Combinatorial Problems in Chip Design

Bernhard Korte Jens Vygen

Research Institute for Discrete Mathematics, University of Bonn

Lennéstr. 2, 53113 Bonn, Germany

June 5, 2008

Abstract

The design of very large scale integrated (VLSI) chips is an exciting area of applying mathematics, posing constantly new challenges.

We present some important and challenging open problems in various areas of chip design. Although the problems are motivated by chip design, they are formulated mathematically; understanding and solving them does not require any knowledge of chip design. We give some partial results and argue why a full resolution of one of the problems could result in an advance of the state of the art in algorithms for chip design.

Introduction

This paper is dedicated to Laci Lovász on the occasion of his 60th birthday. We have learned a lot from Laci about combinatorial problems and how to solve them. However, the paper presents mainly open problems and challenges the reader to solve them.

Chip design is one of the most interesting and important application areas of mathematics, in particular discrete mathematics. Many different mathematical techniques have been applied in chip design, and a lot of practical design problems have led to new interesting theoretical results. Still there are many challenging open problems, some of which known for decades, others raised only recently because of technological advances.

For many years we have been designing algorithms for chip design that are based on mathematics as much as possible. The resulting BonnTools (Korte,

Rautenbach and Vygen [2007]) are widely used in industry, but also competing industrial tools include more and more advanced algorithms. Nevertheless we are convinced that there is still a lot of room for improvement. We try to give several examples in this paper.

We also hope to stimulate more theoretical research that is relevant for practical problems. All examples in this paper are defined rigorously as abstract mathematical problems, which can be understood without any knowledge in chip design. Nevertheless we also mention the background briefly and argue why these problems are important.

Almost all problems arising in chip design are *NP*-hard. Sometimes it is not even clear whether exponential-time algorithms, or polynomial-time approximation algorithms exist. Nevertheless one can prove interesting positive results, for example by considering important special cases or by decomposing a problem into well-solved subproblems.

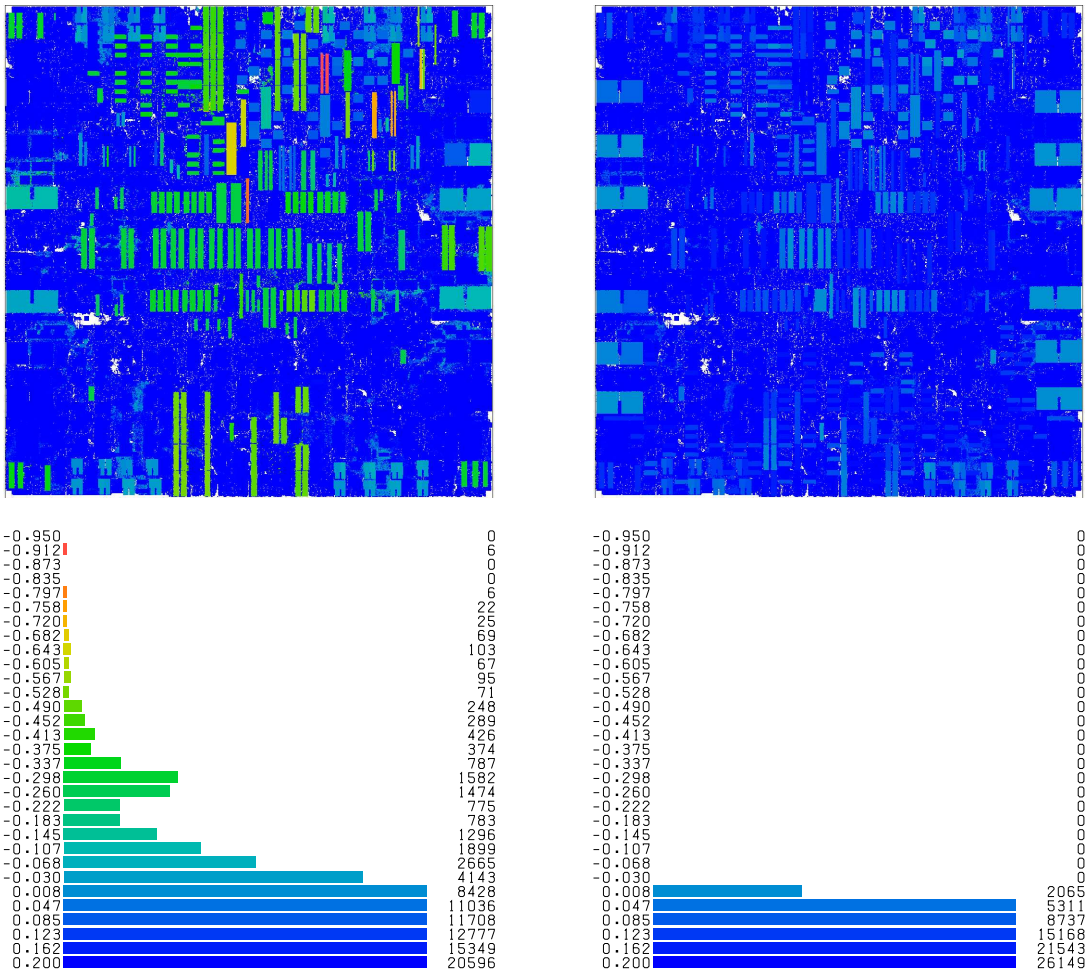
The design of a chip consists of many steps, which are of course not independent of each other. The main steps are logic synthesis, floorplanning, placement, timing optimization, clock tree design, and routing (roughly in this order). Each of these steps is decomposed further into sub-tasks. For example, routing is split into global and detailed routing; timing optimization comprises at least fanout tree design and gate sizing, etc. Most of these tasks will be explained briefly in later sections. For a detailed decscription we refer to Korte, Rautenbach and Vygen [2007].

Let us mention one of the few examples where an important problem can be solved optimally in polynomial time, and fast enough to be used in practice: clock skew scheduling. While traditionally designers were aiming for zero skew clock trees, i.e. simultaneously switching storage elements, this is far from optimal. By optimizing the arrival times one can obtain smaller cycle times and thus higher frequencies. Modelling the timing graph, where arcs correspond to signal propagations, as a digraph G with delays $d : E(G) \to \mathbb{R}$, and given a subset $F \subseteq E(G)$ and a threshold $\Theta \in \mathbb{R}$, the task is to find arrival times $\pi : V(G) \to \mathbb{R}$ with $\pi(x) + d(e) \leq \pi(y)$ for $e = (x, y) \in E(G) \setminus F$ such that the vector of relevant *slacks*

$$(\min\{\Theta, \pi(y) - \pi(x) - d(e)\})_{e=(x,y)\in F},$$

after sorting entries in non-decreasing order, is lexicographically maximal. The edges in F are normally those incident to storage elements; their worst slack will eventually determine what frequency can be achieved.

Figure 1: The effect of clock skew scheduling on the Apple G5 system controller (Held et al. [2003]). The left-hand side shows the timing result with zero skew, the right-hand side with optimal arrival times. This improved the speed (i.e., cycle time) by 27%. Placement (top) and slack histograms (showing the number of slacks in certain intervals) are colored in the same way: signals at red modules arrive much too late, while signals at blue modules have positive slack (i.e., arrive in time).



This problem can be solved in $O(mn+n^2 \log n)$ time (Albrecht et al. [2002]), where $n = |V(G)|$ and $m = |E(G)|$. For reasonable thresholds Θ it runs very fast in practice. See Held [2001], Vygen [2006], and Albrecht [2006] for generalizations and implementation aspects. See Figure 1 for an example.

Held et al. [2007] survey this and many other results that we achieved. The present paper complements this with a focus on unsolved problems.

There are well-known open problems that we do not list although they are very important for chip design (and many other applications). Examples are the questions what is the fastest running time of an algorithm for fundamental combinatorial optimization problems (such as the minimum cost flow problem), and of course whether $P = NP$. These problems have been studied for decades and seem to be very hard.

The rest of this paper consists of ten sections, each with a challenging open problem.

