Resource Sharing, Routing, Chip Design

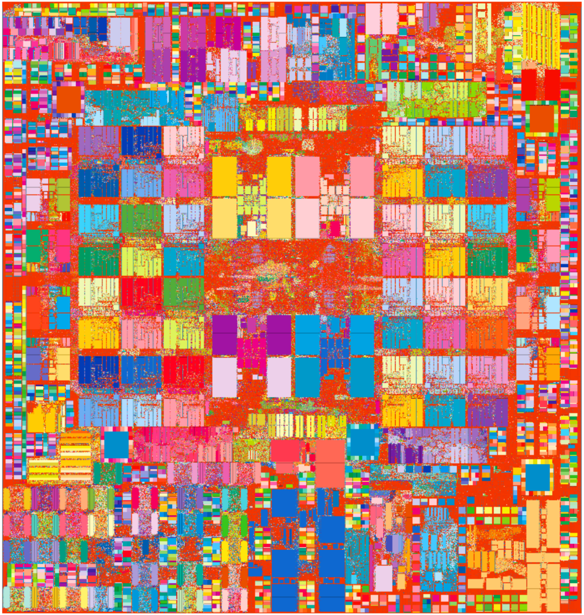
Jens Vygen

University of Bonn

---

joint work with Markus Ahrens, Stephan Held, Niko Klewinghaus, Dirk Müller, Klaus Radke, Daniel Rotter, Pietro Saccardi, Rudolf Scheifele, Christian Schulte, Vera Traub, et al.

Chip design: placement

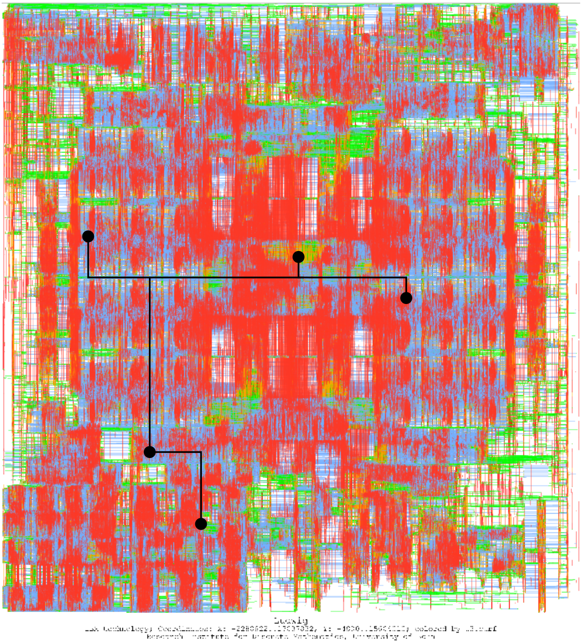


telecommunication chip with ≈

1 billion transistors

8 million circuits

Chip design: routing



9 routing layers

30 million pins

8 million nets

100 million
wire segments

100 million vias

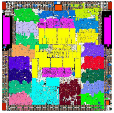
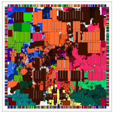
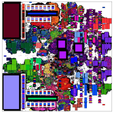
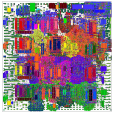
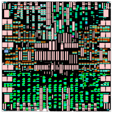
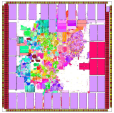
1 kilometer
total wire length

Combinatorial optimization in chip design

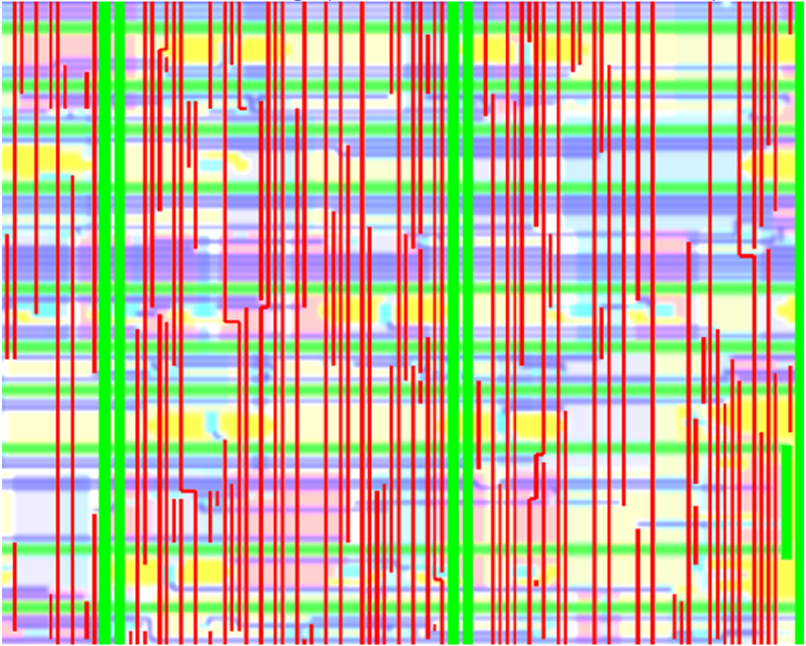
- Boolean functions and circuits
- shortest paths, TSP, spanning trees, Steiner trees
- rectangle packing, knapsack problem, bin packing
- facility location, partitioning, clustering
- maximum flows, discrete time-cost tradeoff problem
- minimum mean cycles, parametric shortest paths
- transportation and minimum cost flows
- multicommodity flows, disjoint paths and trees, resource sharing

The BonnTools,

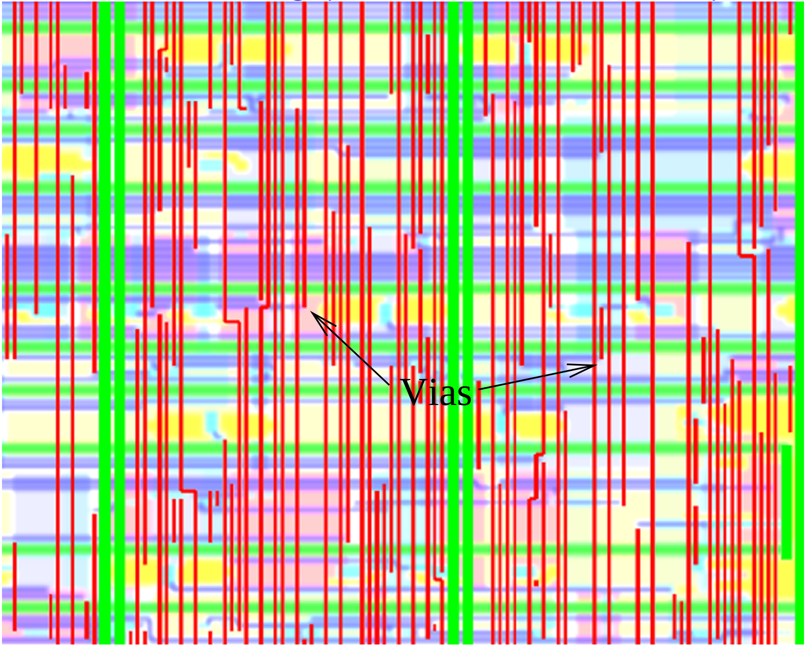
- developed by our group at the University of Bonn,
- cover all major areas of layout and timing optimization,
- include libraries for combinatorial optimization, advanced data structures, computational geometry, etc.,
- have more than a million lines of C++ code,
- are being used worldwide by IBM and other companies,
- have been used for the design of thousands of chips,
- including several complete microprocessor series
- and the most complex chips of major technology companies.

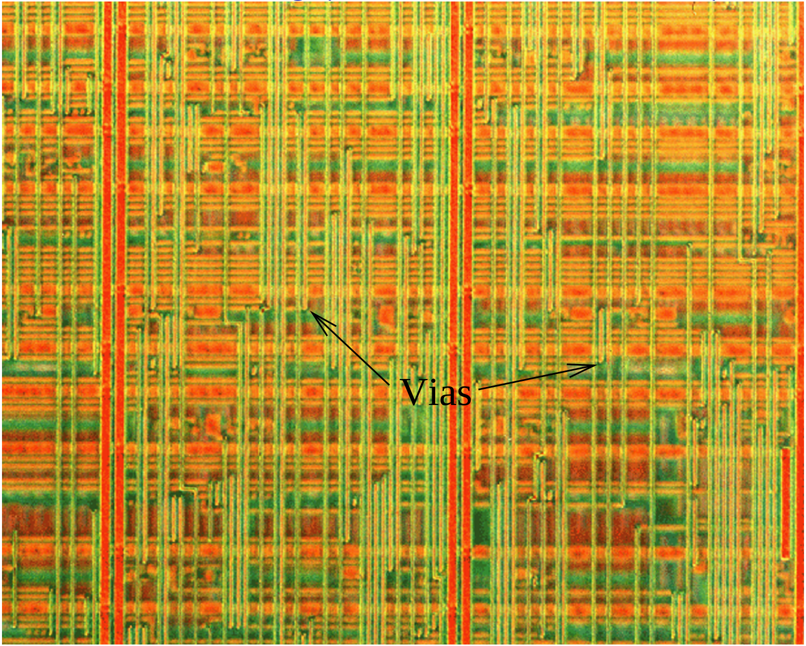
Detail of the routing (less than one millionth)



Detail of the routing (less than one millionth)



Detail of the routing (less than one millionth)



Routing: task

Instance:

- a number of routing planes
- a set of nets, where each net is a set of pins (terminals)
- a set of shapes for each pin
- a set of blockage shapes
- rules that tell if two given shapes are connected, separated

Task:

Compute a feasible routing, i.e.,
a set of wire shapes for each net, connecting the pins,
separate from blockages and shapes of other nets

Routing: task

Instance:

- a number of routing planes
- a set of nets, where each net is a set of pins (terminals)
- a set of shapes for each pin
- a set of blockage shapes
- rules that tell if two given shapes are connected, separated
- timing constraints
- information on power consumption, yield, ...

