–35 (cit. on pp. 20, 23–26, 96, 102, 110, 111, 129).

[MSW14] M. Mihalák, R. Srámek, and P. Widmayer. “Approximately counting approximately shortest paths in directed acyclic graphs”. In: *Theory of Computing Systems* 58.1 (2014), pp. 1–15 (cit. on p. 16).

[Nat17] B. Natura. “Algorithms for Routing and Buffer Insertion”. Master Thesis. University of Bonn, 2017 (cit. on pp. 36, 105, 107, 113, 119).

[Ngu+03] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer. “Minimization of Dynamic and Static Power through Joint Assignment of Threshold, Voltages and Sizing Optimization”. In: *Proceedings of the 2003 International Symposium* *on Low Power Electronics and Design*. ISLPED (Seoul, South Korea). 2003, pp. 158–163 (cit. on p. 90).

[OBH12] M. Ozdal, S. Burns, and J. Hu. “Algorithms for Gate Sizing and Device Parameter Selection for High-Performance Designs”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31.10 (2012), pp. 1558–1571 (cit. on pp. 39, 90).

[Ozd+13] M. Ozdal, C. Amin, A. Ayupov, S. M. Burns, G. R. Wilke, and C. Zhuo. “An Improved Benchmark Suite for the ISPD-2013 Discrete Cell Sizing Contest”. In: *Proceedings of the 2013 International Symposium on Physical Design*. ISPD (Stateline, Nevada, USA). 2013, pp. 168–170 (cit. on p. 14).

[PRS94] D. Paik, S. Reddy, and S. Sahni. “Deleting vertices to bound path length”. In: *Transactions on Computers* 43.9 (1994), pp. 1091–1096 (cit. on p. 76).

[RD13] H. Ren and S. Dutt. “Fast and Near-Optimal Timing-Driven Cell Sizing under Cell Area and, Leakage Power Constraints Using a Simplified Discrete Network Flow Algorithm”. In: *VLSI Design* (2013) (cit. on p. 90).

[Roc18] B. Rockel. “Exact Algorithms for Interconnect Optimization”. Master’s thesis. University of Bonn, 2018 (cit. on p. 107).

[Rot17] D. Rotter. “Timing-Constrained Global Routing with Buffered Steiner Trees”. Dissertation. University of Bonn, 2017 (cit. on pp. 5, 21, 33–35, 105–107, 110, 112, 113, 119, 120, 127, 134).

[RP94] C. Ratzlaff and L. Pillage. “RICE: Rapid Interconnect Circuit Evaluation Using Asymptotic Waveform Evaluation”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13.6 (1994), pp. 736–776 (cit. on pp. 13, 130).

[RS12] M. Rahman and C. Sechen. “Post-Synthesis Leakage Power Minimization”. In: *Proceedings of the 2012 Conference on Design, Automation and Test in Europe*. DATE (Dresden, Germany). 2012, pp. 99–104 (cit. on pp. 38, 39).

[RSR15] T. Reimann, C. Sze, and R. Reis. “Gate sizing and threshold voltage assignment for high performance microprocessor designs”. In: *Proceedings of the 20th Asia and South Pacific Design Automation Conference*. ASP-DAC (Chiba/Tokyo, Japan). 2015, pp. 214–219 (cit. on pp. 38, 50).

[RSR16a] T. Reimann, C. Sze, and R. Reis. “Cell selection for high-performance designs in an industrial design flow”. In: *Proceedings of the 2016 International Symposium on Physical Design*. ISPD (Santa Rosa, California, USA). 2016, pp. 65–72 (cit. on pp. 4, 37, 38, 49, 50, 52, 55, 89, 101–103, 133, 134).

[RSR16b] T. Reimann, C. C. Sze, and R. Reis. “Challenges of cell selection algorithms in industrial high performance microprocessor designs”. In: *Integration* 52 (2016), pp. 347–354 (cit. on pp. 15, 46, 51).

[Sap+93] S. Sapatnekar, V. Rao, P. Vaidya, and S.-M. Kang. “An Exact Solution to the Transistor Sizing Problem for CMOS Circuits Using Convex Programming”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 12.11 (1993), pp. 1621–1634 (cit. on p. 90).

[Sch15] U. Schorr. “Algorithms for gate sizing in VLSI design”. PhD thesis. University of Bonn, 2015 (cit. on pp. 4, 89–93, 98, 99, 101, 133).

[Sha+05] S. Shah, A. Srivastava, D. Sharma, D. Sylvester, D. Blaauw, and V. Zolotov. “Discrete vt assignment and gate sizing using a self-snapping continuous formulation”. In: ICCAD (San Jose, California, USA). 2005, pp. 705–712 (cit. on p. 38).

[SK87] E. Shragowitz and S. Keel. “A global router based on a multicommodity flow model”. In: *Integration. the VLSI Journal* 5.1 (1987), pp. 3–16 (cit. on pp. 17, 20, 21, 108).

[Sku98] M. Skutella. “Approximation algorithms for the discrete time-cost tradeoff problem”. In: *Mathematics of Operations Research* 23.4 (1998), pp. 909–929 (cit. on pp. 58, 60, 61, 84).

[Sve12] O. Svensson. “Hardness of Vertex Deletion and Project Scheduling”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Ed. by A. Gupta, K. Jansen, J. Rolim, and R. Servedio. 2012, pp. 301–312 (cit. on pp. 4, 58, 60, 76, 77).

[Sze05] C. Szegedy. “Some Applications of the weighted combinatorial, Laplacian”. PhD thesis. University of Bonn, 2005 (cit. on pp. 89, 90).