1 Floorplanning

Motivation

At an early design stage designers have a target chip area, i.e. a rectangle. They also have a list of rectangular objects (*modules* or *macros*) and their interconnection (the modules have *pins*, and sets of pins that must be connected are called *nets*). The task is to place these objects without overlaps such that an estimate of the interconnect length is minimized. This is called *floorplanning*. In *hierachical design* the number of objects is relatively small (about 20 to 200), as each object represents either a large memory array or a logic macro (which itself consists of many small objects). Another approach (*flat design*) is to deal with millions of small objects directly. Although the problem formulation is essentially the same, completely different approaches are needed due to the orders of magnitude (cf. Problem 2).

Instance

- a finite set M (of modules or macros)
- a rectangular chip area $A(\Box) = [0, x_{\max}] \times [0, y_{\max}]$
- an outline $A(m) = [0, w(m)] \times [0, h(m)] \subseteq A(\Box)$ for $m \in M$

- a finite set $\mathcal{N}$ of nets. Each net is a finite set with at least two elements; these elements are called pins
- an assignment $\gamma(p) \in M \dot{\cup} \{\square\}$ and an offset $(x(p), y(p))$ for each pin p (the pins p with $\gamma(p) = \square$ are fixed, e.g. I/O-ports). $(M, \mathcal{N}, \gamma)$ is called a *netlist*; it can be regarded as a hypergraph with a special vertex $\square$

Task

Compute

- a placement $(x, y) : M \to A(\square)$ such that $(x(m) + w(m), y(m) + h(m)) \in A(\square)$ for each $m \in M$, and for each $m, m' \in M$ at least one of the following conditions holds: $x(m) + w(m) \leq x(m')$ or $x(m') + w(m') \leq x(m)$ or $y(m) + h(m) \leq y(m')$ or $y(m') + h(m') \leq y(m)$
- such that $\sum_{N \in \mathcal{N}} (\max_{p \in N}(x(\gamma(p)) + x(p)) - \min_{p \in N}(x(\gamma(p)) + x(p)) + \max_{p \in N}(y(\gamma(p)) + y(p)) - \min_{p \in N}(y(\gamma(p)) + y(p)))$ is minimum, where $x(\square) := y(\square) := 0$ (this is called the *bounding box netlength*)

or decide that no solution exists.

Challenge

Find an $O(k!4^k p(k, |\mathcal{N}|, |P|))$-algorithm for this problem, where $k = |M|$ and p is a polynomial.

What is Known

The problem to decide whether a feasible placement exists is strongly *NP*-complete as it includes bin packing.

Suppose that we know which of the four conditions shall be satisfied for each pair of macros. Then one can formulate the problem as two separate linear programs (one for x- and one for y-coordinates), by adding variables representing the coordinates of the bounding box. These linear programs are duals of uncapacitated minimum cost flow instances and can thus be solved in $O(n \log n(m + n \log n))$ time (Orlin [1993]), where $n = |M| + |\mathcal{N}|$ and $m = |P|$. This was noted first by Cabot, Francis and Stary [1970].

Of course one can enumerate all $4^{\binom{k}{2}}$ possibilities, but this is too slow and many of these contradict each other. Murata et al. [1995] proposed a more efficient representation of the solution space by so-called *sequence pairs*: given any two permutations π, ρ of M, we require that m is to the west (south, north, east) of m' if m precedes m' in π and ρ (in π but not in ρ, in ρ but not in π, neither in π nor in ρ). They showed that for every feasible placement there is a sequence pair implying constraints that are satisfied for this placement. Hence one needs to enumerate only $(k!)^2$ combinations. The resulting running time of $O((k!)^2 n \log n(m + n \log n))$ seems to be the best known bound today.

Even with branch-and-bound techniques instances with $k \geq 15$ currently cannot be solved optimally. For larger instances local search heuristics (such as simulated annealing) based on these representations have been proposed and used in practice. There are hundreds of papers on floorplanning heuristics in the engineering literature, but there is no substantial progress in exact algorithms.

Other representations are more concise than sequence pairs but can only represent compactified placements, where no object can be moved downwards or to the left without destroying feasibility. The most efficient ones are O-trees (Guo, Cheng and Yoshimura [1999], Takahashi [2000]) and B*-trees (Chang et al. [2000]), both with $k!C_k \approx \frac{(k-1)!4^k}{\sqrt{\pi k}}$ combinations, where $C_k = \frac{(2k)!}{k!(k+1)!}$ is the k-th Catalan number. However, it is not clear how to use them for finding a placement with minimum bounding box netlength.

If we just want to find a feasible solution or decide that none exists, the B*-representation together with the binary tree enumeration algorithm of Solomon and Finkel [1980] does the job in $O((k-1)!4^k\sqrt{k})$ time. Polynomial-time approximation algorithms in special cases where feasibility is trivial will be discussed in Problem 2.

Extensions

- Macros can often be flipped horizontally and/or vertically. In rare cases they can also be rotated, but normally not as this would require an adapation of the routing structure on top of the macro.

- Some areas of the chip may be forbidden (or, equivalently, some macros are pre-placed and must not be moved).

- Some of the objects may in fact be *soft macros* (also called random logic macros, RLMs). They come with an (estimated) area but no fixed aspect

ratio. They have pins, but no pin locations. Soft macros correspond to units that are not designed yet and will be part of the chip. For each of them one needs to choose an aspect ratio and assign its pins to locations within its outline. Then they can be placed just as hard macros. The aspect ratio can be subject to lower and upper bounds or part of the objective function (approximately squarish outlines are often preferable). Ibaraki and Nakamura [2006] observed that there is a finite algorithm even in the case of soft macros by showing that this is a convex programming problem.

- Some soft macros may actually appear several times on a chip. In this case all realizations, i.e. aspect ratios and pin locations must be the same. In other words, for each group of identical soft macros one needs to choose a realization as hard macro first, and then place copies of this hard macro. The advantage is that this soft macro needs to be designed and checked only once and can be plugged in at several places on the chip.

Importance

Even an algorithm which can handle, say, twenty macros, would be useful. It could be used as part of a heuristic which decomposes larger instances and solves sub-instances optimally. Moreover, many practical instances are easy except for a specific local area where the problem may be very hard, but involves only relatively few macros.

Currently, due to the lack of good algorithms, floorplanning involves a lot of manual work.

2 Lower Bounds for Placement

Motivation

Placement is one of the most important tasks in chip design, but also (as shown in Problem 1) one of the hardest. Although studied for many decades, not much is known theoretically. In fact, even simplified formulations of the placement problem contain the notoriously hard quadratic assignment problem as special case. In practice the most successful approaches heuristically combine optimally solved subproblems; see e.g. Brenner, Struzyna and Vygen [2008]. But almost nothing is known about how far from optimal they are.

Instance

- a graph G (whose vertices are called *modules* or *cells* and whose edges are called *nets*),
- an array of feasible locations $A = \{1, \ldots, x_{\max}\} \times \{1, \ldots, y_{\max}\}$

Task

Compute

- a placement $\pi : V(G) \to A$ satisfying $\pi(v) \neq \pi(w)$ for all $v, w \in V(G)$ with $v \neq w$
- such that
  - (a) $\sum_{\{v,w\} \in E(G)} |\pi(v) - \pi(w)|$ is minimum.
  - (b) $\sum_{\{v,w\} \in E(G)} (\pi(v) - \pi(w))^2$ is minimum.

Challenge

Find an algorithm which computes the optimum value of (a) or (b) up to a constant factor, or even an $O(\log |V(G)|)$-factor, in polynomial time.

