Faster Resource Sharing and Global Routing in Theory and Practice

Jens Vygen

University of Bonn

joint work with Dirk Müller and Klaus Radke
Chip design: placement

telecommunication chip with $\approx$

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

- shortest paths, Steiner trees
- rectangle packing
- facility location
- maximum flows, discrete time-cost tradeoff problems
- minimum mean cycles, parametric shortest paths
- transportation and minimum cost flows
- multicommodity flows, disjoint paths and trees
- resource sharing
- ... and others: minimum spanning trees, knapsack problem, bin packing, traveling salesman problem, Huffman codes, ...
- ... also used: advanced data structures, computational geometry, nonlinear programming, parallelization ...
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 one million lines of code, mostly in C++,
- are being used worldwide by IBM and other companies,
- have been used for the design of more than 1000 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, crosstalk, 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.

Order of magnitude:
10 million Steiner trees in a graph with 100 billion vertices!

→ Even linear-time algorithms are too slow!
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: **100 000 000 000 vertices** in detailed routing, **10 000 000 vertices** in global routing.
Global routing

In each routing plane:
contract regions of approx. 50x50 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 the detailed wires in these corridors by a very fast goal-oriented interval-labeling variant of Dijkstra’s algorithm (Hetzel [1998], Müller [2009], Peyer, Rautenbach, Vygen [2009], Humpola [2009])
Detailed routing in global routing corridor
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.
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.

Even simple special cases are \( NP \)-hard
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.

Even simple special cases are $NP$-hard

Graph smaller but still large: 10 000 000 vertices
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.

Even simple special cases are \textit{NP}-hard

Graph smaller but still large: 10 000 000 vertices

Timing, yield, power consumption, etc. ignored!
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 \( \text{NP}\)-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
Fractional relaxation: multicommodity flow problem

Instance:

- an undirected graph $G$ with capacities $u : E(G) \rightarrow \mathbb{N}$ and lengths $l : E(G) \rightarrow \mathbb{R}_{\geq 0}$
- a family $\mathcal{N}$ of nets (terminal pairs) with demands $w : \mathcal{N} \rightarrow \mathbb{N}$
Fractional relaxation: multicommodity flow problem

Instance:
- an undirected graph $G$ with capacities $u : E(G) \to \mathbb{N}$ and lengths $l : E(G) \to \mathbb{R}_{\geq 0}$
- a family $\mathcal{N}$ of nets (terminal pairs) with demands $w : \mathcal{N} \to \mathbb{N}$

Task: Find a flow $f_N$ of value $w(N)$ for each net $N \in \mathcal{N}$ such that

$$\sum_{N \in \mathcal{N}} f_N(e) \leq u(e) \quad \text{for } e \in E(G),$$

and

$$\sum_{N \in \mathcal{N}} \sum_{e \in E(G)} l(e)f_N(e)$$

is minimum.
Multicommodity flows: positive results

- Can be solved by linear programming (but too slow)
- Combinatorial fully polynomial approximation schemes:

- If edges have sufficient capacity, randomized rounding yields an integral solution violating capacity constraints only slightly (Raghavan, Thompson [1987,1991], Raghavan [1988])
- Can be generalized to Steiner trees instead of paths; applied to global routing instances (Albrecht [2001])

But this does not take timing constraints and global objectives (power consumption, yield) into account.
Multicommodity flows: positive results

- Can be solved by linear programming (but too slow)
- Combinatorial fully polynomial approximation schemes:
- If edges have sufficient capacity, randomized rounding yields an integral solution violating capacity constraints only slightly (Raghavan, Thompson [1987, 1991], Raghavan [1988])
Multicommodity flows: positive results

- Can be solved by linear programming (but too slow)
- If edges have sufficient capacity, randomized rounding yields an integral solution violating capacity constraints only slightly (Raghavan, Thompson [1987,1991], Raghavan [1988])
- Can be generalized to Steiner trees instead of paths; applied to global routing instances (Albrecht [2001])
Multicommodity flows: positive results

- Can be solved by linear programming (but too slow)
- If edges have sufficient capacity, randomized rounding yields an integral solution violating capacity constraints only slightly (Raghavan, Thompson [1987,1991], Raghavan [1988])
- Can be generalized to Steiner trees instead of paths; applied to global routing instances (Albrecht [2001])

But this does not take timing constraints and global objectives (power consumption, yield) into account.
Example: global routing congestion map
Constraints and objectives in routing

meet timing constraints

- all signals must arrive in time
- delays depend on electrical capacitances of nets
- capacitance of a net depends on length, width, plane, and distance to neighbour wires (nonlinearly!)

minimize power consumption

- power consumption roughly proportional to the electrical capacitance, weighted by switching activity

minimize cost

- minimize number of masks (number of routing planes), maximize yield, minimize design effort
Constraints and objectives in routing

meet timing constraints

- all signals must arrive in time
- delays depend on electrical capacitances of nets
- capacitance of a net depends on length, width, plane, and distance to neighbour wires (nonlinearly!)