[Tom71] N. Tomizawa. “On some techniques useful for solution of transportation network problems”. In: *Networks. An International Journal* 1.2 (1971), pp. 173–194 (cit. on p. 73).

[Tra15] V. Traub. “Global Routing mit Delay-Beschränkungen”. Bachelor’s thesis (in German). University of Bonn, 2015 (cit. on pp. 105, 127).

[TS02] H. Tennakoon and C. Sechen. “Gate Sizing Using Lagrangian Relaxation Combined with a Fast Gradient-Based, Pre-Processing Step”. In: ICCAD (San Jose, California, USA). 2002, pp. 395–402 (cit. on pp. 90, 99).

[TS08] H. Tennakoon and C. Sechen. “Nonconvex Gate Delay Modeling and Delay Optimization”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.9 (2008), pp. 1583–1594 (cit. on p. 94).

[TZ11] J. Tu and W. Zhou. “A primal-dual approximation algorithm for the vertex cover P3 problem”. In: *Theoretical Computer Science* 412.50 (2011), pp. 7044–7048 (cit. on p. 76).

[Van90] L. Van Ginneken. “Buffer Placement in Distributed RC-Tree Networks for Minimal Elmore Delay”. In: *Proceedings of the 1990 International Symposium on Circuits and Systems.* ISCAS (New Orleans, Louisiana, USA). 1990, pp. 865–868 (cit. on pp. 35, 106, 107).

[Vyg01] J. Vygen. “Theory of VLSI Layout”. Habilitation thesis. University of Bonn, 2001 (cit. on p. 7).

[Vyg04] J. Vygen. “Near-Optimum Global Routing with Coupling, Delay Bounds, and Power Consumption”. In: *Proceedings of the 10th International Conference on Integer Programming and Combinatorial Optimization.* IPCO (New York, NY, USA). 2004, pp. 308–324 (cit. on p. 20).

[WA04] Z. L. Weiping Shi and C. J. Alpert. “Complexity analysis and speedup techniques for optimal buffer insertion with minimum cost”. In: *Proceedings of the 9th Asia and South Pacific Design Automation Conference.* ASP-DAC (Yokohama, Japan). 2004, pp. 609–614 (cit. on p. 21).

[WDL11] T. Wu, A. Davoodi, and J. T. Linderoth. “GRIP: Global Routing via Integer Programming”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30.1 (2011), pp. 72–84 (cit. on p. 20).

[WDZ07] J. Wang, D. Das, and H. Zhou. “Gate Sizing by Lagrangian Relaxation Revisited”. In: ICCAD (San Jose, California, USA). 2007, pp. 111–118 (cit. on pp. 90, 99).

[Wei+13] Y. Wei, Z. Li, C. Sze, S. Hu, C. Alpert, and S. S.S. “CATALYST: planning layer directives for effective design closure”. In: *Proceedings of the 2013 Conference on Design, Automation and Test in Europe.* DATE (Grenoble, France). 2013, pp. 1873–1878 (cit. on pp. 108, 129).

[Wei+14] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. Alpert, L. Reddy, A. Huber, G. Téllez, D. Keller, and S. a. Sapatnekar. “Techniques for scalable and effective routability evaluation”. In: *ACM Transactions on Design Automation of Electronic Systems* 19.2 (2014), Art. No. 17 (cit. on p. 130).

[Wit11] S. Wittke. “Diskrete time-cost tradeoff Probleme”. Bachelor’s thesis. University of Bonn, 2011 (cit. on p. 37).

[YYC09] Yue Xu, Yanheng Zhang, and Chris Chu. “FastRoute 4.0: Global router with efficient via minimization”. In: *2009 Asia and South Pacific Design Automation Conference.* 2009, pp. 576–581 (cit. on p. 20).

Notation

D	The timing graph, a directed graph representing the signal propagation in a chip.
WS	The worst slack, measures the maximum timing violation on a chip. See Section 2.6.
TNS, TTNS	The (true) total negative slack. See Section 2.6.
P_{inp}	Set of input ports in the timing graph. See Section 2.1.
P_{out}	Set of output ports in the timing graph. See Section 2.1.
$\text{at}(p)$	Arrival time of an input pin $p \in P_{\text{inp}}$. See Section 2.1.
$\text{rat}(p)$	Required arrival time of an output pin $p \in P_{\text{out}}$. See Section 2.1.
$\mathcal{N}$	Set of nets. See Section 3.2.
T	The main cycle time of a chip. Usually used as an upper bound on the maximum allowable signal path delay.
$\mathcal{P}$	The set of paths in a graph.
B	A power budget.
d	An upper bound on the depth of a graph. For hypergraphs the maximum size of a hyperedge.
$\mathcal{R}$	Set of resources in the min-max resource sharing problem. See Section 3.3.
$\mathcal{C}$	Set of customers in the min-max resource sharing problem. See Section 3.3.
$\mathcal{I}$	The chip image. Defined as $\mathcal{I} = \Box \times \{0, \dots, Z\}$, where $\Box$ is the chip area and Z the amount of layers. See Section 3.2.
$\mathcal{T}$	The set of wire types. See Section 3.2.
$\delta^-(v)$	Set of entering edges of node v in a directed graph [KV11, Chapter 2.1].
$\delta^+(v)$	Set of leaving edges of node v in a directed graph [KV11, Chapter 2.1].
$\delta(v)$	Set of incident edges of node v in an undirected graph [KV11, Chapter 2.1].
$\text{E}(G)$	Edges of graph G [KV11, Chapter 2.1].
$\text{V}(G)$	Vertices of graph G [KV11, Chapter 2.1].