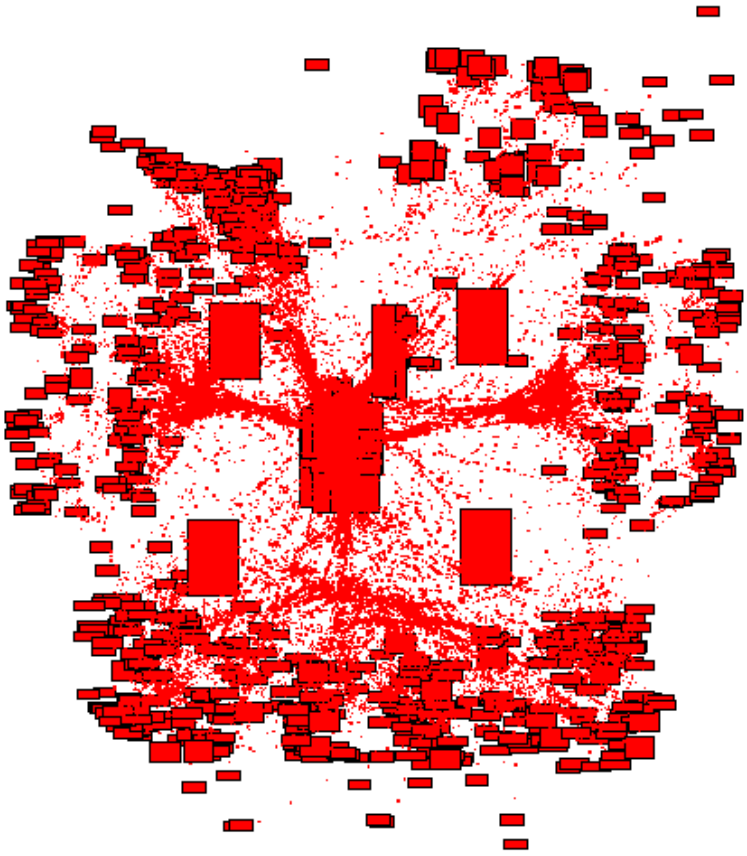
What is Known

The best known approximation algorithm for (a) has a performance guarantee of $O(\log |V(G)| \log \log |V(G)|)$ (Even et al. [2000]). In the one-dimensional case ($y_{\max} = 1$) this is the famous optimum linear arrangement problem, for which an $O(\sqrt{\log |V(G)|} \log \log |V(G)|)$-factor approximation algorithm was found by Charikar et al. [2006] and Feige and Lee [2007]. However, the optimum linear arrangement problem is not known to be *MAXSNP*-hard (but see Ambühl, Mastrolilli and Svensson [2007]). If some areas are forbidden, the problem has no constant-factor approximation algorithm (Queyranne [1986]).

Case (b) might be easier, but there is no positive result either. However, there are interesting connections to random walks and eigenvectors of the Laplacian of G (Vygen [2007], Spielman [2007]), which could potentially help.

In practice, there are only two techniques to compute lower bounds for large instances (which can be combined). The first one is to consider each net separately and assume the cells of this net to be placed as closely together as possible.

Figure 2: Quadratic placement for the same chip as in Figure 1. The cells are colored red. Most cells are tiny and lie close to the center of the chip. Thus the placement is far from being legal.



The second one relaxes the problem by ignoring the constraint that no two cells can be placed at the same position. Assuming that some pins are fixed or some cells are preplaced, which is often the case in practice, this relaxation yields a positive lower bound, but a very weak one, in particular for (a). The relaxation for (b), called quadratic placment, is a basic ingredient of many placement tools (Brenner and Vygen [2008]). See Figure 2 for an example.

Extensions

- To be really useful the algorithm should be fast enough to handle millions of cells.

- In practice the problem is more complex. Cell sizes vary (as in Problem 1), although most cells have the same height (standard cells) and are to be arranged in rows. Exceptions are macros where different problems arise (see Problem 1). Even the problem to decide whether a feasible solution exists is strongly *NP*-hard. But in practice it is very easy to find a feasible solution because a substantial portion of the available area is not used.

  Normally 99% of the cells are small (their height is uniform and their width varies by about a factor of ten), and different cell sizes typically mix in a good placement. Moreover, many cell sizes are actually determined only after placement (by a procedure called *gate sizing*, which chooses one out of several logically equivalent implementions for each cell). Hence it does not seem to be essential to work with non-uniform cell sizes.

- Pre-placed macros (as output of floorplanning; cf. Problem 1) or, equivalently, forbidden areas, also make the problem harder.

- In practice we have a hypergraph rather than a graph. Again, the vertices are cells that are to be placed and hyperedges are nets encoding the information that its pins need to be connected. More precisely, we have a netlist as in Problem 1. The pin offsets are rather small and can be ignored except at a very detailed level, but fixed pins really change the problem — just as pre-placed macros or forbidden areas do. But the number of fixed pins is typically quite small, and in some cases, where the I/O-ports are not yet placed, zero.

  It is common practice to represent hyperedges of cardinality more than two (*multi-terminal nets*) by a set of edges (two-terminal nets), e.g. a clique

(cf. Brenner and Vygen [2001]).

- Our formulation ignores important aspects like routing congestion and timing constraints. Nevertheless just minimizing netlength has proved a very useful model in practice.

Importance

Although our problem formulation is a radical simplification, it is the best approximation of the placement problem where we can hope for a positive solution even if $P \neq NP$.

It is not immediately clear how a positive answer would impact design practice. But the current status is that we have essentially no means to evaluate how close to optimum the placements that we compute are. Any progress here would be extremely interesting.

3 Legalizing a Placement

Motivation

Placement is usually divided into two subtasks: global placement and legalization. After global placement (cf. Problem 2), the size of many cells is changed and additional cells (in particular repeaters; cf. Problem 9) are added. Then the placement needs to be legalized without moving any cell too far. The (weighted) sum of quadratic movement is a good measure of legalization quality.

Instance

- an array of feasible locations $A = \{1, \ldots, x_{\max}\} \times \{1, \ldots, y_{\max}\}$
- a set C of cells to be placed, with $|C| \leq |A|$
- an initial placement $(x, y) : C \to A$

Task

Compute

- a legal placement $(\bar{x}, \bar{y}) : C \to A$, i.e. one with $\bar{x}(c) \neq \bar{x}(c')$ or $\bar{y}(c) \neq \bar{y}(c')$ for $c, c' \in C$ with $c \neq c'$,
- such that
$$\sum_{c \in C} (|\bar{x}(c) - x(c)| + |\bar{y}(c) - y(c)|)^2$$
is minimum.

Challenge

Find an $o(|C|^3)$ algorithm for this problem.

What is Known

This is a special case of the well-known linear assignment problem, which can be solved in $O(|A|^3)$ time even if the cost of assigning an element of C to an element of A is arbitrary (see, e.g., Korte and Vygen [2008]).

It seems that no better algorithms are known for the special case. However, for linear movement costs (instead of quadratic) one can reduce the problem to an instance of the minimum cost flow problem, where the underlying graph is the two-dimensional grid graph given by A (with edges connecting nodes of Manhattan distance 1, oriented in both directions). Unit costs, infinite capacities, and balance values defined by $b(a) := |\{c \in C : (x, y)(c) = a\}| - 1$ for $a \in A$ complete the instance. An integral minimum cost flow exists and can be realized by moving cells. This yields an $O(|A|^2 \log^2 |A|)$-algorithm for the variant with linear movement costs. It seems plausible that $|A|$ can be replaced by $|C|$ in this time bound as locations that are never used do not require any computation.

This minimum cost flow approach has been used a lot in practice (Vygen [1998], Brenner and Vygen [2004]). Of course, when used as a heuristic for minimizing quadratic movement costs, the flow cannot be realized arbitrarily as it is better to move many cells a little rather than one cell far.

Extensions

- Typically some locations are pre-occupied by large macros and must not be used.

- Some cells are more important (because of timing constraints); so one has weights $w : C \to \mathbb{R}_+$ and should minimize the weighted sum of squared movements.

- Cells have different sizes, although in a typical legalization instance they all have the same height. But even with varying widths the problem to decide whether any feasible solution exists is strongly *NP*-hard. In practice this is not an issue, as rarely more than 90% of the chip area is used, and as the widths are typically small integer multiples of some minimum width.

  Nevertheless, with varying cell widths the problem becomes harder and can probably no longer be formulated as an assignment problem. Still Brenner and Vygen [2004] use the above minimum cost flow approach, construct an appropriate (more sophisticated) minimum cost flow instance, and solve knapsack problems to realize the flow. This paper also proposes an integer programming formulation whose LP relaxation provides reasonable lower bounds in practice.

Importance

Placement legalization is a key task in chip design and an increasing challenge with progressing technology. Poor legalization makes it hard to find a solution satisfying all timing constraints. A solution to the above problem formulation, although not capturing different cell widths, would be a significant step towards a practical and provably near-optimal solution.