minimize power consumption

- power consumption roughly proportional to the electrical capacitance, weighted by switching activity
Constraints and objectives in routing

meet timing constraints

▶ all signals must arrive in time
▶ delays depend on electrical capacitances of nets
▶ capacitance of a net depends on length, width, plane, and distance to neighbour wires (nonlinearly!)

minimize power consumption

▶ power consumption roughly proportional to the electrical capacitance, weighted by switching activity

minimize cost

▶ minimize number of masks (number of routing planes), maximize yield, minimize design effort
Capacitance estimation

- **area capacitance** (parallel plate capacitor): proportional to length times width
- **fringing capacitance**: proportional to length
- **coupling capacitance**: proportional to length, inversely proportional to distance to neighbour
Yield analysis: expected number of faults

The wiring yield loss (the expected percentage of chips that will not work properly due to wiring defects) is estimated by

\[ 1 - e^{-CA}, \]

where

1. \( CA = \sum_{z=1}^{z_{\text{max}}} \sum_{t \in T} w_{t,z} \int_{x} \int_{y} \int_{r_{t}(x,y,z)} 1 \frac{1}{r^{3}} \, dr \, dy \, dx \) (the critical area),
2. \( z_{\text{max}} \) is the number of routing planes,
3. \( T = \{ \text{open, short, via} \} \) are the defect types,
4. \( w_{t,z} > 0 \) are weights reflecting the defect probability, and
5. \( r_{t}(x,y,z) \) is the smallest size of a particle that is expected to cause a fault of type \( t \) if present at location \( (x,y,z) \).
General idea

Compute for each net $n$

- a Steiner tree $Y$ for $n$,
- and for each edge of $Y$ the amount of space assigned to net $n$ on edge $e$. 
General idea

Compute for each net $n$

- a Steiner tree $Y$ for $n$,
- and for each edge of $Y$ the amount of space assigned to net $n$ on edge $e$.

The contribution of $(n, e)$ to

- power consumption
- critical area
- delay

depends on whether $e$ is used and how much space is assigned.

These functions are convex.
Define resources for
- each edge of the global routing graph
- the total power consumption
- the critical area

Three ways to model timing constraints: based on
- the nets (the delay on each critical net is a resource; requires delay budgeting)
- paths (the delay of each critical path is a resource)
- the timing graph (work in progress)
Resource constraints

- We look for a solution where the total consumption of each resource is bounded by a given number.
- Without loss of generality, these bounds are all 1.
- An objective function can be viewed as an additional resource.
- Then it is equivalent to minimize the largest total consumption of a resource.
Min-max resource sharing

Instance

- finite sets $\mathcal{R}$ of resources and $\mathcal{C}$ of customers
Min-max resource sharing

Instance

- finite sets $\mathcal{R}$ of resources and $\mathcal{C}$ of customers
- for each $c \in \mathcal{C}$:
  - a convex set $\mathcal{B}_c$ of feasible solutions (a block) and
  - a convex resource consumption function $g_c : \mathcal{B}_c \rightarrow \mathbb{R}_\geq_0$

For all $y \in \mathbb{R}_{\geq 0}$ and some $\sigma \geq 1$, find a $b_c \in \mathcal{B}_c$ for each $c \in \mathcal{C}$ with minimum congestion:

$$\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (g_c(b_c))^r \leq \sigma \inf_{b \in \mathcal{B}_c} y^\top g_c(b)$$
Min-max resource sharing

Instance

- finite sets $\mathcal{R}$ of resources and $\mathcal{C}$ of customers
- for each $c \in \mathcal{C}$:
  - a convex set $\mathcal{B}_c$ of feasible solutions (a block) and
  - a convex resource consumption function $g_c : \mathcal{B}_c \to \mathbb{R}^{\mathcal{R} \geq 0}$

Task

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

$$\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (g_c(b_c))_r$$.
Min-max resource sharing

Instance

- finite sets $\mathcal{R}$ of resources and $\mathcal{C}$ of customers
- for each $c \in \mathcal{C}$:
  - a convex set $\mathcal{B}_c$ of feasible solutions (a block) and
  - a convex resource consumption function $g_c : \mathcal{B}_c \to \mathbb{R}^\mathcal{R}_{\geq 0}$

given by an oracle function $f_c : \mathbb{R}^\mathcal{R}_{\geq 0} \to \mathcal{B}_c$ with

$$y^\top g_c(f_c(y)) \leq \sigma \inf_{b \in \mathcal{B}_c} y^\top g_c(b)$$

for all $y \in \mathbb{R}^\mathcal{R}_{\geq 0}$ and some $\sigma \geq 1$ (a block solver).