Task:

Compute a feasible routing, i.e.,
a set of wire shapes for each net, connecting the pins,
separate from blockages and shapes of other nets

- such that all timing constraints are met
- and the (estimated) power consumption and/or manufacturing cost is minimized.

Routing: simplified view

Find vertex-disjoint Steiner trees connecting given terminal sets in a 3-dimensional grid graph.

NP-hard:
no polynomial-time algorithm unless $P = NP$

Order of magnitude:
10 million Steiner trees in a graph with 500 billion vertices
$\rightarrow$ Even linear-time algorithms are too slow!

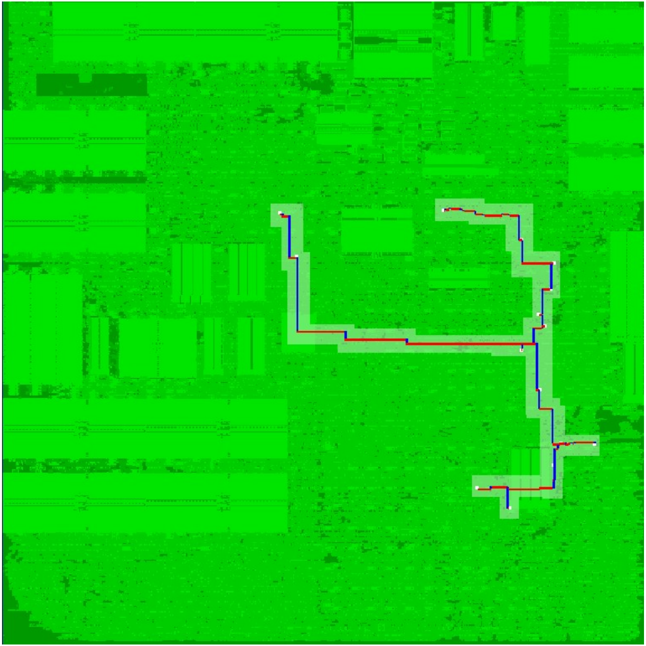
Global and detailed routing

Routing is usually performed in three phases:

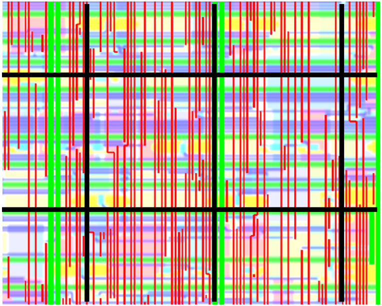
- Global routing: performs global optimization, determines a routing area (corridor) for each net, but no detailed view
- Detailed routing: constructs wires connecting each net within this corridor, respecting all design rules necessary for the lithographic process in fabrication
- Postoptimization: fix remaining violated constraints, improve the wiring by spreading and do some postprocessing for more robust manufacturing

Graphs are huge: 500 000 000 000 vertices in detailed routing, still 100 000 000 vertices in global routing.

Detailed routing in global routing corridor



Global routing: classical model



In each routing plane: contract regions of approx. 70x70 tracks to a single vertex

- compute capacities of edges between adjacent regions
- pack Steiner trees with respect to these edge capacities
- global optimization of objective functions
- Steiner tree yields routing corridor for each net
- Detailed routing computes detailed wires in these corridors by fast goal-oriented variants of Dijkstra's algorithm and optimal pin access (Hetzel [1998], Müller [2009], Peyer, Rautenbach, Vygen [2009], Klewinghaus [2013], Gester et al. [2013], Ahrens et al. [2015], Henke [2016], Ahrens, Rabenstein [2019])

Global routing: classical problem formulation

Instance:

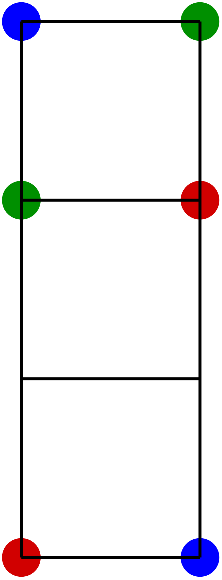
- a global routing (grid) graph with edge capacities
- a set of nets, each consisting of a set of vertices (terminals)

Task: find a Steiner tree for each net such that

- the edge capacities are respected,
- and (weighted) netlength is minimum.

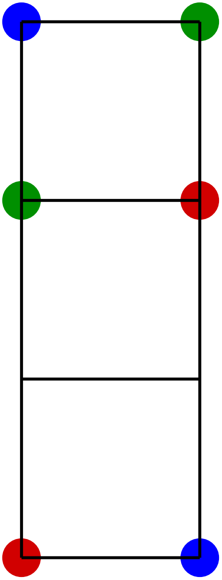
Simple example

- edge-disjoint paths problem
- 3 terminal pairs: blue, red, green
- each terminal pair has demand 1
- each edge has capacity 1



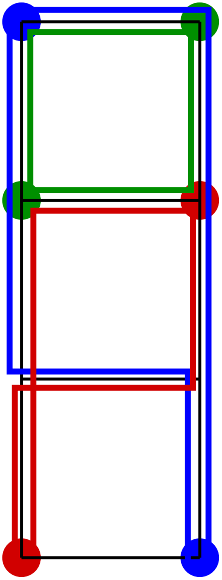
Simple example

- edge-disjoint paths problem
- 3 terminal pairs: blue, red, green
- each terminal pair has demand 1
- each edge has capacity 1
- no solution exists
- edge-disjoint paths problem is *NP*-hard (even in planar grids)



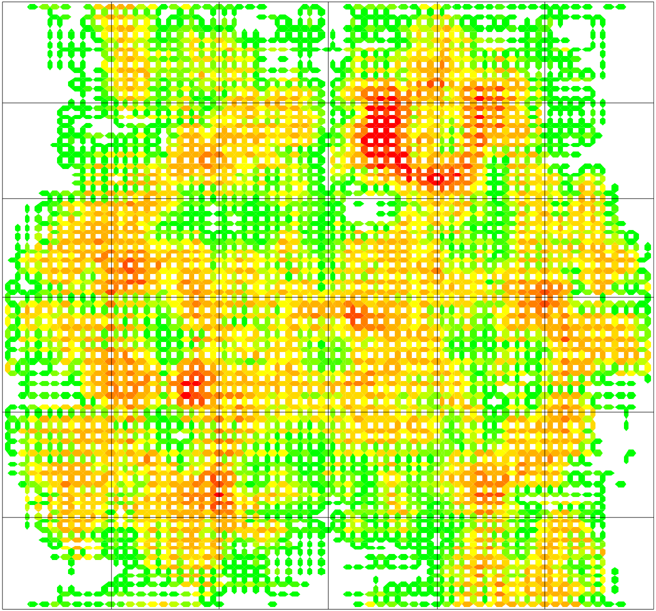
Simple example

- edge-disjoint paths problem
- 3 terminal pairs: blue, red, green
- each terminal pair has demand 1
- each edge has capacity 1
- no solution exists
- fractional solution exists
  (route $\frac{1}{2}$ along each colored path)



- edge-disjoint paths problem is *NP*-hard (even in planar grids)
- fractional relaxation can be solved in polynomial time by linear programming (but not fast enough)

Real example: global routing congestion map



Min-max resource sharing

Instance

- finite sets $\mathcal{R}$ of **resources** and $\mathcal{C}$ of **customers**
- for each $c \in \mathcal{C}$: a set $\mathcal{B}_c \subseteq \mathbb{R}^{\mathcal{R}}_{\geq 0}$ of **feasible solutions**

Task

- Find a $b_c \in \mathcal{B}_c$ for each $c \in \mathcal{C}$
  with minimum **congestion**

$$\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (b_c)_r \ .$$

Min-max resource sharing

Instance

- finite sets $\mathcal{R}$ of **resources** and $\mathcal{C}$ of **customers**
- for each $c \in \mathcal{C}$: a set $\mathcal{B}_c \subseteq \mathbb{R}^{\mathcal{R}}_{\geq 0}$ of **feasible solutions** given by an **oracle function** $f_c : \mathbb{R}^{\mathcal{R}}_{\geq 0} \to \mathcal{B}_c$ with

$$\sum_{r \in \mathcal{R}} y_r (f_c(y))_r \ \leq \ \sigma \inf_{b \in \mathcal{B}_c} \sum_{r \in \mathcal{R}} y_r b_r$$

for all price vectors $y \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ and some fixed $\sigma \geq 1$ (a σ-approximate **oracle**).