4 Steiner Trees Minimizing Elmore Delay

Motivation

On a chip there are millions of signals that are generated at a certain place (the source) and must be transmitted to several destinations (sinks). Each net contains one source and at least one sink. The elements of a net need to be connected by a network of copper wires, which can be approximated well by a *rectilinear Steiner tree* (ignoring the third dimension). However, a shortest Steiner tree is often not best possible to meet tight constraints on the delay of the signal from the source to each sink.

To estimate the delay from the source to a sink we use the popular *Elmore delay* model (Elmore [1948]) which is relatively simple and accurate (see below). To account for different timing constraints we have an extra additive term for each sink.

Instance

- a source $s \in \mathbb{R}^2$
- a nonempty finite set of sinks $T \subset \mathbb{R}^2$
- a delay adder $a_t \in \mathbb{R}$ for $t \in T$
- a source resistance $r_s > 0$
- a sink capacitance $c_t > 0$ for each $t \in T$
- the capacitance $c > 0$ and resistance $r > 0$ of a wire per unit length.

Task

Compute

- a rectilinear Steiner tree Y for $\{s\} \cup T$, oriented as an arborescence rooted at s whose vertices are elements of $\mathbb{R}^2$
- such that
$$\max_{t \in T} (a_t + \mathrm{ED}(s,t))$$
is minimum,

where for $t \in T$ the Elmore delay from s to t is

$$\mathrm{ED}(s,t) := r_s C_s + \sum_{e=(v,w) \in E(Y_{[s,t]})} r||v-w||_1 \left(\frac{c}{2}||v-w||_1 + C_w \right),$$

$Y_{[s,v]}$ denotes the path from s to $v \in V(Y)$ in Y, and

$$C_v := \sum_{e=(x,y): v \in V(Y_{[s,x]})} c||x-y||_1 + \sum_{t \in T: v \in V(Y_{[s,t]})} c_t.$$

Challenge

Find a finite algorithm for this problem.

What is Known

The problem is known to be *NP*-hard (Boese et al. [1994]). For $|T| = 1$ it is trivial; a shortest path does the job. For $|T| \leq 3$ the problem can be solved in constant time (Kadodi [1999], Peyer [2000]).

But even for $|T| = 2$ there are instances in which the only optimal solution is not part of the *Hanan grid*, the grid of horizontal and vertical lines induced by the source and the sinks (Boese et al. [1994]). This is not true for the variant in which $\sum_{t \in T} a_t \mathrm{ED}(s,t)$ is minimized for weights $a_t > 0$ ($t \in T$); hence this variant can be solved in exponential time. See also Cong et al. [1996] and Kahng and Robins [1995] for other variants.

Extensions

- Actually we can choose the width of any wire (within a given range) and have several (currently up to 10) routing planes with different characteristics available. Routing planes normally have alternating preference directions (horizontal/vertical).

  Wider wires have smaller resistance but larger capacitance per unit length (and they consume more space). Wires on upper planes are often better, but the pins (source and sinks) are usually on the lower planes and the resistance of vias (metal contacts connecting adjacent planes) are not negligible.

  In practice, most of the nets are realized by a set of wires almost all of which (except short ones for pin access) have the same width and lie on two adjacent planes (one horizontal and one vertical) with similar electrical properties. Therefore the above planar simplification is meaningful.

- Our formulation ignores routing blockages. Sometimes there are significant blockages (in particular on top of macros) that must be avoided. But in most cases there are only small blockages (mostly due to the power supply) which can be neglected here.

- Pins are not single points but sets of metal shapes. Consequently we have a *group Steiner tree problem*. But since each pin spans only a small area, this is less important.
- The Elmore delay is the most accurate model with such a simple description. More accurate models depend on more parameters (in particular the so-called *slew rate*: the average rate of voltage changes); we do not go into details here.
- In some cases, in particular for clock networks, signals must not arrive too early either. In this case a reasonable objective function combines *latency* (maximum delay) and *skew* (maximum difference of delays), such as

  $$(1+\alpha)\max_{t\in T}(a_t + \mathrm{ED}(s,t)) - \min_{t\in T}(a_t + \mathrm{ED}(s,t))$$

  for some $\alpha > 0$. This variant is also unsolved.

Importance

The task to transmit a signal as fast as possible to a given set of destinations is a fundamental problem in chip design. Although shortest rectilinear Steiner trees are often good enough, one would of course like to solve such basic sub-tasks optimally.

5 Resource Sharing and Multiflows

Motivation

Routing is also a major task in chip design. Millions of nets have to be connected by wires, which must satisfy many rules. Routing is typically split into global routing and detailed routing. In global routing we determine a corridor for each net which restricts the solution space explored in detailed routing.

The simplest formulation of the global routing problem asks for packing Steiner trees for given terminal sets in a three-dimensional grid graph with edge capacities. In the case of two-terminal nets this is an integer multicommodity flow problem. But besides routing space (reflected by edge capacities) there are other resources that must be shared between nets, such as time (for signal delay), and space for buffers (needed for repowering signals; cf. Problem 9). A general

resource sharing problem models all this and does not seem to be much harder than a standard multicommodity flow formulation. The common approach is to solve the fractional relaxation first and use randomized rounding for obtaining an integral solution.

Instance

- a finite set $\mathcal{R}$ of resources
- a finite set $\mathcal{C}$ of customers
- a number $u_r > 0$ specifying how many units of resource $r \in \mathcal{R}$ are available
- for each customer $c \in \mathcal{C}$ an implicitly given convex set $\mathcal{A}_c \subseteq \prod_{r \in \mathcal{R}} [0, u_r]$ of feasible resource allocation vectors (satisfying customer c). We assume an oracle for computing a function $f_c : \mathbb{R}_+^{\mathcal{R}} \to \mathcal{A}_c$ which approximately minimizes linear functions over $\mathcal{A}_c$: for a customer $c \in \mathcal{C}$ and a price vector $\omega \in \mathbb{R}_+^{\mathcal{R}}$ we require that $\omega^\top f_c(\omega) \leq (1 + \epsilon_0) \operatorname{opt}_c(\omega)$, where $\operatorname{opt}_c(\omega) := \min_{a \in \mathcal{A}_c} \omega^\top a$ and $\epsilon_0 \geq 0$ is a constant

Task

Compute

- a feasible resource allocation vector for each customer
- such that the maximum relative utilization (or congestion) of all resources is minimized:

$$\min_{(a_c \in \mathcal{A}_c)_{c \in \mathcal{C}}} \max_{r \in \mathcal{R}} \frac{\sum_{c \in \mathcal{C}} (a_c)_r}{u_r}$$

Challenge

Find an algorithm which computes an $(1 + \epsilon_0 + \epsilon)$-approximate solution in $O^*(\frac{1}{\epsilon}|\mathcal{C}|\gamma)$ time, for any $\epsilon > 0$, where γ denotes the time for an oracle call and the O^*-notation suppresses logarithmic terms.

What is Known

If ϵ_0 can be chosen arbitrarily small, this is a special case of the equivalence of weak optimization and weak separation (Grötschel, Lovász and Schrijver [1988]).

Combinatorial approximation algorithms (some of which assume $\epsilon_0 = 0$) were proposed by Plotkin, Shmoys and Tardos [1995], Grigoriadis and Khachiyan [1996], Garg and Könemann [2007], and Jansen and Zhang [2008], based on earlier similar algorithms for the special case of multicommodity flows. All algorithms have a dependence on ϵ which is quadratic or worse. Müller and Vygen [2008] (see Albrecht [2001] and Vygen [2004] for special cases) found an $O(\frac{1}{\epsilon^2}|\mathcal{C}|\gamma \ln^2 |\mathcal{R}|)$-algorithm. Bienstock and Iyengar [2006] showed how to reduce the term from $\frac{1}{\epsilon^2}$ to $\frac{1}{\epsilon}$, but by increasing the dependence on other parameters. It is not clear whether these techniques can be combined.

Extensions

- All known algorithms also compute an approximate dual solution. This can be useful in particular if the routing instance turns out to be infeasible.