Task

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

$$\max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (g_c(b_c))_r.$$
Special case: multicommodity flow

Instance:
- an undirected graph \( G \) with capacities \( u : E(G) \to \mathbb{N} \) and lengths \( l : E(G) \to \mathbb{R}_{\geq 0} \)
- a family \( \mathcal{N} \) of nets (terminal pairs) with demands \( w : \mathcal{N} \to \mathbb{N} \)

Task: Find a flow \( f_N \) of value \( w(N) \) for each net \( N \in \mathcal{N} \) such that

\[
\sum_{N \in \mathcal{N}} f_N(e) \leq u(e) \quad \text{for } e \in E(G),
\]

and

\[
\sum_{N \in \mathcal{N}} \sum_{e \in E(G)} l(e)f_N(e) \quad \text{is minimum.}
\]
Special case: multicommodity flow

Instance:

- an undirected graph $G$ with capacities $u : E(G) \rightarrow \mathbb{N}$ and lengths $l : E(G) \rightarrow \mathbb{R}_{\geq 0}$
- a family $\mathcal{N}$ of nets (terminal pairs) with demands $w : \mathcal{N} \rightarrow \mathbb{N}$

Task: Find a flow $f_N$ of value $w(N)$ for each net $N \in \mathcal{N}$ such that

$$\sum_{N \in \mathcal{N}} \frac{f_N(e)}{u(e)} \leq 1 \quad \text{for } e \in E(G),$$

and

$$\sum_{N \in \mathcal{N}} \frac{1}{L} \left( \sum_{e \in E(G)} l(e) f_N(e) \right) \leq 1.$$ 

for some suitable number $L$ (e.g., found by binary search).
Special case: multicommodity flow

Instance:
- an undirected graph $G$ with capacities $u : E(G) \rightarrow \mathbb{N}$ and lengths $l : E(G) \rightarrow \mathbb{R}_{\geq 0}$
- a family $\mathcal{N}$ of nets (terminal pairs) with demands $w : \mathcal{N} \rightarrow \mathbb{N}$

Task: Find a flow $f_N$ of value $w(N)$ for each net $N \in \mathcal{N}$ such that

$$\sum_{N \in \mathcal{N}} \frac{f_N(e)}{u(e)} \leq 1 \quad \text{for } e \in E(G),$$

and

$$\sum_{N \in \mathcal{N}} \frac{1}{L} \left( \sum_{e \in E(G)} l(e)f_N(e) \right) \leq 1.$$
More general: application to global routing

Given a global routing graph (3D grid with millions of vertices).

- **Customers** = nets (sets of pins; roughly: sets of vertices)
- **Resources** = edge capacities, power consumption, wiring yield loss, timing constraints, ...
- Objective function is viewed as an additional resource
- **Block** = (convex hull of) set of Steiner trees for a net, with space consumption for each edge
- Resource consumption is a nonlinear but convex function for wiring yield loss, timing, power consumption
- **Block solver** = approximation algorithm for the Steiner tree problem in the global routing graph (with edge weights)
The min-max resource sharing problem once again

Instance

- finite sets $\mathcal{R}$ of resources and $\mathcal{C}$ of customers
- for each $c \in \mathcal{C}$:
  - a convex set $\mathcal{B}_c$ of feasible solutions (a block) and
  - a convex resource consumption function $g_c : \mathcal{B}_c \rightarrow \mathbb{R}_{\geq 0}$

given by an oracle function $f_c : \mathbb{R}_{\geq 0}^\mathcal{R} \rightarrow \mathcal{B}_c$ with

$$y^\top g_c(f_c(y)) \leq \sigma \inf_{b \in \mathcal{B}_c} y^\top g_c(b)$$

for all $y \in \mathbb{R}_{\geq 0}^\mathcal{R}$ and some $\sigma \geq 1$ (a block solver).

Task

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

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

A block solver is an oracle function $f_c : \mathbb{R}^\mathcal{R}_{\geq 0} \rightarrow \mathcal{B}_c$ with

$$y^\top g_c(f_c(y)) \leq \sigma \text{opt}_c(y)$$

for all $y \in \mathbb{R}^\mathcal{R}_{\geq 0}$ and some given constant $\sigma \geq 1$, where

$$\text{opt}_c(y) := \inf_{b \in \mathcal{B}_c} y^\top g_c(b)$$
Block solvers

A block solver is an oracle function $f_c : \mathbb{R}_{\geq 0}^\mathcal{R} \rightarrow \mathcal{B}_c$ with

$$y^\top g_c(f_c(y)) \leq \sigma \text{opt}_c(y)$$

for all $y \in \mathbb{R}_{\geq 0}^\mathcal{R}$ and some given constant $\sigma \geq 1$, where

$$\text{opt}_c(y) := \inf_{b \in \mathcal{B}_c} y^\top g_c(b)$$

A block solver is called

- **exact** if $\sigma = 1$
- **strong** if $\sigma > 1$ can be chosen arbitrary small
- **weak** otherwise
Block solvers