Task

- Find a $b_c \in \mathcal{B}_c$ for each $c \in \mathcal{C}$ with minimum **congestion**

$$\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (b_c)_r \ .$$

Min-max resource sharing

Instance

- finite sets $\mathcal{R}$ of **resources** and $\mathcal{C}$ of **customers**
- for each $c \in \mathcal{C}$: a set $\mathcal{B}_c \subseteq \mathbb{R}^{\mathcal{R}}_{\geq 0}$ of **feasible solutions** given by an **oracle function** $f_c : \mathbb{R}^{\mathcal{R}}_{\geq 0} \to \mathcal{B}_c$ with

$$\sum_{r \in \mathcal{R}} y_r (f_c(y))_r \;\leq\; \sigma \inf_{b \in \mathcal{B}_c} \sum_{r \in \mathcal{R}} y_r b_r$$

for all price vectors $y \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ and some fixed $\sigma \geq 1$ (a σ-approximate **oracle**).

Task

- Find a $b_c \in \mathrm{conv}(\mathcal{B}_c)$ for each $c \in \mathcal{C}$ with minimum **congestion**

$$\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (b_c)_r \,.$$

Resource sharing for global routing

Each net is a customer.

Define a resource r (with a capacity $\mathrm{cap}(r)$) for

- each edge of the global routing graph
- the total power consumption
- the critical area (for estimating the yield loss)
- each edge of the timing propagation graph

For each net c, let $\mathcal{B}_c$ contain, for each routing solution s for c, the vector $\left(\frac{\mathrm{usg}(s,r)}{\mathrm{cap}(r)}\right)_{r\in\mathcal{R}}$ where $\mathrm{usg}(s,r)$ tells how much s uses from r.

- We look for a solution with congestion at most 1.
- An objective function can be viewed as an additional resource (determine its capacity by binary search).

Rounding

- A solution to the min-max resource sharing instance is a convex combination $\sum_{b\in\mathcal{B}_c} p_b b$ of solutions for each net c ($p_b \geq 0$ ($b \in \mathcal{B}_c$), $\sum_{b\in\mathcal{B}_c} p_b = 1$)
- However, we need a single solution for each net.
- Use randomized rounding: randomly choose b with probability p_b, independently for each net. (Raghavan, Thompson [1987,1991], Raghavan [1988])
- If no solution consumes very much of any capacity, this yields a solution that exceeds capacities only slightly.
- New rounding and correction algorithms (e.g., Harris, Srinivasan [2013]) not better in practice (Bihler [2017])

Algorithms for min-max resource sharing

	oracle	# oracle calls
Grigoriadis, Khachiyan [1994]	strong, bounded	$\tilde{O}(\omega^{-2}\|\mathcal{C}\|^2)$
Grigoriadis, Khachiyan [1994]	strong, bounded	$\tilde{O}(\omega^{-2}\|\mathcal{C}\|)$ random.
Grigoriadis, Khachiyan [1996]	strong, unbounded	$\tilde{O}(\omega^{-2}\|\mathcal{C}\|\|\mathcal{R}\|)$
Jansen, Zhang [2008]	weak, unbounded	$\tilde{O}(\omega^{-2}\|\mathcal{C}\|\|\mathcal{R}\|)$
Müller, Radke, Vygen [2011]	weak, unbounded	$\tilde{O}(\omega^{-2}(\|\mathcal{C}\| + \|\mathcal{R}\|))$
Müller, Radke, Vygen [2011]	weak, bounded	$\tilde{O}(\omega^{-2}\|\mathcal{C}\|)$

All these algorithms and predecessors for special cases (Plotkin, Shmoys, Tardos [1995], Young [1995], Villavicencio, Grigoriadis [1996], Garg, Könemann [2007], ...) compute a $\sigma(1+\omega)$-approximate solution for any given $\omega > 0$.

Running times dominated by number of oracle calls.
Logarithmic terms and dependency on σ omitted.

For a strong oracle, σ can be chosen arbitrarily close to 1.

A bounded oracle respects a given upper bound on resource usage.

Bienstock and Iyengar [2004] obtained $\tilde{O}(\omega^{-1}\cdots)$ for fractional packing

Algorithms for min-max resource sharing

	oracle	# oracle calls
Grigoriadis, Khachiyan [1994]	strong, bounded	$\tilde{O}(\omega^{-2}\lvert\mathcal{C}\rvert^2)$
Grigoriadis, Khachiyan [1994]	strong, bounded	$\tilde{O}(\omega^{-2}\lvert\mathcal{C}\rvert)$ random.
Grigoriadis, Khachiyan [1996]	strong, unbounded	$\tilde{O}(\omega^{-2}\lvert\mathcal{C}\rvert\lvert\mathcal{R}\rvert)$
Jansen, Zhang [2008]	weak, unbounded	$\tilde{O}(\omega^{-2}\lvert\mathcal{C}\rvert\lvert\mathcal{R}\rvert)$
$\rightarrow$Müller, Radke, Vygen [2011]	weak, unbounded	$\tilde{O}(\omega^{-2}(\lvert\mathcal{C}\rvert + \lvert\mathcal{R}\rvert))$
Müller, Radke, Vygen [2011]	weak, bounded	$\tilde{O}(\omega^{-2}\lvert\mathcal{C}\rvert)$

All these algorithms and predecessors for special cases (Plotkin, Shmoys, Tardos [1995], Young [1995], Villavicencio, Grigoriadis [1996], Garg, Könemann [2007], ...) compute a $\sigma(1+\omega)$-approximate solution for any given $\omega > 0$.

Running times dominated by number of oracle calls.
Logarithmic terms and dependency on σ omitted.

For a strong oracle, σ can be chosen arbitrarily close to 1.

A bounded oracle respects a given upper bound on resource usage.

Bienstock and Iyengar [2004] obtained $\tilde{O}(\omega^{-1}\cdots)$ for fractional packing

Weak duality

Let
$$\lambda^* := \inf\left\{\max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (b_c)_r : b_c \in \mathcal{B}_c \,(c \in \mathcal{C})\right\}$$
(the "*optimum congestion*").

Lemma (Weak duality)

Let $y \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ be some price vector, not all-zero, and $opt_c(y) := \inf_{b\in\mathcal{B}_c} \sum_{r\in\mathcal{R}} y_r b_r$. Then

$$\frac{\sum_{c\in\mathcal{C}} opt_c(y)}{\sum_{r\in\mathcal{R}} y_r} \leq \lambda^*.$$

Weak duality

Let $$\lambda^* := \inf\left\{\max_{r\in\mathcal{R}} \sum_{c\in\mathcal{C}} (b_c)_r : b_c \in \mathcal{B}_c\,(c\in\mathcal{C})\right\}$$

(the "*optimum congestion*").