- In practice one is of course interested in an integral solution. The corresponding problem includes the edge-disjoint paths problem and is of course *NP*-hard. Randomized rounding works quite well unless $\frac{a_r}{u_r}$ is large for some $r \in \mathcal{R}$, $c \in \mathcal{C}$ and $a \in \mathcal{A}_c$. However, this two-stage approach (first solve the fractional relaxation approximately, then apply randomized rounding) may not be optimal.

Importance

Sharing resources optimally is a fundamental problem with numerous applications. In chip design it models global routing very well and can also handle various constraints and objectives that make global routing today much different from just packing Steiner trees subject to edge capacities. For example, Vygen [2004] and Müller [2006] showed how to model timing constraints and even manufacturing yield by resource constraints. The full value of this approach is yet to be explored.

6 Shortest Paths in Grids

Motivation

The main sub-task in detailed routing is to connect two sets of metal shapes by a set of wires of minimum total length. Each of these sets of shapes can be a pin or a previously routed set of wires connecting some pins. The problem can be modelled as a shortest path problem in a three-dimensional grid graph and solved by Dijkstra's algorithm (Dijkstra [1959]).

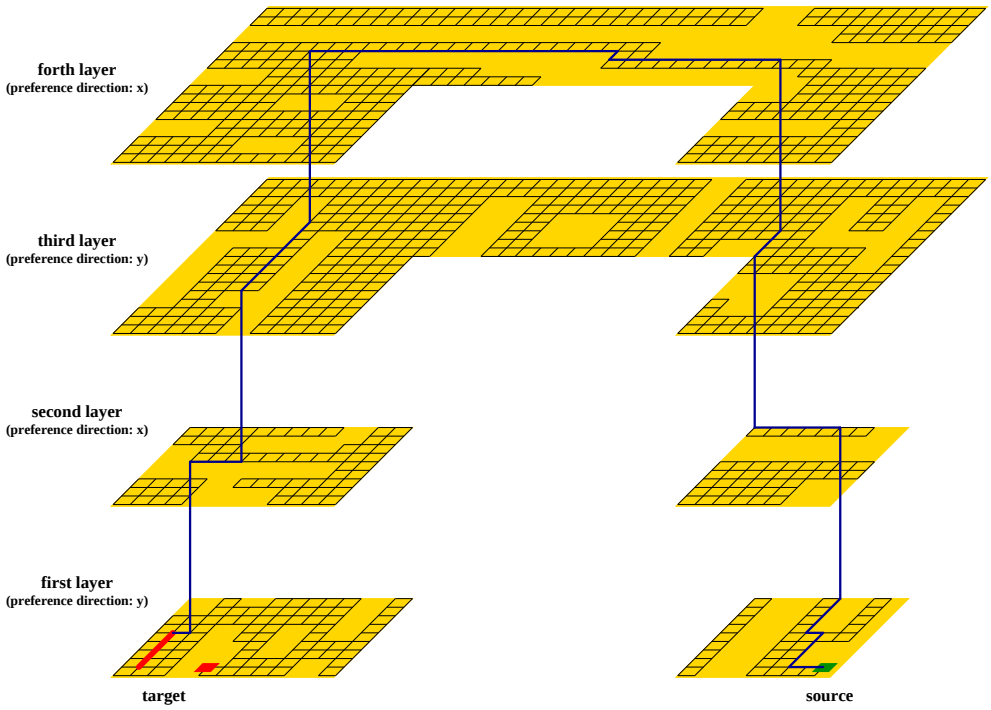
The main problem is that we need to find more than 20 million paths, and the grid graph can have more than 100 billion vertices. Although the considered subgraphs for each net are restricted as a result of global routing, this remains an enormous computational challenge.

Routing layers normally have alternating preference directions (horizontal/vertical). Edges orthogonal to the preference directions, edges corresponding to vias (connecting adjacent planes), and edges on lower metal planes have higher cost to account for waste of space, effects on manufacturing yield, and electrical properties. See Figure 3 for an example.

Instance

- positive integers $x_{\max}$, $y_{\max}$, and $z_{\max}$
- for each odd $z \in \{1, \ldots, z_{\max}\}$ and each $y \in \{1, \ldots, y_{\max}\}$ an integer $i \geq 0$ and a list of coordinates $1 \leq x_1 < x_2 < \cdots < x_{2i-1} < x_{2i} \leq x_{\max}$ defining i intervals $\{(x_1, y, z), (x_1 + 1, y, z), \ldots, (x_2, y, z)\}, \ldots, \{(x_{2i-1}, y, z), (x_{2i-1} + 1, y, z), \ldots, (x_{2i}, y, z)\}$
- for each even $z \in \{1, \ldots, z_{\max}\}$ and each $x \in \{1, \ldots, x_{\max}\}$ an integer $i \geq 0$ and a list of coordinates $1 \leq y_1 < y_2 < \cdots < y_{2i-1} < y_{2i} \leq y_{\max}$ defining i intervals $\{(x, y_1, z), (x, y_1+1, z), \ldots, (x, y_2, z)\}, \ldots, \{(x, y_{2i-1}, z), (x, y_{2i-1}+ 1, z), \ldots, (x, y_{2i}, z)\}$
- G is defined as follows: $V(G)$ is the union of all these intervals, and $E(G) := \{\{(x, y, z), (x', y', z')\} : (x, y, z), (x', y', z') \in V(G), |x - x'| + |y - y'| + |z - z'| = 1\}$
- numbers $c_{z,i} \in \mathbb{N}$ for $z = \{1, \ldots, z_{\max}\}$ and $i \in \{1, 2, 3\}$, defining edge weights $c : E(G) \to \mathbb{N}$ by $c(e) := c_{z,i}$ for $e = \{(x, y, z), (x', y', z')\} \in E(G)$ with $z \leq z'$, where $i = 1$ if $x \neq x'$, $i = 2$ if $y \neq y'$, and $i = 3$ if $z + 1 = z'$

Figure 3: This example shows a shortest path between source (green) and target (red) in a subgrid with four layers determined by global routing (yellow). The cost of an edge running in and orthogonal to the preference direction is 1 and 4, respectively, and the cost of a via is 13. With these costs the blue path has length 153, which is shortest possible. Note that the graph can be represented by relatively few intervals.



- vertex sets $S, T \subseteq V(G)$

Task

Compute

- a shortest S-T-path in (G, c), i.e. a path P in G from a vertex $s \in S$ to a vertex $t \in T$ such that $\sum_{e \in E(P)} c(e)$ is minimum.

Challenge

Find an algorithm with running time $O(K \log K)$, where K is the number of intervals I for which there is an S-I-path of length L, where L is the length of a shortest S-T-path.

What is Known

If all intervals are singletons, Dijkstra's algorithm with a heap implementation does the job. For the general case there is an $O(LK \log K)$-algorithm by Peyer, Rautenbach and Vygen [2006].

Extensions

- A well-known idea for speeding up Dijkstra's algorithm in practice is to replace the cost of an edge (v, w), originally $c(\{v, w\})$, by $c'(v, w) := c(\{v, w\}) - \pi(v) + \pi(w)$, where $\pi : V(G) \to \mathbb{R}$ satisfies $\pi(t) = 0$ for $t \in T$ and $\pi(v) \leq c(\{v, w\}) + \pi(w)$ for $\{v, w\} \in E(G)$ (and hence $\pi(v)$ is a lower bound on the length of a path from v to T) (Pohl [1971], Rubin [1974]). The better this lower bound is, the shorter is the distance from S to T with respect to c', and the fewer vertices are labelled by Dijkstra's algorithm.

  Of course, this technique (often called *goal-oriented search*) changes the instance: it makes the graph directed and changes some of the costs. By splitting intervals when necessary the edges within an interval can be assumed to have constant cost in each direction. The result of Peyer, Rautenbach and Vygen [2006] extends to this more general situation.

- G may actually not be an induced subgraph of the complete three-dimensional grid graph. Some edges, in particular some of those representing vias, are often missing (e.g. in order to model via distance rules).
- Many nets have more than two terminals; we need a Steiner tree connecting their pins. It may be possible to extend the algorithm to k-terminal nets for any fixed constant k.
- Not all rules which must be followed in routing can be encoded by the incomplete grid graph as above. An examples of such a rule is that there must be a certain minimum amount of metal (i.e., horizontal or vertical wiring) between two adjacent vias. As an approximate formulation in graph-theoretic terms, each connected component of the subgraph of the path that is induced by one routing layer must have minimum size.
- The rules for pin access are quite involved and cannot be discussed here, but the good news is that they require only local techniques and can be ignored when searching for global (long) connections.