A block solver is an oracle function \( f_c : \mathbb{R}^{\mathcal{R}_{\geq 0}} \rightarrow \mathcal{B}_c \) with

\[
y^\top g_c(f_c(y)) \leq \sigma \text{opt}_c(y)
\]

for all \( y \in \mathbb{R}^{\mathcal{R}_{\geq 0}} \) and some given constant \( \sigma \geq 1 \), where

\[
\text{opt}_c(y) := \inf_{b \in \mathcal{B}_c} y^\top g_c(b)
\]

A block solver is called

- **exact** if \( \sigma = 1 \)
- **strong** if \( \sigma > 1 \) can be chosen arbitrary small
- **weak** otherwise

A block solver is called

- **bounded** if it can also optimize over

\[
\{ b \in \mathcal{B}_c : g_c(b) \leq \mu \mathbb{1} \}
\]

for any given \( \mu > 0 \) (\( c \in \mathcal{C} \)).
- **unbounded** otherwise
# Summary of theoretical results, previous work

## min-max resource sharing

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Block solver</th>
<th># oracle calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grigoriadis, Khachiyan [1994]</td>
<td>strong, bounded</td>
<td>$\tilde{O}(\omega^{-2}(</td>
</tr>
<tr>
<td>Grigoriadis, Khachiyan [1994]</td>
<td>strong, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Grigoriadis, Khachiyan [1996]</td>
<td>strong, unbounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Jansen, Zhang [2008]</td>
<td>weak, unbounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Müller, Radke, Vygen [2011]</td>
<td>weak, unbounded</td>
<td>$\tilde{O}(\omega^{-2}(</td>
</tr>
<tr>
<td>Müller, Radke, Vygen [2011]</td>
<td>weak, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
</tbody>
</table>

This is faster **and** more general than all previous algorithms.

Algorithms compute a $\sigma(1 + \omega)$-approximate solution for any $\omega > 0$. Running times dominated by # oracle calls. Logarithmic terms and dependency on $\sigma$ omitted.
### Summary of theoretical results, previous work

<table>
<thead>
<tr>
<th>min-max resource sharing</th>
<th>block solver</th>
<th># oracle calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grigoriadis, Khachiyan [1994]</td>
<td>strong, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Grigoriadis, Khachiyan [1994]</td>
<td>strong, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Grigoriadis, Khachiyan [1996]</td>
<td>strong, unbounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Jansen, Zhang [2008]</td>
<td>weak, unbounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Müller, Radke, Vygen [2011]</td>
<td>weak, unbounded</td>
<td>$\tilde{O}(\omega^{-2}(</td>
</tr>
<tr>
<td>Müller, Radke, Vygen [2011]</td>
<td>weak, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
</tbody>
</table>

This is faster **and** more general than all previous algorithms.

<table>
<thead>
<tr>
<th>fractional packing (all $g_c$ linear)</th>
<th>block solver</th>
<th>running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Villavicencia, Grigoriadis [1996]</td>
<td>strong, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Garg, Könemann [2007]</td>
<td>exact, unbounded</td>
<td>$\tilde{O}(\omega^{-2}(</td>
</tr>
</tbody>
</table>

Algorithms compute a $\sigma(1 + \omega)$-approximate solution for any $\omega > 0$.

Running times dominated by # oracle calls.

Logarithmic terms and dependency on $\sigma$ omitted.

Bienstock and Iyengar [2004] reduced dependency on $\omega$ to $\tilde{O}(\omega^{-1})$ for the linear case with explicitly given matrix, but otherwise slower.
### Summary of theoretical results, previous work

<table>
<thead>
<tr>
<th>min-max resource sharing</th>
<th>block solver</th>
<th># oracle calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grigoriadis, Khachiyan [1994]</td>
<td>strong, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Grigoriadis, Khachiyan [1994]</td>
<td>strong, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Grigoriadis, Khachiyan [1996]</td>
<td>strong, unbounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Jansen, Zhang [2008]</td>
<td>weak, unbounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Müller, Radke, Vygen [2011]</td>
<td>weak, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
</tbody>
</table>

This is faster and more general than all previous algorithms.