Lemma (Weak duality)

Let $y \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ be some price vector, not all-zero, and $opt_c(y) := \inf_{b\in\mathcal{B}_c} \sum_{r\in\mathcal{R}} y_r b_r$. Then

$$\frac{\sum_{c\in\mathcal{C}} opt_c(y)}{\sum_{r\in\mathcal{R}} y_r} \leq \lambda^*.$$

Proof

Let $(b_c \in \mathcal{B}_c)_{c\in\mathcal{C}}$ be a solution with congestion $\leq (1+\delta)\lambda^*$. Then

$$\frac{\sum_{c\in\mathcal{C}} \mathrm{opt}_c(y)}{\sum_{r\in\mathcal{R}} y_r} \leq \frac{\sum_{c\in\mathcal{C}} \sum_{r\in\mathcal{R}} y_r (b_c)_r}{\sum_{r\in\mathcal{R}} y_r} = \frac{\sum_{r\in\mathcal{R}} y_r \sum_{c\in\mathcal{C}} (b_c)_r}{\sum_{r\in\mathcal{R}} y_r}$$

$$\leq \frac{\sum_{r\in\mathcal{R}} y_r (1+\delta)\lambda^*}{\sum_{r\in\mathcal{R}} y_r} = (1+\delta)\lambda^*. \qquad \square$$

Bounding λ^*

Lemma (Weak duality)

Let $y \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ be some price vector, not all-zero. Then

$$\frac{\sum_{c \in \mathcal{C}} opt_c(y)}{\sum_{r \in \mathcal{R}} y_r} \leq \lambda^*.$$

Bounding λ^*

Lemma (Weak duality)

Let $y \in \mathbb{R}^{\mathcal{R}}_{\geq 0}$ be some price vector, not all-zero. Then

$$\frac{\sum_{c \in \mathcal{C}} opt_c(y)}{\sum_{r \in \mathcal{R}} y_r} \leq \lambda^*.$$

Corollary

Let $b_c := f_c(\mathbb{1})$ ($c \in \mathcal{C}$) and $\lambda^{ub} := \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (b_c)_r$. Then

$$\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \leq \frac{\sum_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (b_c)_r}{|\mathcal{R}|\sigma} \leq \frac{\sum_{c \in \mathcal{C}} opt_c(\mathbb{1})}{|\mathcal{R}|} \leq \lambda^* \leq \lambda^{ub} \, .$$

□

Scaling

We know $\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \leq \lambda^* \leq \lambda^{ub}$.

Scaling

We know $\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \leq \lambda^* \leq \lambda^{ub}$.

1. Set $j := 0$.
2. Scale $\mathcal{B}_c^{(j)} := \{\frac{2^j}{\lambda^{ub}} b : b \in \mathcal{B}_c\}$. Note that $\lambda^{*(j)} \leq 1$.

Scaling

We know $\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \leq \lambda^* \leq \lambda^{ub}$.

1. Set $j := 0$.
2. Scale $\mathcal{B}_c^{(j)} := \{\frac{2^j}{\lambda^{ub}} b : b \in \mathcal{B}_c\}$. Note that $\lambda^{*(j)} \leq 1$.
3. Find a solution with congestion $\lambda^{(j)} \leq \frac{5}{4}\sigma\lambda^{*(j)} + \frac{1}{4}$.

Scaling

We know $\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \leq \lambda^* \leq \lambda^{ub}$.

1. Set $j := 0$.
2. Scale $\mathcal{B}_c^{(j)} := \{\frac{2^j}{\lambda^{ub}} b : b \in \mathcal{B}_c\}$. Note that $\lambda^{*(j)} \leq 1$.
3. Find a solution with congestion $\lambda^{(j)} \leq \frac{5}{4}\sigma\lambda^{*(j)} + \frac{1}{4}$.
4. If $\lambda^{(j)} \leq \frac{1}{2}$, then increment j and go to 2.

Scaling

We know $\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \leq \lambda^* \leq \lambda^{ub}$.

1. Set $j := 0$.
2. Scale $\mathcal{B}_c^{(j)} := \{\frac{2^j}{\lambda^{ub}} b : b \in \mathcal{B}_c\}$. Note that $\lambda^{*(j)} \leq 1$.
3. Find a solution with congestion $\lambda^{(j)} \leq \frac{5}{4}\sigma\lambda^{*(j)} + \frac{1}{4}$.
4. If $\lambda^{(j)} \leq \frac{1}{2}$, then increment j and go to 2.
5. Now $\frac{1}{5\sigma} \leq \lambda^{*(j)} \leq 1$.
6. Find a solution with congestion $\lambda^{(j)} \leq \sigma(1 + \frac{\omega}{2})\lambda^{*(j)} + \frac{\omega}{10}$ (hence at most $\sigma(1 + \omega)$ times the optimum).

Scaling

We know $\frac{\lambda^{ub}}{|\mathcal{R}|\sigma} \leq \lambda^* \leq \lambda^{ub}$.

1. Set $j := 0$.
2. Scale $\mathcal{B}_c^{(j)} := \{\frac{2^j}{\lambda^{ub}} b : b \in \mathcal{B}_c\}$. Note that $\lambda^{*(j)} \leq 1$.
3. Find a solution with congestion $\lambda^{(j)} \leq \frac{5}{4}\sigma\lambda^{*(j)} + \frac{1}{4}$.
4. If $\lambda^{(j)} \leq \frac{1}{2}$, then increment j and go to 2.
5. Now $\frac{1}{5\sigma} \leq \lambda^{*(j)} \leq 1$.
6. Find a solution with congestion $\lambda^{(j)} \leq \sigma(1 + \frac{\omega}{2})\lambda^{*(j)} + \frac{\omega}{10}$ (hence at most $\sigma(1 + \omega)$ times the optimum).

Lemma (Main Lemma)

Let $\delta, \delta' > 0$. Suppose that $\lambda^* \leq 1$.
Then we can compute a solution with congestion at most

$$\sigma(1 + \delta)\lambda^* + \delta'$$

in

$$O\left((\delta\delta')^{-1}(|\mathcal{C}| + |\mathcal{R}|)\,\theta\,\log|\mathcal{R}|\right)$$

time, where θ is the time for an oracle call.

Core algorithm ("multiplicative weights method")

Input: An instance of the min-max resource sharing problem.
Output: A convex combination of vectors in $\mathcal{B}_c$ for each $c \in \mathcal{C}$.

$t := \left\lceil \frac{3\sigma \ln |\mathcal{R}|}{\delta\delta'} \right\rceil$, $\epsilon := \frac{\delta}{3\sigma}$.
$\alpha_r := 0$, $y_r := 1$ for each $r \in \mathcal{R}$.
$x_{c,b} := 0$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$.
For $p := 1$ to t **do**: (perform t **phases**)
  For $c \in \mathcal{C}$ **do**:
    $b := f_c(y)$. (call oracle)

    $x_{c,b} := x_{c,b} + 1$. (record solution)
    $\alpha := \alpha + b$. (update resource consumption)
    For each $r \in \mathcal{R}$ with $b_r \neq 0$ **do**:
      $y_r := e^{\epsilon\alpha_r}$. (update prices)
$x_{c,b} := \frac{1}{t} x_{c,b}$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$. (normalize)

Core algorithm ("multiplicative weights method")

Input: An instance of the min-max resource sharing problem.
Output: A convex combination of vectors in $\mathcal{B}_c$ for each $c \in \mathcal{C}$.

$t := \left\lceil \frac{3\sigma \ln |\mathcal{R}|}{\delta\delta'} \right\rceil, \ \epsilon := \frac{\delta}{3\sigma}$.
$\alpha_r := 0, \ y_r := 1$ for each $r \in \mathcal{R}$.
$x_{c,b} := 0, \ X_c := 0$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$.
For $p := 1$ to t **do**: (perform t **phases**)
    While there exists $c \in \mathcal{C}$ with $X_c < p$ **do**:
        $b := f_c(y)$. (call oracle)
        $\xi := \min\{p - X_c, 1/\max\{b_r : r \in \mathcal{R}\}\}$.
        $x_{c,b} := x_{c,b} + \xi, \ X_c := X_c + \xi$. (record solution)
        $\alpha := \alpha + \xi b$. (update resource consumption)
        For each $r \in \mathcal{R}$ with $b_r \neq 0$ **do**:
            $y_r := e^{\epsilon \alpha_r}$. (update prices)
$x_{c,b} := \frac{1}{t} x_{c,b}$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$. (normalize)

Proof of performance guarantee (sketch)

Lemma

Let (x, y) be the output of the algorithm with congestion $\lambda := \max_{r \in \mathcal{R}} \lambda_r$, where

$$\lambda_r := \sum_{c \in \mathcal{C}} \Big(\sum_{b \in \mathcal{B}_c} x_{c,b} b \Big)_r$$

Then

$$\lambda \leq \frac{1}{\epsilon t} \ln \Big(\sum_{r \in \mathcal{R}} y_r \Big).$$

Proof of performance guarantee (sketch)

Lemma

Let (x, y) *be the output of the algorithm with congestion* $\lambda := \max_{r\in\mathcal{R}} \lambda_r$, *where*

$$\lambda_r := \sum_{c\in\mathcal{C}} \Big(\sum_{b\in\mathcal{B}_c} x_{c,b} b \Big)_r$$

Then

$$\lambda \leq \frac{1}{\epsilon t} \ln \Big(\sum_{r\in\mathcal{R}} y_r \Big).$$

Proof: For $r \in \mathcal{R}$:

$$\lambda_r = \sum_{c\in\mathcal{C}} \sum_{b\in\mathcal{B}_c} x_{c,b} b_r = \frac{\alpha_r}{t} = \frac{1}{\epsilon t} \ln \left(e^{\epsilon\alpha_r} \right) = \frac{1}{\epsilon t} \ln y_r \leq \frac{1}{\epsilon t} \ln \Big(\sum_{r\in\mathcal{R}} y_r \Big).$$

$\square$

Proof of performance guarantee (sketch)

Lemma (Main Lemma)

Let $\delta, \delta' > 0$. Suppose that $\epsilon' \lambda^* < 1$, where $\epsilon' := (e^\epsilon - 1)\sigma$. Then the algorithm computes a solution with congestion at most

$$\sigma(1 + \delta)\lambda^* + \delta' \, .$$

Proof of performance guarantee (sketch)

Lemma (Main Lemma)

Let $\delta, \delta' > 0$. Suppose that $\epsilon' \lambda^* < 1$, where $\epsilon' := (e^{\epsilon} - 1)\sigma$. Then the algorithm computes a solution with congestion at most

$$\sigma(1 + \delta)\lambda^* + \delta' \,.$$

Sketch of proof:

- Congestion is at most $\frac{1}{\epsilon t} \ln\left(\sum_{r \in \mathcal{R}} y_r^{(t)}\right)$.

where $y^{(i)}$ is the price vector at the end of the i-th phase.