Importance

Many hours of computing time are spent with routing on a chip in leading-edge technology. In particular long-distance paths in regions with many obstacles are difficult to find. Computing such a path can take several seconds. Speeding this up would have a great benefit.

7 Sink Clustering

Motivation

Some signals on a chip are distributed to thousands, and sometimes even millions of receivers (sinks). An example is given by clock networks distributing a clock signal to all storage elements (registers, flip-flops, latches) in a clock domain. Such networks consume a lot of resources, and in particular a lot of electric power.

Often the network has two levels. A set of special drivers receives the signal from a top-level network, and each of these drivers distributes it further to a set of sinks. The placement of drivers and assignment of sinks to drivers is a kind of facility location problem that we call sink clustering (cf. Figure 4).

The electrical capacitance of a net depends on the total length of the wires and the input capacitance of its sinks. There is a limit on the electrical capacitance that a driver can cope with. The overall power consumption also depends on the total capacitance and on the number of drivers. This yields the following problem formulation.

Instance

- a finite set $D \subseteq \mathbb{R}^2$ of sinks
- demands (sink capacitances) $d : D \to \mathbb{R}_+$
- a facility opening cost $f \in \mathbb{R}_+$
- a capacity $u \in \mathbb{R}_+$

Task

Compute

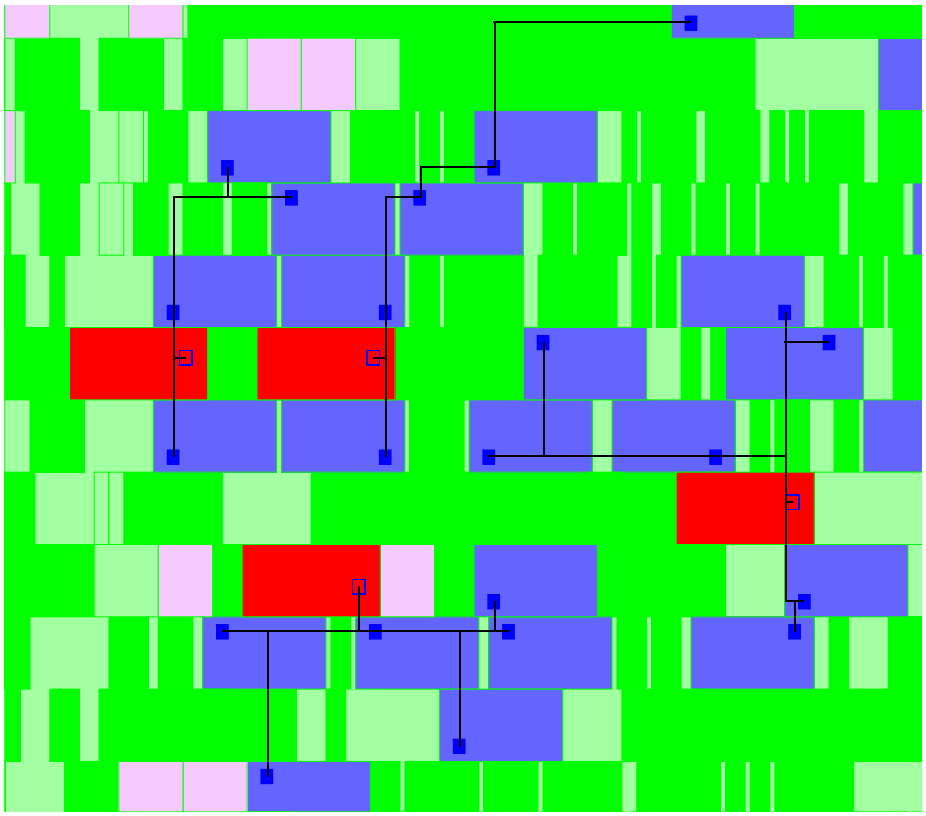
- a partition $D = D_1 \dot\cup \cdots \dot\cup D_k$,
- a rectilinear Steiner tree T_i for each D_i $(i = 1, \ldots, k)$
- such that $c(T_i) + \sum_{s \in D_i} d(s) \leq u$ for $i = 1, \ldots, k$
- and
$$\sum_{i=1}^{k} c(T_i) + kf$$
is minimum.

Here $c(T_i)$ denotes the length of T_i.

Challenge

Find a polynomial-time 2-approximation algorithm.

Figure 4: Example for sink clustering (small part of a chip). The blue squares are the sinks (here: input pins of storage elements which need to receive a periodic clock signal). The red objects are the drivers that are to be placed. We may assume that each driver can be placed on a shortest Steiner tree connecting the sinks that it drives (or very close). The green objects are irrelevant here.



What is Known

Maßberg and Vygen [2005] found a 4-approximation algorithm which runs in $O(|D| \log |D|)$ time. Unless $P = NP$ there is no approximation algorithm with performance guarantee better than 2, which can easily be seen by a reduction from the *NP*-hard rectilinear Steiner tree problem (Maßberg and Vygen [2005]): it is *NP*-complete to decide whether there is a feasible solution with $k = 1$.

Extensions

- The above description uses just one type of drivers. In practice one often has several types, cheaper ones that can drive less and stronger ones that can drive more capacitance. This corresponds to several pairs (u_i, f_i), $i = 1, \ldots, t$. Then $u := \max_{i=1}^{t} u_i$, and the objective function becomes

$$\sum_{i=1}^{k} \left(c(T_i) + \min_{i:u_i \geq c(T_i)+d(D_i)} f_i \right)$$

- Our description assumes that drivers can be placed anywhere. This is a good approximation except for macros which block a larger part of a chip. A proper formulation would take a list of rectangular blockages into account, although we do not see the need for this in practice.

- The drivers must be connected by a top-level network, often a tree. Therefore one could think of including the length of a Steiner tree connecting the drivers in the objective function.

- The problem can also be considered in general metric spaces. Here the best known approximation algorithm has a performance guarantee of 4.099 (Maßberg and Vygen [2005]).

Importance

On many chips the clock networks consume a substantial portion of the power. Reducing power consumption becomes more and more important. The objective function of the sink clustering problem models power consumption (of the lowest stage, which typically makes up more than 80% of the clock tree's overall power consumption) quite accurately.

8 Octagon Representation

Motivation

An *octilinear octagon* is a nonempty set of the form $\{(x, y) \in \mathbb{R}^2 : ix + jy \geq c_{ij}$ for $(i, j) \in \{-1, 0, 1\}^2\}$ for some constants c_{ij}. In the special case when $c_{ij} = -\infty$ for $|i| + |j| \neq 1$ they are called *rectilinear rectangles*.

Octilinear octagons arise from ℓ_1-discs by subtracting blockages. More precisely, consider all sets that can be generated from points by the following operations: (a) replace $X \subseteq \mathbb{R}^2$ by $\{y \in \mathbb{R}^2 : ||y - x||_1 \leq d$ for some $x \in X\}$ for some $d > 0$; (b) replace $X \subseteq \mathbb{R}^2$ by the closure of $X \setminus R$, where R is a rectilinear rectangle. All such sets can be written as union of octilinear octagons. They arise for example in clock tree design (Held et al. [2003]).

Instance

- a set S of n octilinear octagons

Task

Compute

- a set S' of octilinear octagons such that the union of S' equals the union of S and the interiors of any two elements of S' are disjoint.

Challenge

Find an $O(n \log n + k)$-algorithm, where k is the cardinality of a smallest set S' which is a feasible output.

What is Known

This problem can be reduced to the so-called contour problem, asking for the contour of the union of S: given the contour with k corners one can generate a solution S' with $|S'| \leq k$. As every corner of the contour must be a corner of an element of S', this is optimum up to a factor of 8.

There are simple examples for which $k = \Theta(n^2)$, even when we restrict to rectilinear rectangles. But such instances will probably not arise in practice.