<table>
<thead>
<tr>
<th>fractional packing (all $g_c$ linear)</th>
<th>block solver</th>
<th>running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Villavicencio, Grigoriadis [1996]</td>
<td>strong, bounded</td>
<td>$\tilde{O}(\omega^{-2}</td>
</tr>
<tr>
<td>Garg, Könemann [2007]</td>
<td>exact, unbounded</td>
<td>$\tilde{O}(\omega^{-2}(</td>
</tr>
</tbody>
</table>

Algorithms compute a $\sigma(1 + \omega)$-approximate solution for any $\omega > 0$.
Running times dominated by # oracle calls.
Logarithmic terms and dependency on $\sigma$ omitted.
Bienstock and Iyengar [2004] reduced dependency on $\omega$ to $\tilde{O}(\omega^{-1})$ for the linear case with explicitly given matrix, but otherwise slower.
Weak duality

Let

$$\lambda^* := \inf \left\{ \max_{r \in \mathbb{R}} \sum_{c \in C} (g_c(b_c))_r : b_c \in B_c(c \in C) \right\}$$

(the "optimum congestion").

Lemma (Weak duality)

Let $y \in \mathbb{R}_{\geq 0}^R$ be some cost vector with $1^\top y \neq 0$. Then

$$\frac{\sum_{c \in C} opt_c(y)}{1^\top y} \leq \lambda^*.$$
Weak duality
Let
\[
\lambda^* := \inf \left\{ \max_{r \in \mathbb{R}} \sum_{c \in C} (g_c(b_c))_r : b_c \in B_c(c \in C) \right\}
\]
(the “optimum congestion”).

Lemma (Weak duality)
Let \( y \in \mathbb{R}_{\geq 0}^R \) be some cost vector with \( 1^\top y \neq 0 \). Then
\[
\frac{\sum_{c \in C} \text{opt}_c(y)}{1^\top y} \leq \lambda^*.
\]

Proof
Let \((b_c \in B_c)_{c \in C}\) be a solution with congestion \( \leq (1 + \delta)\lambda^* \). Then
\[
\sum_{c \in C} \text{opt}_c(y) \leq y^\top \sum_{c \in C} g_c(b_c) = y^\top \sum_{c \in C} g_c(b_c) \leq \frac{y^\top (1 + \delta)\lambda^*}{1^\top y}
\]
\[
= (1 + \delta)\lambda^* \quad \square
\]
Bounding $\lambda^*$

Lemma (Weak duality)

Let $y \in \mathbb{R}_{\geq 0}^R$ be some cost vector with $1^\top y \neq 0$. Then

$$\sum_{c \in C} \text{opt}_c(y) \leq \lambda^*.$$

Corollary

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

$$\lambda_{ub} |\mathbb{R}| \sigma \leq \sum_{r \in \mathbb{R}} \sum_{c \in C} (g_c(b_c)) r |\mathbb{R}| \sigma \leq \sum_{c \in C} \text{opt}_c(1^\top) |\mathbb{R}| \leq \lambda^* \leq \lambda_{ub}.$$

$\Box$
Bounding $\lambda^*$

**Lemma (Weak duality)**

Let $y \in \mathbb{R}_{\geq 0}^R$ be some cost vector with $1^\top y \neq 0$. Then

$$\sum_{c \in C} \text{opt}_c(y) \leq \lambda^*.$$

**Corollary**

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

$$\frac{\lambda^{ub}}{|\mathcal{R}| \sigma} \leq \frac{\sum_{r \in \mathcal{R}} \sum_{c \in C} (g_c(b_c))_r}{|\mathcal{R}| \sigma} \leq \frac{\sum_{c \in C} \text{opt}_c(1)}{|\mathcal{R}|} \leq \lambda^* \leq \lambda^{ub}.$$
Scaling

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

1. Set $j := 0$.
2. Scale $g(j) c(b) := g_c(b) 2^j \lambda_{ub}$. Note that $\lambda^*(j) \leq 1$.
3. Find a solution with congestion $\lambda(j) \leq \left(\sigma + \frac{1}{4}\right) \lambda^*(j) + \frac{1}{4}$.
4. If $\lambda(j) \leq \frac{1}{2}$, then increment $j$ and go to 2.
5. Now $\sigma \leq \lambda^*(j) \leq 1$.
6. Find a solution with congestion $\lambda(j) \leq \sigma \left(1 + \omega \frac{2}{\sigma}\right) \lambda^*(j) + \omega$ (hence at most $\sigma \left(1 + \omega\right)$ 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 $\sigma \left(1 + \delta\right) \lambda^* + \frac{\delta}{2}$ in $O\left(\left(\frac{\delta}{\delta'}\right)^{-1} \left(|C| + |R|\right) \theta \log |R|\right)$ time, where $\theta$ is the time for an oracle call.
Scaling

We know \( \frac{\lambda_{ub}}{\|R\|\sigma} \leq \lambda^* \leq \lambda_{ub} \).

1. Set \( j := 0 \).
2. Scale \( g_c^{(j)}(b) := g_c(b) \frac{2^j}{\lambda_{ub}} \). Note that \( \lambda^*(j) \leq 1 \).
Scaling

We know \( \frac{\lambda^{ub}}{|R|\sigma} \leq \lambda^* \leq \lambda^{ub} \).