Proof of performance guarantee (sketch)

Lemma (Main Lemma)

Let $\delta, \delta' > 0$. Suppose that $\epsilon' \lambda^* < 1$, where $\epsilon' := (e^\epsilon - 1)\sigma$. Then the algorithm computes a solution with congestion at most

$$\sigma(1 + \delta)\lambda^* + \delta' .$$

Sketch of proof:

- Congestion is at most $\frac{1}{\epsilon t} \ln\left(\sum_{r \in \mathcal{R}} y_r^{(t)}\right)$.
- Initially, we have $\left(\sum_{r \in \mathcal{R}} y_r^{(0)}\right) = |\mathcal{R}|$.

where $y^{(i)}$ is the price vector at the end of the i-th phase.

Proof of performance guarantee (sketch)

Lemma (Main Lemma)

Let $\delta, \delta' > 0$. Suppose that $\epsilon' \lambda^* < 1$, where $\epsilon' := (e^\epsilon - 1)\sigma$.
Then the algorithm computes a solution with congestion at most

$$\sigma(1+\delta)\lambda^* + \delta' \ .$$

Sketch of proof:

- Congestion is at most $\frac{1}{\epsilon t} \ln\Big(\sum_{r \in \mathcal{R}} y_r^{(t)}\Big)$.
- Initially, we have $\Big(\sum_{r \in \mathcal{R}} y_r^{(0)}\Big) = |\mathcal{R}|$.
- Price y_r multiplied by $e^{\epsilon \xi b_r} \leq (1 + (e^\epsilon - 1)\xi b_r)$ in each iteration.
- Short calculation yields

$$\sum_{r \in \mathcal{R}} y_r^{(p)} \leq \sum_{r \in \mathcal{R}} y_r^{(p-1)} + \epsilon' \sum_{c \in \mathcal{C}} \mathrm{opt}_c(y^{(p)}),$$

where $y^{(i)}$ is the price vector at the end of the i-th phase.

Proof of performance guarantee (sketch)

We had $\sum_{r \in \mathcal{R}} y_r^{(p)} \leq \sum_{r \in \mathcal{R}} y_r^{(p-1)} + \epsilon' \sum_{c \in \mathcal{C}} \mathrm{opt}_c(y^{(p)})$.

Proof of performance guarantee (sketch)

We had $\sum_{r\in\mathcal{R}} y_r^{(p)} \le \sum_{r\in\mathcal{R}} y_r^{(p-1)} + \epsilon' \sum_{c\in\mathcal{C}} \mathrm{opt}_c(y^{(p)})$.

By weak duality, $\epsilon' \frac{\sum_{c\in\mathcal{C}} \mathrm{opt}_c(y^{(p)})}{\sum_{r\in\mathcal{R}} y_r^{(p)}} \le \epsilon'\lambda^* < 1$.

We get

$$\sum_{r\in\mathcal{R}} y_r^{(p)} \le \frac{1}{1-\epsilon'\lambda^*} \sum_{r\in\mathcal{R}} y_r^{(p-1)}$$

Proof of performance guarantee (sketch)

We had $\sum_{r\in\mathcal{R}} y_r^{(p)} \le \sum_{r\in\mathcal{R}} y_r^{(p-1)} + \epsilon' \sum_{c\in\mathcal{C}} \mathrm{opt}_c(y^{(p)})$.

By weak duality, $\epsilon' \frac{\sum_{c\in\mathcal{C}} \mathrm{opt}_c(y^{(p)})}{\sum_{r\in\mathcal{R}} y_r^{(p)}} \le \epsilon'\lambda^* < 1$.

We get

$$\sum_{r\in\mathcal{R}} y_r^{(p)} \le \frac{1}{1-\epsilon'\lambda^*} \sum_{r\in\mathcal{R}} y_r^{(p-1)}$$

and thus

$$\sum_{r\in\mathcal{R}} y_r^{(t)} \le \frac{|\mathcal{R}|}{(1-\epsilon'\lambda^*)^t} = |\mathcal{R}| \left(1 + \frac{\epsilon'\lambda^*}{1-\epsilon'\lambda^*}\right)^t \le |\mathcal{R}| e^{t\epsilon'\lambda^*/(1-\epsilon'\lambda^*)} .$$

Proof of performance guarantee (sketch)

We had $\sum_{r \in \mathcal{R}} y_r^{(p)} \leq \sum_{r \in \mathcal{R}} y_r^{(p-1)} + \epsilon' \sum_{c \in \mathcal{C}} \mathrm{opt}_c(y^{(p)})$.

By weak duality, $\epsilon' \frac{\sum_{c \in \mathcal{C}} \mathrm{opt}_c(y^{(p)})}{\sum_{r \in \mathcal{R}} y_r^{(p)}} \leq \epsilon' \lambda^* < 1$.

We get

$$\sum_{r \in \mathcal{R}} y_r^{(p)} \leq \frac{1}{1 - \epsilon' \lambda^*} \sum_{r \in \mathcal{R}} y_r^{(p-1)}$$

and thus

$$\sum_{r \in \mathcal{R}} y_r^{(t)} \leq \frac{|\mathcal{R}|}{(1 - \epsilon' \lambda^*)^t} = |\mathcal{R}| \left(1 + \frac{\epsilon' \lambda^*}{1 - \epsilon' \lambda^*}\right)^t \leq |\mathcal{R}| e^{t \epsilon' \lambda^* / (1 - \epsilon' \lambda^*)} .$$

Together with $\lambda \leq \frac{1}{\epsilon t} \ln\left(\sum_{r \in \mathcal{R}} y_r^{(t)}\right)$, this proves the claim. $\square$

Number of oracle calls

After each oracle call

- either $X_c = p$ (happens only $t\,|\mathcal{C}|$ times)
- or $\xi b_r = 1$ for some $r \in \mathcal{R}$ (increases the price of r by e^{ϵ}).

Hence the number of oracle calls is

$$O\left((\delta\delta')^{-1}(|\mathcal{C}| + |\mathcal{R}|) \log |\mathcal{R}|\right).$$

Number of oracle calls

After each oracle call

- either $X_c = p$ (happens only $t\,|\mathcal{C}|$ times)
- or $\xi b_r = 1$ for some $r \in \mathcal{R}$ (increases the price of r by e^ϵ).

Hence the number of oracle calls is

$$O\left((\delta\delta')^{-1}(|\mathcal{C}| + |\mathcal{R}|)\log|\mathcal{R}|\right).$$

The above scaling algorithm computes a $\sigma(1+\omega)$-approximate solution in

$$O((\omega^{-2} + \log|\mathcal{R}|)(|\mathcal{C}| + |\mathcal{R}|)\,\theta\log|\mathcal{R}|)$$

time, where θ is the time for an oracle call.

Main result

The above scaling algorithm computes a $\sigma(1+\omega)$-approximate solution in $O((\omega^{-2}+\log|\mathcal{R}|)(|\mathcal{C}|+|\mathcal{R}|)\,\theta\log|\mathcal{R}|)$ time.

Main result

The above scaling algorithm computes a $\sigma(1+\omega)$-approximate solution in $O((\omega^{-2} + \log|\mathcal{R}|)(|\mathcal{C}| + |\mathcal{R}|)\,\theta \log|\mathcal{R}|)$ time.

Using binary search instead of simple scaling, and a variant of the Main Lemma in which $\lambda^* \leq 1$ is not guaranteed, we obtain:

Theorem

We can compute a $\sigma(1+\omega)$-approximate solution in $O((\omega^{-2} + \log\log|\mathcal{R}|)(|\mathcal{C}| + |\mathcal{R}|)\,\theta \log|\mathcal{R}|)$ time.

Faster *and* more general than all previous algorithms!

Main result

The above scaling algorithm computes a $\sigma(1+\omega)$-approximate solution in $O((\omega^{-2} + \log|\mathcal{R}|)(|\mathcal{C}| + |\mathcal{R}|)\,\theta \log|\mathcal{R}|)$ time.