For rectilinear rectangles (i.e. $c_{ij} = -\infty$ for $|i| + |j| = 2$) the contour problem can be solved by a sweepline algorithm (Güting [1984a]) or by divide-and-conquer (Güting [1984b]).

Extensions

- Ideally the algorithm would compute a set S' of minimum possible cardinality. Then there is no longer a simple reduction to the contour problem.
- The most interesting instances are those that arise from a set S of n octagons for which the interiors of any two elements are disjoint by an operation (a) followed by a sequence of n' operations (b). Assuming that S and the sequence of these operations is given explicitly, Then an $O((n + n') \log(n + n'))$-time algorithm could be possible.

Importance

This is the task of a basic subroutine in bottom-up algorithms for clock tree design and could also be used by fanout tree algorithms (cf. Section 9). Currently we are using an algorithm with a quadratic worst-case running time and apply heuristics to bound the number of octagons. This is not a satisfactory solution.

9 Short and Fast Fanout Trees

Motivation

If a signal is sent over a large distance or distributed to many sinks (i.e., has a large fanout), it has to be re-powered. This is done by buffers or pairs of inverters (implementing the identity function). Buffers and inverters are also called *repeaters*. Optimal buffering makes the delay per unit distance grow almost linearly. However, the delay along a path also depends on the number of bifurcations (more precisely, on the electrical capacitance driven by the repeaters).

Designing trees with repeaters in an optimal way, such that signals arrive in time but not too many resources are consumed, is an important task. Traditional approaches started with a shortest Steiner tree and inserted buffers and inverters, but this is far from optimal. Better trees are needed to meet tight timing constraints.

Instance

- a source $r \in \mathbb{R}^2$
- a finite set $S \subset \mathbb{R}^2$ of sinks
- required arrival times a_s ($s \in S$), assuming that the signals starts at r at time 0
- a delay $d > 0$ per unit distance
- a bifurcation cost $c > 0$

Task

Compute

- an arborescence A with $\{r\} \dot\cup S \subseteq V(A) \subset \mathbb{R}^2$ and $|\delta^+(r)| = 1$ and $|\delta^+(v)| = 2$ for $v \in V(A) \setminus (\{r\} \cup S)$
- such that the worst slack

$$\min_{s \in S} \left(a_s \; - \sum_{e=(v,w) \in E(A_{[r,s]})} d||v-w||_1 \; - \; c(|E(A_{[r,s]})| - 1) \right)$$

  is as large as possible up to an additive constant of c,
- and among all such solutions $\sum_{e=(v,w) \in E(A)} ||v-w||_1$ is minimum.

Here $A_{[r,s]}$ denotes the unique r-s-path in A.

Challenge

Find a polynomial-time $\frac{3}{2}$-approximation algorithm.

What is Known

Just maximizing the worst slack is trivial by *Huffman coding* as long as all numbers $a_s - d||r-s||_1$ ($s \in S$) are integer multiples of c. Here all $|S| - 1$ Steiner points can be placed at the position of r. Defining the criticality of a sink s by $a_s - d||r-s||_1$, the arborescence is constructed in a bottom-up fashion

by iteratively replacing the two sinks with minimum criticality by a new sink whose criticality is by c less than the smaller of the two.

It is easy to see that this yields optimum worst slack (up to a rounding error of at most c), but it will generally lead to intolerably large length. Bartoschek et al. [2006] showed how the length can be improved by successively inserting the sinks at an optimal place, in an order of nondecreasing criticality. However, this guarantees neither optimum worst slack nor short length. If we always insert a sink that is closest to the current tree in a shortest possible way, we get a well-known $\frac{3}{2}$-approximation algorithm (sometimes called Prim heuristic) for the rectilinear Steiner tree problem.

The question is whether the two results can be combined somehow. However, this will not be straightforward, as Alon and Azar [1993] gave an example showing that for the online rectilinear Steiner tree problem the best appproximation ratio we can hope for is $\Theta(\log n / \log \log n)$, where n is the number of terminals.

Extensions

- Ideally the algorithm should have at most quadratic runnning time.
- Some areas of the chip are blocked and must not be used. All vertices and edges must be embedded in the remaining area. Other areas are blocked by macros and cannot be used for inverters and buffers but can be used for wiring. Maximal connected subtrees of the arborescence that run across such blockages must not exceed a certain length (corresponding to the capacitance that a repeater can drive).
- There are several routing planes available. Upper planes are faster but also more expensive in terms of resource consumption. Moreover, the planes can be used either only for horizontal or only for vertical wires. The extended task includes the assignment of each edge to a pair of orthogonal planes.
- Of course it would be even better if one could directly find approximately optimal repeater trees rather than first constructing a topology and then inserting inverters and buffers. However, this seems to be even harder.

Importance

A state-of-the-art chip has millions of buffers and inverters, and timing constraints are very hard to meet. Buffering, in particular designing fanout trees, is a major part of timing optimization and a key problem in chip design today.

10 Logic Synthesis

Motivation

Logic synthesis is the problem of implementing a given Boolean function $f : \{0,1\}^m \to \{0,1\}^n$ by a *Boolean circuit*. A Boolean circuit can be represented by an acyclic digraph with n sources and m sinks. The other vertices are *gates*, each of which has an elementary Boolean function, such as an inverter, AND, OR, NAND, NOR, XOR, etc. A Boolean circuit computes a Boolean function in the natural way. It is essentially equivalent to a netlist. f is typically given either in a certain hardware description language (similar to a programming language), or by a Boolean circuit.

The main criteria for the quality of a solution are area (which is estimated by the number of gates) and timing (see below).

In spite of its tremendous relevance, not much is known about algorithms for logic synthesis. Quite simple heuristics are used in practice. Of course it is *coNP*-complete to decide whether a given Boolean circuit implements a constant function, but this does not mean that all problems are hopeless.

Instance

- a Boolean function $f : \{0,1\}^m \to \{0,1\}^n$ by a list of all 2^m function values (each consisting of n bits)

Task

Compute

- a Boolean circuit (using some fixed library) with minimum number of gates implementing f.

Challenge

Find a polynomial-time algorithm, or at least an approximation algorithm.

What is Known

Essentially nothing. Even in the case $n = 1$ we know no approximation algorithm. The library is probably less important, for example we could restrict to NANDs only. To the best of our knowledge the only known theoretical results consider variants of this problem (see below).

Extensions

- As a variation, f might be given by some Boolean circuit using k gates; now we ask for an algorithm whose running time is $2^{O(k)}$.
- Often we also want to bound the depth, i.e. the length of a longest path. This roughly corresponds to delay. The variation where the Boolean circuit must have depth 2 was shown to be *NP*-complete by Masek [1978] and inapproximable by Feldman [2006]. See also the work of Kabanets and Cai [2000]. The problem seems to be open even for depth 3.
- The inputs $1, \ldots, m$ may be associated with arrival times $a_1, \ldots, a_m$, and the outputs have required times $r_1, \ldots, r_n$. Then we want to maximize $\min_{i,j}(r_j - a_i - d_{ij})$, where d_{ij} denotes the length of a longest path from i to j. Rautenbach, Szegedy and Werber [2003] suggested an approximation scheme for a special case of the case $n = 1$.
- In fact we often have an incompletely specified Boolean function $f : \{0,1\}^m \to \{0,1,d\}^n$, where d stands for *don't care*. We look for a *Boolean circuit* computing some concretion of f, i.e. some $g : \{0,1\}^m \to \{0,1\}^n$ with $f_i(x) \in \{g_i(x), d\}$ for all $x \in \{0,1\}^m$ and $i = 1, \ldots, n$.

Importance

Logic synthesis is key for the quality of a netlist. The state of the art is rather poor, compared to other areas of chip design. Therefore we see much room for improvement here. While physical chip layout has improved substantially in the last two decades due to better algorithms, logic synthesis has not been

approached seriously by mathematicians. Consequently an improvement of logic synthesis by mathematical ideas is badly needed.

Conclusion

Although a lot of progress has been made, and many theoretical results and new algorithms made their way to real design tools, chip design practice is also a constant source of interesting problems, many of which are of combinatorial nature. On the one hand, new problems come up or become important due to technological advances. On the other hand, new ideas lead to new formulations in classical problem areas. We hope that we could demonstrate the variety of interesting problems and could give a fresh view on some of the most important combinatorial problems in chip design.