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

We know \( \frac{\lambda^{ub}}{|R|} \sigma \leq \lambda^* \leq \lambda^{ub} \).

1. Set \( j := 0 \).
2. Scale \( g_c^{(j)}(b) := g_c(b) \frac{2^j}{\lambda^{ub}} \). Note that \( \lambda^*(j) \leq 1 \).
3. Find a solution with congestion \( \lambda^{(j)} \leq (\sigma + \frac{1}{4})\lambda^*(j) + \frac{1}{4} \).
4. If \( \lambda^{(j)} \leq \frac{1}{2} \), then increment \( j \) and go to 2.
We know $\frac{\lambda_{ub}}{|R|\sigma} \leq \lambda^* \leq \lambda_{ub}$.

1. Set $j := 0$.
2. Scale $g_c^{(j)}(b) := g_c(b) \frac{2^j}{\lambda_{ub}}$. Note that $\lambda^*(j) \leq 1$.
3. Find a solution with congestion $\lambda(j) \leq (\sigma + \frac{1}{4})\lambda^*(j) + \frac{1}{4}$.
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 $O\left(\left(\frac{\delta}{\delta'}\right)^{-1} (|C| + |R|) \theta \log |R|\right)$ time, where $\theta$ is the time for an oracle call.
Scaling

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

1. Set \( j := 0 \).
2. Scale \( g_c^{(j)}(b) := g_c(b) \frac{2^j}{\lambda_{ub}} \). Note that \( \lambda^*(j) \leq 1 \).
3. Find a solution with congestion \( \lambda(j) \leq (\sigma + \frac{1}{4}) \lambda^*(j) + \frac{1}{4} \).
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 \( \lambda(j) \leq \sigma(1 + \omega) \lambda^*(j) + \frac{\omega}{10} \) (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

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

time, where \( \theta \) is the time for an oracle call.
**Core algorithm**

**Input:** An instance of the min-max resource sharing problem.

**Output:** A convex combination of vectors in $\mathcal{B}_c$ for each $c \in \mathcal{C}$.

\[
t := \left\lceil \frac{3\sigma \ln |\mathcal{R}|}{\delta \delta'} \right\rceil, \quad \epsilon := \frac{\delta}{3\sigma}.
\]

$\alpha_r := 0, \quad y_r := 1$ for each $r \in \mathcal{R}$.

$x_{c,b} := 0, \quad X_c := 0$ 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:

(call oracle)

\[
b := f_c(y), \quad a := g_c(b).
\]

(record solution)

$\xi := \min\{p - X_c, 1/\max\{a_r : r \in \mathcal{R}\}\}$.

(update resource consumption)

$x_{c,b} := x_{c,b} + \xi, \quad X_c := X_c + \xi$.

(update prices)

$\alpha := \alpha + \xi a$.

(normalize)

For each $r \in \mathcal{R}$ with $a_r \neq 0$ do:

$y_r := e^{\epsilon \alpha r}$.

$x_{c,b} := \frac{1}{t} x_{c,b}$ for each $c \in \mathcal{C}$ and $b \in \mathcal{B}_c$. 
Proof of performance guarantee (sketch)

Lemma

Let \((x, y)\) be the output of the algorithm, and let

\[
\lambda_r := \sum_{c \in C} \left( g_c \left( \sum_{b \in B_c} x_{c,b} b \right) \right)_r
\]

and \(\lambda := \max_{r \in R} \lambda_r\). Then

\[
\lambda \leq \frac{1}{\epsilon t} \ln(1^\top y).
\]
Proof of performance guarantee (sketch)

**Lemma**

Let \((x, y)\) be the output of the algorithm, and let

\[
\lambda_r := \sum_{c \in C} \left( g_c \left( \sum_{b \in B_c} x_{c,b} b \right) \right)_r
\]

and \(\lambda := \max_{r \in \mathcal{R}} \lambda_r\). Then

\[
\lambda \leq \frac{1}{\epsilon t} \ln(1^\top y).
\]

**Proof:** Since the functions \(g_c\) are convex, we have for \(r \in \mathcal{R}\):

\[
\lambda_r \leq \sum_{c \in C} \sum_{b \in B_c} x_{c,b}(g_c(b))_r = \frac{\alpha_r}{t} = \frac{1}{\epsilon t} \ln (e^{\epsilon \alpha_r}) = \frac{1}{\epsilon t} \ln y_r \leq \frac{1}{\epsilon t} \ln(1^\top y).
\]
Proof of performance guarantee (sketch)

**Lemma (Main Lemma)**

Let $\delta, \delta' > 0$. Suppose that $\lambda^* \leq 1$.

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 $\lambda^* \leq 1$.
Then the algorithm computes a solution with congestion at most

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

Sketch of proof:

- Congestion is at most $\frac{1}{\epsilon t} \ln(\mathbf{1}^\top \mathbf{y}^{(t)})$.