Using binary search instead of simple scaling, and a variant of the Main Lemma in which $\lambda^* \leq 1$ is not guaranteed, we obtain:

Theorem

We can compute a $\sigma(1+\omega)$-approximate solution in $O((\omega^{-2} + \log\log|\mathcal{R}|)(|\mathcal{C}| + |\mathcal{R}|)\,\theta \log|\mathcal{R}|)$ time.

Faster *and* more general than all previous algorithms!

Extensions for practical application:

- Most oracle calls not necessary; reuse previous result if still good enough. Use lower bounds to decide
- Avoid binary search by re-scaling objective function resource
- Parallelization (without loss of theoretical guarantees!)

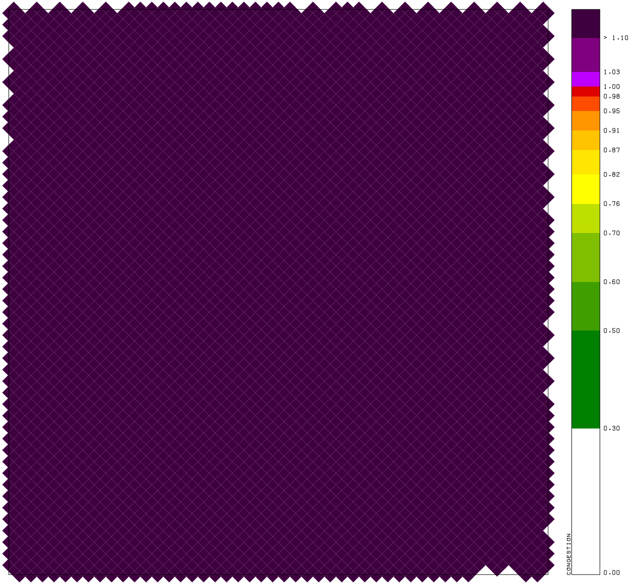
The algorithm in practice

- In practice, results are much better than theoretical performance guarantees. Usually 20–100 iterations suffice.
- Only few constraints are violated (even after rounding); these are corrected easily by *ripup-and-reroute*.
- Detailed routing can realize the solution well, due to good capacity estimations.
- Small integrality gap and approximate dual solution
$\Rightarrow$ infeasibility proof can be found for most infeasible instances
- New constraints can be added easily

Open problems:

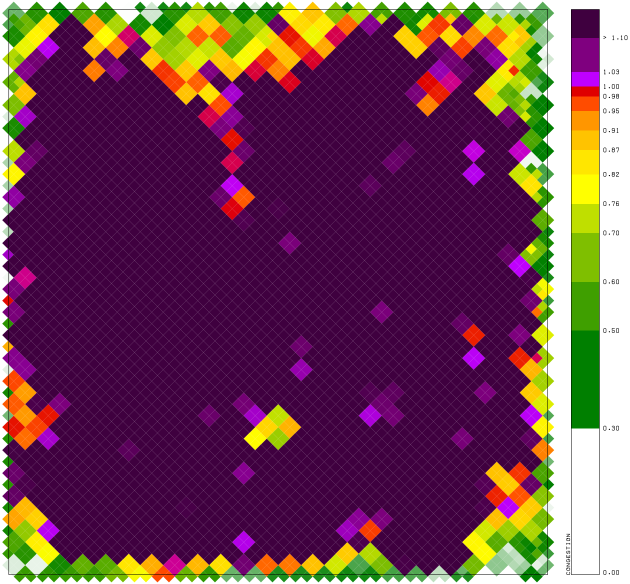
- What if one resource is very bad? Guarantee for the rest?
- Warmstart with good prices?
- Need binary search?