References

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

Albrecht, C. [2006]: Efficient incremental clock latency scheduling for large circuits. Design, Automation and Test in Europe, Proceedings, IEEE 2006, 1091–1096

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

Alon, N., and Azar, Y. [1993]: On-line Steiner trees in the Euclidean plane. Discrete and Computational Geometry 10 (1993), 113–121

Ambühl, C., Mastrolilli, M., and Svensson, O. [2007]: Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (2007), 329–337

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

Bienstock, D., and Iyengar, G. [2006]: Solving fractional packing problems in $O^*(\frac{1}{\epsilon})$ iterations. SIAM Journal on Computing 35 (2006), 825–854

Boese, K.D., Kahng, A.B., McCoy, B.A., and Robins, G. [1994]: Rectilinear Steiner trees with minimum Elmore delay. Proceedings of the 31st IEEE/ACM Design Automation Conference (1994), 381–386

Brenner, U., Struzyna, M., and Vygen, J. [2008]: BonnPlace: placement of leading-edge chips by advanced combinatorial algorithms. IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, to appear

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

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

Brenner, U., and Vygen, J. [2008]: Analytical methods in VLSI placement. In: Handbook of Algorithms for VLSI Physical Design Automation (C.J. Alpert, D.P. Mehta, S.S. Sapatnekar, Eds.), Taylor and Francis, 2008

Cabot, A.V., Francis, R.L., and Stary, A.M. [1970]: A network flow solution to a rectilinear distance facility location problem. AIIE Transactions 2 (1970), 132–141

Chang, Y.-C., Chang, Y.-W., Wu, G.-M., and Wu, S.-W. [2000]: B*-trees: a new representation for non-slicing floorplans. Proceedings of the 37th ACM/IEEE Design Automation Conference (2000), 458–463

Charikar, M., Hajiaghayi, M., Karloff, H. and Rao, S. [2006]: ℓ_2^2 spreading metrics for vertex ordering problems. Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (2006), 1018–1027

Cong, J., He, L., Koh, C.-K., and Madden, P.H. [1996]: Performance optimization of VLSI interconnect layout. Integration, the VLSI Journal 21 (1996), 1–94

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

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

Even, G., Naor, J., Rao, S., and Schieber, B. [2000]: Divide-and-conquer approximation algorithms via spreading metrics. Journal of the ACM 47 (2000), 585–616

Feige, U., and Lee, J.R. [2007]: An improved approximation ratio for the minimum linear arrangement problem. Information Processing Letters 101 (2007), 26–29

Feldman, V. [2006]: Hardness of approximate two-level logic minimization and PAC learning with membership queries. Proceedings of the 38th Annual ACM Symposium on the Theory of Computing (2006), 363–372

Garg, N., and Könemann, J. [2007]: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. SIAM Journal on Computing 37 (2007), 630–652

Grigoriadis, M.D., and Khachiyan, L.D. [1996]: Coordination complexity of parallel price-directive decomposition. Mathematics of Operations Research 21 (1996), 321–340

Grötschel, M., Lovász, L., and Schrijver, A. [1988]: Geometric Algorithms and Combinatorial Optimization. Springer, Berlin 1988

Guo, P.N., Cheng, C.-K., and Yoshimura, T. [1999]: An O-tree representation of non-slicing floorplan and its applications. Proceedings of the 36th ACM/IEEE Design Automation Conference (1999), 268–273

Güting, R.H. [1984a]: An optimal contour algorithm for iso-oriented rectangles. Journal of Algorithms 5 (1984), 303–326

Güting, R.H. [1984b]: Optimal divide-and-conquer to compute measure and contour for a set of iso-oriented rectangles. Acta Informatica 21 (1984), 271–291

Held, S. [2001]: Algorithmen für Potential-Balancierungs-Probleme und Anwendungen im VLSI-Design. Diploma thesis, University of Bonn, 2001

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

Held, S., Korte, B., Rautenbach, D., and Vygen, J. [2007]: Combinatorial optimization in VLSI design. Report No. 07974, Research Institute for Discrete Mathematics, University of Bonn, 2007

Ibaraki, T., and Nakamura, K. [2006]: Packing problems with soft rectangles. In: Hybrid Metaheuristics; Proceedings of the 3rd International Workshop on Hybrid Metaheuristics (HM 2006); LNCS 4030 (F. Almeida et al., eds.), Springer, Berlin 2006, pp. 13–27

Jansen, K., and Zhang, H. [2008]: Approximation algorithms for general packing problems and their application to the multicast congestion problem. Mathematical Programming 114 (2008), 183–206

Kabanets, V., and Cai, J.-Y. [2000]: Circuit minimization problem. Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing (2000), 73–79

Kadodi, T. [1999]: Steiner routing based on Elmore delay model for minimizing maximum propagation delay. Master's Thesis, Japan Advanced Institute of Science and Technology, 1999

Kahng, A.B., and Robins, G. [1995]: On Optimal Interconnections for VLSI. Kluwer Academic Publishers, Boston 1995

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

Korte, B. and Vygen, J. [2008]: Combinatorial Optimization: Theory and Algorithms. Fourth Edition. Springer, Berlin 2008

Masek, W.J. [1978]: Some *NP*-complete set covering problems. Unpublished manuscript, M.I.T., Cambridge 1978

Maßberg, J., and Vygen, J. [2005]: Approximation algorithms for a facility location problem with service capacities. ACM Transactions on Algorithms, to

appear. A preliminary extended abstract appeared in: Approximation, Randomization and Combinatorial Optimization; Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2005); LNCS 3624 (C. Chekuri, K. Jansen, J.D.P. Rolim, L. Trevisan, eds). Springer, Berlin 2005, pp. 158–169

Müller, D. [2006]: Optimizing yield in global routing. Proceedings of the IEEE International Conference on Computer-Aided Design (2006), 480–486

Müller, D., and Vygen, J. [2008]: Faster min-max resource sharing and applications. Manuscript, in preparation

Murata, H., Fujiyoshi, K., Nakatake, S., and Kajitani, Y. [1995]: Rectangle-packing-based module placement. Proceedings of the IEEE International Conference on Computer-Aided Design (1995), 472–479

Orlin, J.B. [1993]: A faster strongly polynomial minimum cost flow algorithm. Operations Research 41 (1993), 338–350

Peyer, S. [2000]: Elmore-Delay-optimale Steinerbäume im VLSI-Design. Diploma thesis, University of Bonn, 2000

Peyer, S., Rautenbach, D., and Vygen, J. [2006]: A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing. Report No. 06964-OR, Research Institute for Discrete Mathematics, University of Bonn, 2006

Plotkin, S.A., Shmoys, D.B., and Tardos, É. [1995]: Fast approximation algorithms for fractional packing and covering problems. Mathematics of Operations Research 2 (1995), 257–301

Pohl, I. [1971]: Bi-directional search. Machine Intelligence 6 (1971), 124–140

Queyranne, M. [1986]: Performance ratio of polynomial heuristics for triangle inequality quadratic assignment problems. Operations Research Letters 4 (1986), 231–234

Rautenbach, D., Szegedy, C., and Werber, J. [2003]: Asymptotically optimal Boolean circuits for functions of the form $g_{n-1}(g_{n-2}(...g_3(g_2(g_1(x_1,x_2),x_3),x_4)...,x_{n-1}),x_n)$. Report No. 03931, Research Institute for Discrete Mathematics, University of Bonn, 2003

Rubin, F. [1974]: The Lee path connection algorithm. IEEE Transactions on Computers 23 (1974), 907–914

Solomon, M., and Finkel, R.A. [1980]: A note on enumerating binary trees. Journal of the ACM 27 (1980), 3–5

Spielman, D.A. [2007]: Spectral graph theory and its applications. Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (2007), 29–38

Takahashi, T. [2000]: A new encoding scheme for rectangle packing problem. Proceedings of the Asia and South Pacific Design Automation Conference (2000), 175–178

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

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

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

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

Young, N. [1995]: Randomized rounding without solving the linear program. Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (1995), 170–178