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

Let $\delta, \delta' > 0$. Suppose that $\lambda^* \leq 1$.

Then the algorithm computes a solution with congestion at most

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

Sketch of proof:

- Congestion is at most $\frac{1}{\epsilon t} \ln (\mathbf{1}^\top y^{(t)})$.
- Initially, we have $\mathbf{1}^\top y^{(0)} = |\mathcal{R}|$.

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 $\lambda^* \leq 1$.
Then the algorithm computes a solution with congestion at most

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

Sketch of proof:

- Congestion is at most $\frac{1}{\epsilon t} \ln(1^t y^{(t)})$.
- Initially, we have $1^t y^{(0)} = |\mathcal{R}|$.
- Short calculation yields
  $$1^t y^{(p)} \leq 1^t y^{(p-1)} + \epsilon' \sum_{c \in \mathcal{C}} \text{opt}_c(y^{(p)}),$$
  where $y^{(i)}$ is the price vector at the end of the $i$-th phase and $\epsilon' := (e^\epsilon - 1)\sigma$. 
Proof of performance guarantee (sketch)

We had $\mathbf{1}^\top y^{(p)} \leq \mathbf{1}^\top y^{(p-1)} + \epsilon' \sum_{c \in C} \text{opt}_c(y^{(p)})$. 

Together with $\lambda \leq \frac{1}{\epsilon t \ln(\mathbf{1}^\top y(t))}$, this proves the claim. □
Proof of performance guarantee (sketch)

We had $\mathbf{1}^\top y^{(p)} \leq \mathbf{1}^\top y^{(p-1)} + \epsilon' \sum_{c \in C} \text{opt}_c(y^{(p)})$.

By weak duality, $\epsilon' \sum_{c \in C} \text{opt}_c(y^{(p)}) \leq \epsilon' \lambda^* < 1$, and we get

$$
\mathbf{1}^\top y^{(p)} \leq \frac{1}{1 - \epsilon' \lambda^*} \mathbf{1}^\top y^{(p-1)}
$$
Proof of performance guarantee (sketch)

We had $\mathbf{1}^\top y^{(p)} \leq \mathbf{1}^\top y^{(p-1)} + \epsilon' \sum_{c \in C} \text{opt}_c(y^{(p)})$.

By weak duality, $\epsilon' \sum_{c \in C} \text{opt}_c(y^{(p)}) \leq \epsilon' \lambda^* < 1$, and we get

$$\mathbf{1}^\top y^{(p)} \leq \frac{1}{1 - \epsilon' \lambda^*} \mathbf{1}^\top y^{(p-1)}$$

and thus

$$\mathbf{1}^\top y^{(t)} \leq \frac{|\mathcal{R}|}{(1 - \epsilon' \lambda^*)^t} = |\mathcal{R}| \left(1 + \frac{\epsilon' \lambda^*}{1 - \epsilon' \lambda^*}\right)^t \leq |\mathcal{R}| e^{t \epsilon' \lambda^*/(1 - \epsilon' \lambda^*)}.$$
Proof of performance guarantee (sketch)

We had $1^\top y^{(p)} \leq 1^\top y^{(p-1)} + \epsilon' \sum_{c \in C} \text{opt}_c(y^{(p)})$.

By weak duality, $\epsilon' \sum_{c \in C} \text{opt}_c(y^{(p)}) \leq \epsilon' \lambda^* < 1$, and we get

$$1^\top y^{(p)} \leq \frac{1}{1 - \epsilon' \lambda^*} 1^\top y^{(p-1)}$$

and thus

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

Together with $\lambda \leq \frac{1}{\epsilon t} \ln(1^\top y^{(t)})$, this proves the claim. \qed
Number of oracle calls

After each oracle call

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

Hence the number of oracle calls is

\[
O \left( (\delta \delta')^{-1} (|C| + |\mathcal{R}|) \log |\mathcal{R}| \right).
\]
Number of oracle calls

After each oracle call

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

Hence the number of oracle calls is

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

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

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

time, where \( \theta \) 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.

Faster and more general than all previous algorithms!