The algorithm in practice

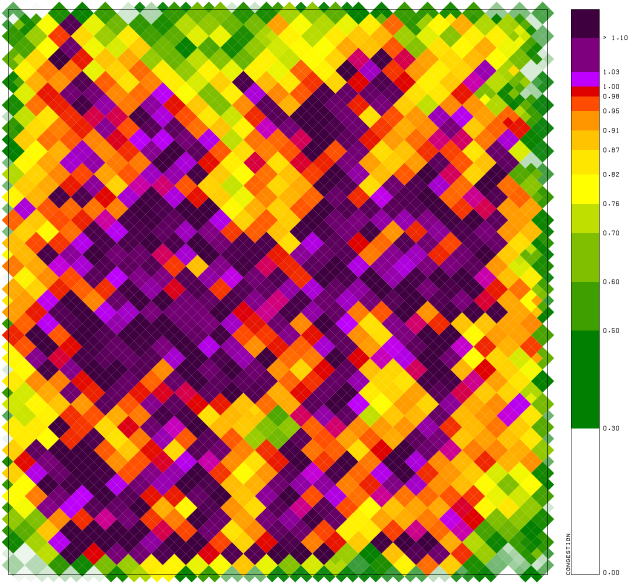


The algorithm in practice

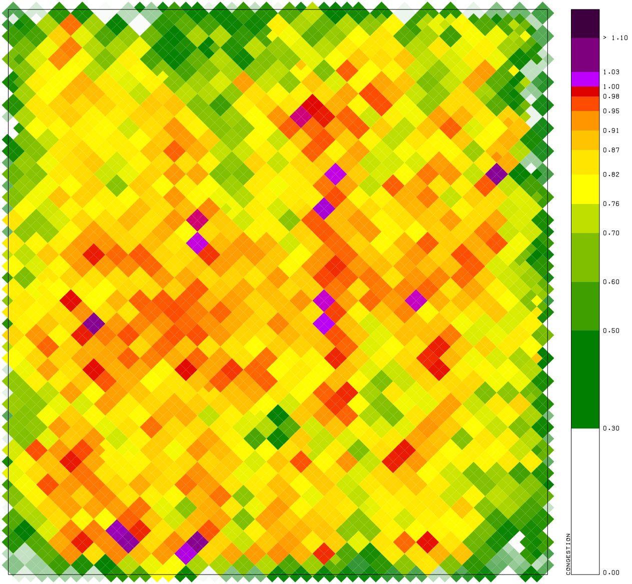
The algorithm in practice



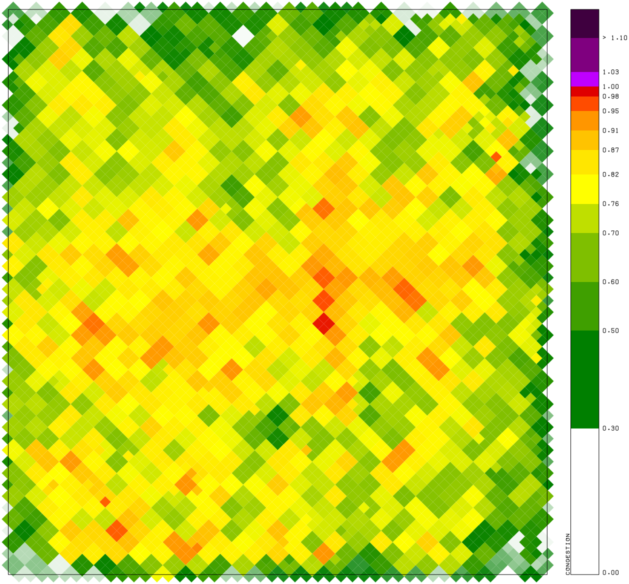
The algorithm in practice



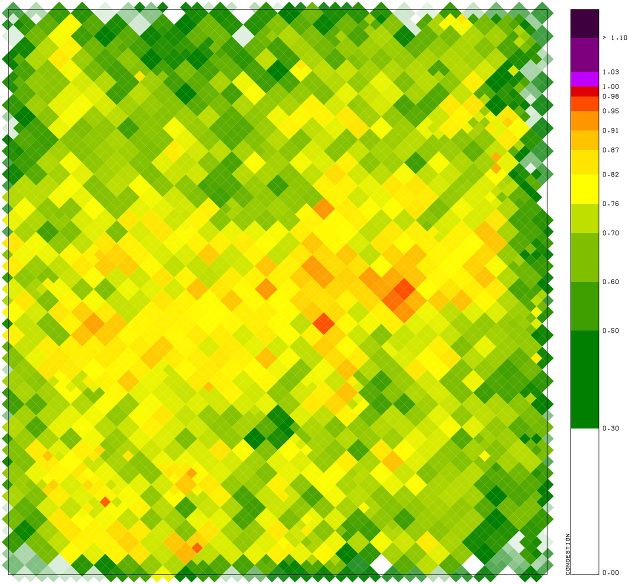
The algorithm in practice



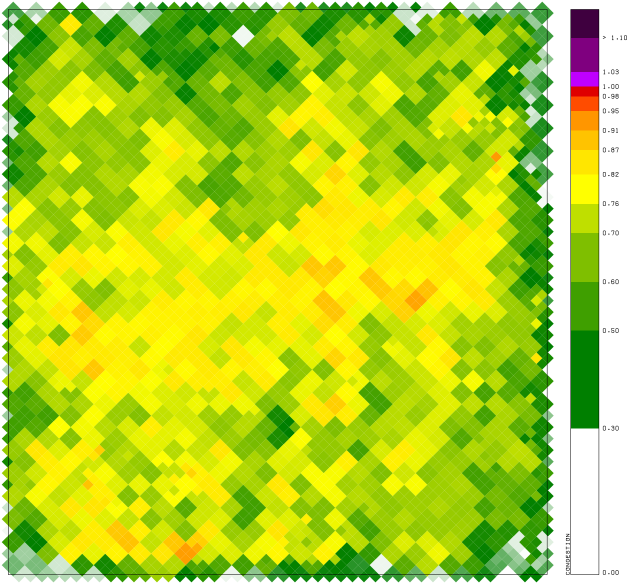
The algorithm in practice



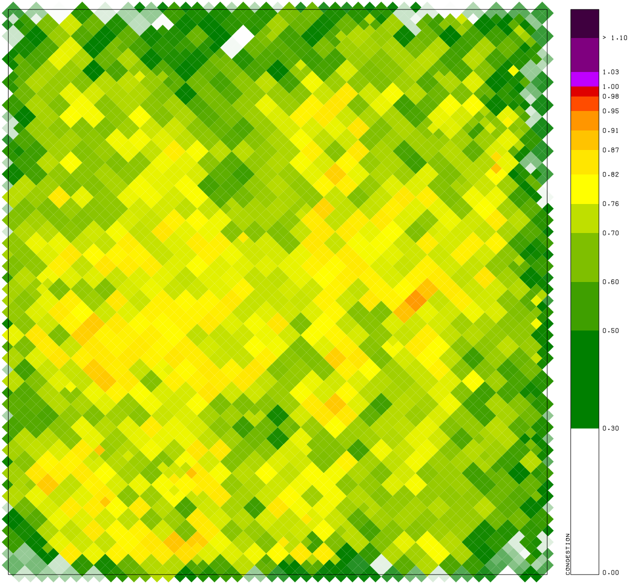
The algorithm in practice



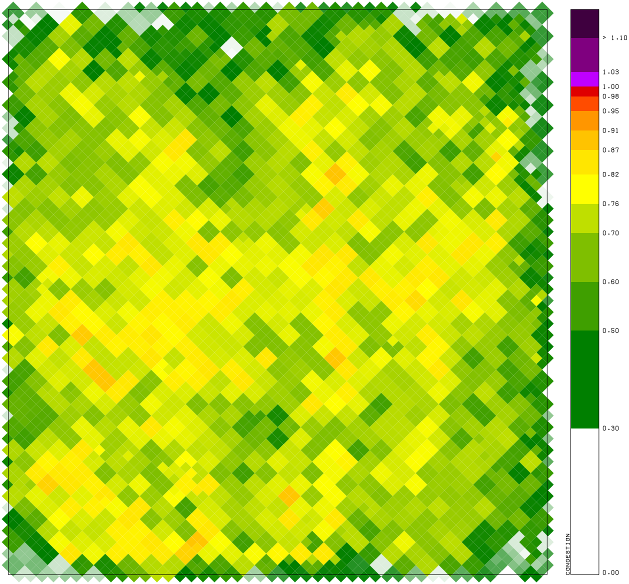
The algorithm in practice



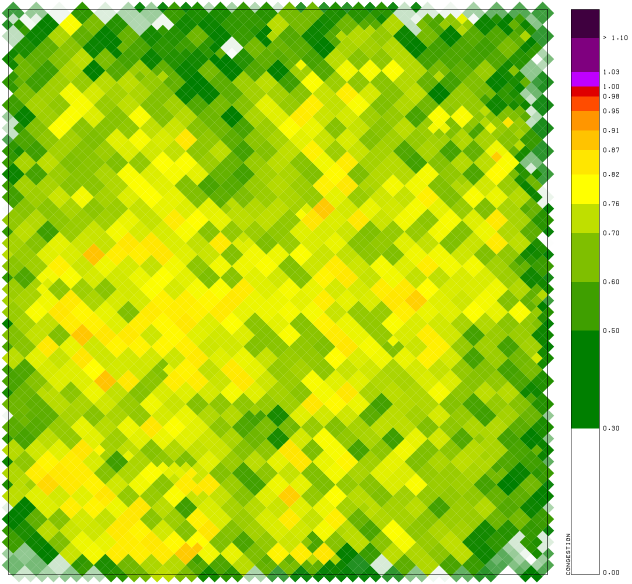
The algorithm in practice



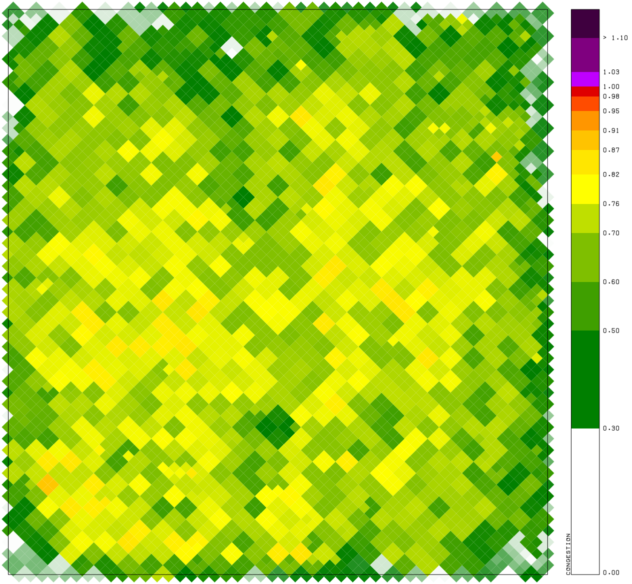
The algorithm in practice



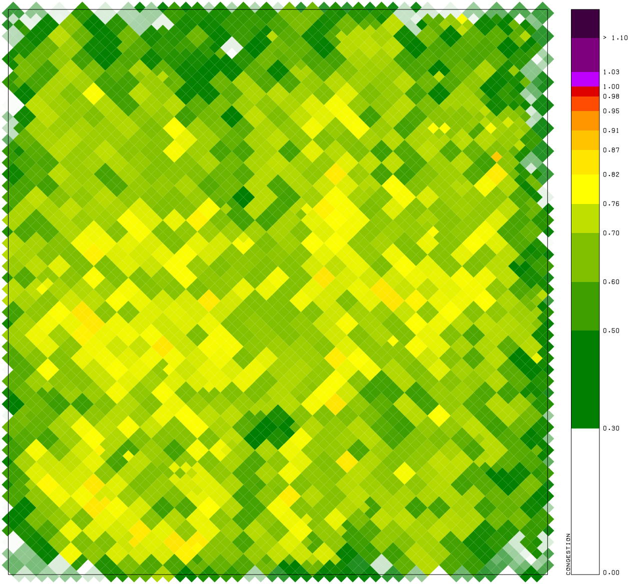
The algorithm in practice



The algorithm in practice



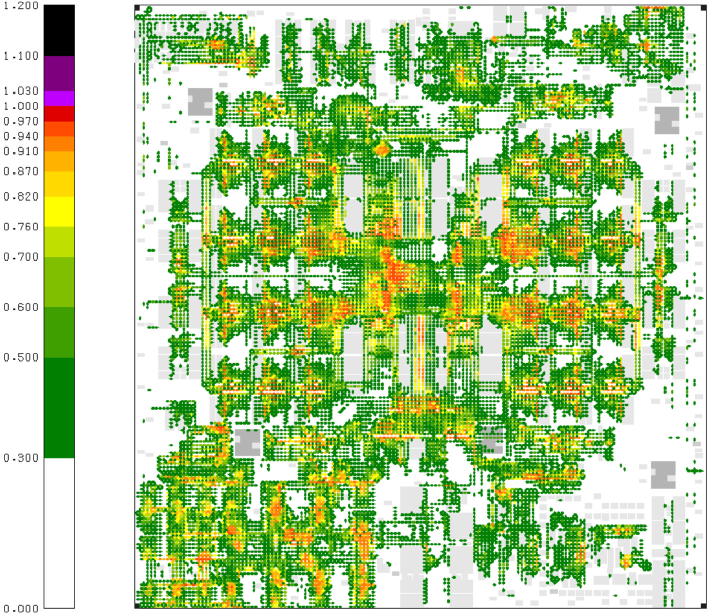
The algorithm in practice



Near optimality

chip (# nets)	$\epsilon = \frac{100}{t}$	λ_{fract}	λ_{lb}	λ_{rounded}	λ_{final}	**% gap (fract.)**	**% gap (final)**	**running time**
Rose	0.5	0.950	0.941	2.000	1.000	1.91	1.96	0:01:45
(594 084)	1.0	0.950	0.939	2.000	1.000	1.97	2.01	0:01:08
	5.0	0.949	0.930	2.000	1.000	2.41	2.43	0:00:28
Georg	0.5	0.950	0.942	2.000	1.000	1.53	1.56	0:02:38
(783 685)	1.0	0.950	0.940	3.000	1.000	1.61	1.63	0:01:40
	5.0	0.949	0.934	2.000	1.000	1.90	1.91	0:00:49
Camilla	0.5	0.950	0.937	4.000	1.000	2.52	2.56	0:51:09
(3 582 559)	1.0	0.950	0.933	4.000	1.000	2.59	2.61	0:33:05
	5.0	0.950	0.918	3.000	1.000	3.34	3.33	0:11:23
Tomoko	0.5	1.629	1.449	3.000	1.600	1.12	1.14	0:23:43
(5 340 088)	1.0	1.625	1.449	3.000	1.600	1.18	1.20	0:15:21
	5.0	1.667	1.369	2.500	1.600	1.36	1.36	0:07:09
Andre	0.5	0.950	0.938	3.000	1.000	2.25	2.38	1:18:54
(7 039 094)	1.0	0.950	0.935	3.000	1.000	2.38	2.48	0:50:43
	5.0	0.950	0.923	3.000	1.000	2.92	2.96	0:17:32

Global routing result for the telecommunication chip

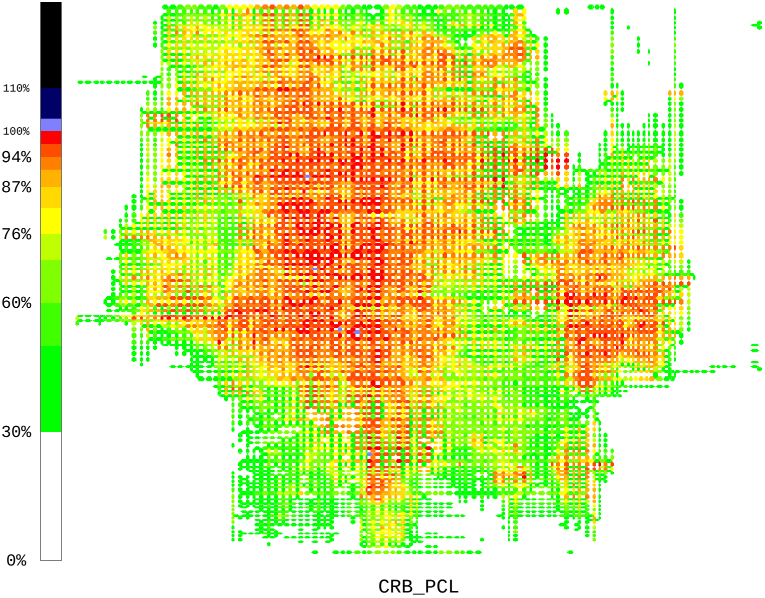


Overall result: comparison to an industrial router

	Time (h:mm:ss)		Wires	Vias	Scenic nets			Errors
	BonnRoute	Total	(m)		25%	50%	100%	
Industrial		9:18:14	16.57	11 160 081	44 419	21 841	2807	570
Bonn	1:17:35	3:55:32	14.86	8 640 867	2689	355	31	222
		-57.81%	-10.3%	-22.6%	-93.9%	-98.4%	-98.9%	-61.1%

- 14 nm testbed (14 instances with a total of 1 159 326 nets)
- both routers with 20 threads
- BonnRoute followed by industrial cleanup tool

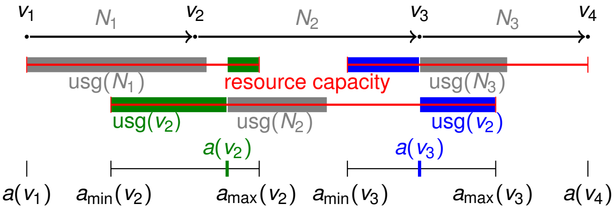
Congestion map of a difficult instance



RESEARCH INSTITUTE FOR DISCRETE MATHEMATICS, UNIVERSITY OF BONN

Modeling timing constraints via dynamic delay budgets

- Timing constraints are modeled by an acyclic digraph
- Arrival times at sources and latest allowed arrival times at sinks are given
- Delays on arcs depend on routing solution
- Add a resource for each of these arcs
- Add new customers for determining arrival times at intermediate vertices
- Results in dynamic delay budgets



Theorem (Held, Müller, Rotter, Scheifele, Traub, Vygen [2018])

Let $\omega > 0$.

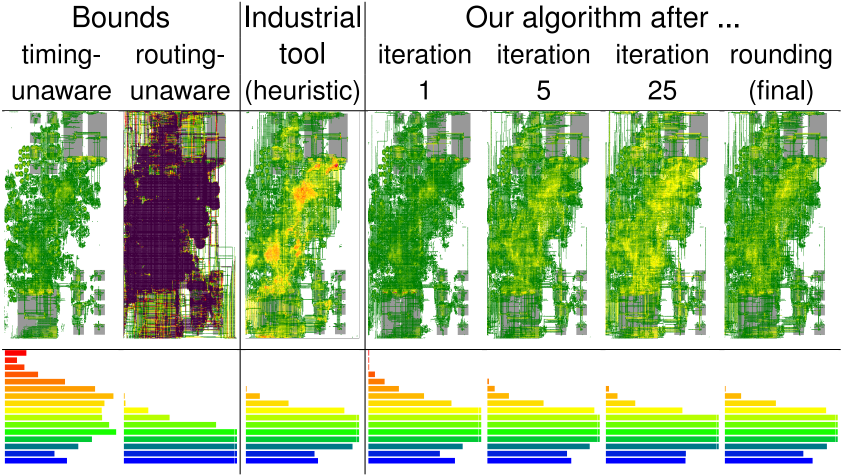
(a) *Given a* σ*-approximate routing oracle, we can compute, with* $\tilde{O}(\omega^{-2}(\mathcal{C}| + |\mathcal{R}|))$ *oracle calls, a solution that minimizes the congestion up to a factor* $\sigma(1 + \omega)$*.*

(b) *For nets with a bounded number of pins, we can obtain* $\sigma = 1$ *in polynomial time. Then, if there is a solution that satisfies all constraints, we get a solution that overloads edges by at most a factor* $1 + \omega$ *and violates timing constraints by at most* $\omega \cdot h$*, where* h *is a constant that depends on the instance only.*

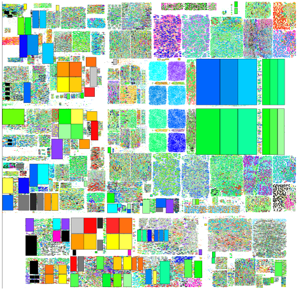
Sketch of proof:

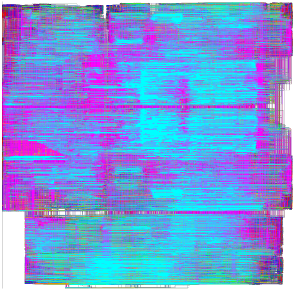
- Oracle for arrival time customers is trivial
- Resource sharing algorithm yields (a)
- Routing oracle for routing a net based on Dijkstra-Steiner algorithm (Hougardy, Silvanus, Vygen [2017])
- Increase capacities of timing resources so that all source-sink paths have the same capacity h.

Experimental results

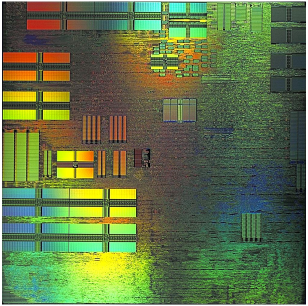


The world's fastest computer (Summit, 2018)





Conclusion: mathematics leads to better chips!



D. Müller, K. Radke, J. Vygen:
Faster min-max resource sharing in theory and practice.
Mathematical Programming Computation 3 (2011), 1–35

M. Gester, D. Müller, T. Nieberg, C. Panten, C. Schulte, J. Vygen:
BonnRoute: Algorithms and data structures for fast and good VLSI routing.
ACM Transactions on Design Automation of Electronic Systems 18 (2013), Article 32

S. Hougardy, J. Silvanus, J. Vygen:
Dijkstra meets Steiner: a fast exact goal-oriented Steiner tree algorithm.
Mathematical Programming Computation 9 (2017), 135–202

S. Held, D. Müller, D. Rotter, R. Scheifele, V. Traub, J. Vygen:
Global routing with timing constraints.
IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 37 (2018), 406–419