Extensions for practical application:

▶ Most oracle calls not necessary; reuse previous result if still good enough.
▶ Speed-up heuristics, fast approximate block solvers
▶ Parallelization (without loss of theoretical guarantees)
▶ Randomized rounding to extreme points of the blocks
▶ Re-choose where rounding violates constraints
Main result

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

Faster and more general than all previous algorithms!
Main result

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

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td>We can compute a $\sigma(1 + \omega)$-approximate solution in $O((\omega^{-2} + \log \log</td>
</tr>
</tbody>
</table>

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
- Speed-up heuristics, fast approximate block solvers
- Parallelization (without loss of theoretical guarantees)
- Randomized rounding to extreme points of the blocks
- Re-choose where rounding violates constraints
The algorithm in practice

- In practice, results are much better than theoretical performance guarantees. Usually 10–100 iterations suffice.
- Only few upper bounds are violated; these are corrected easily by *ripup-and-reroute*.
- Detailed routing can realize the solution well, due to excellent capacity estimations.
- Small integrality gap and approximate dual solution implies that an infeasibility proof can be found for most infeasible instances.
## Running time in practice: parallelization

<table>
<thead>
<tr>
<th>chip</th>
<th>#nets (K)</th>
<th>#res. (K)</th>
<th>Intel Xeon 2.93 GHz</th>
<th>AMD Opteron 2.7 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 thr.</td>
<td>16 threads</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8 threads</td>
<td></td>
</tr>
<tr>
<td>Lucius</td>
<td>78</td>
<td>337</td>
<td>0:00:50</td>
<td>0:01:08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:00:07 (6.7x)</td>
<td>0:00:06 (11.0x)</td>
</tr>
<tr>
<td>Sigurd</td>
<td>515</td>
<td>848</td>
<td>0:21:43</td>
<td>0:30:34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:02:49 (7.7x)</td>
<td>0:02:29 (12.3x)</td>
</tr>
<tr>
<td>Claus</td>
<td>709</td>
<td>1 590</td>
<td>0:17:14</td>
<td>0:25:31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:02:13 (7.8x)</td>
<td>0:01:44 (14.7x)</td>
</tr>
<tr>
<td>Dorothea</td>
<td>679</td>
<td>3 647</td>
<td>0:23:17</td>
<td>0:34:44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:03:11 (7.3x)</td>
<td>0:02:32 (13.7x)</td>
</tr>
<tr>
<td>Laci</td>
<td>1 300</td>
<td>32 652</td>
<td>0:36:35</td>
<td>0:55:04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:04:57 (7.4x)</td>
<td>0:03:58 (13.9x)</td>
</tr>
<tr>
<td>Vašek</td>
<td>434</td>
<td>36 519</td>
<td>7:16:06</td>
<td>11:23:42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:57:39 (7.6x)</td>
<td>0:44:08 (15.5x)</td>
</tr>
<tr>
<td>Camilla</td>
<td>3 582</td>
<td>10 255</td>
<td>0:56:50</td>
<td>1:21:15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:07:10 (7.9x)</td>
<td>0:05:34 (14.6x)</td>
</tr>
<tr>
<td>Tomoko</td>
<td>5 340</td>
<td>23 680</td>
<td>0:12:29</td>
<td>0:17:44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:02:19 (5.4x)</td>
<td>0:02:01 (8.8x)</td>
</tr>
<tr>
<td>Andre</td>
<td>7 039</td>
<td>23 923</td>
<td>1:04:37</td>
<td>1:35:17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:09:14 (7.0x)</td>
<td>0:07:30 (12.7x)</td>
</tr>
<tr>
<td>Ludwig</td>
<td>7 677</td>
<td>23 577</td>
<td>7:58:22</td>
<td>10:25:43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0:57:28 (8.3x)</td>
<td>0:44:21 (14.1x)</td>
</tr>
</tbody>
</table>
Global routing result for the telecommunication chip
## Near optimality

<table>
<thead>
<tr>
<th>chip (# nets)</th>
<th>$\epsilon = \frac{100}{t}$</th>
<th>$\lambda_{\text{fract}}$</th>
<th>$\lambda_{\text{lb}}$</th>
<th>$\lambda_{\text{rounded}}$</th>
<th>$\lambda_{\text{final}}$</th>
<th>% gap (fract.)</th>
<th>% gap (final)</th>
<th>running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rose (594 084)</td>
<td>0.5</td>
<td>0.950</td>
<td>0.941</td>
<td>2.000</td>
<td>1.000</td>
<td>1.91</td>
<td>1.96</td>
<td>0:01:45</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.950</td>
<td>0.939</td>
<td>2.000</td>
<td>1.000</td>
<td>1.97</td>
<td>2.01</td>
<td>0:01:08</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>0.949</td>
<td>0.930</td>
<td>2.000</td>
<td>1.000</td>
<td>2.41</td>
<td>2.43</td>
<td>0:00:28</td>
</tr>
<tr>
<td>Georg (783 685)</td>
<td>0.5</td>
<td>0.950</td>
<td>0.942</td>
<td>2.000</td>
<td>1.000</td>
<td>1.53</td>
<td>1.56</td>
<td>0:02:38</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.950</td>
<td>0.940</td>
<td>3.000</td>
<td>1.000</td>
<td>1.61</td>
<td>1.63</td>
<td>0:01:40</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>0.949</td>
<td>0.934</td>
<td>2.000</td>
<td>1.000</td>
<td>1.90</td>
<td>1.91</td>
<td>0:00:49</td>
</tr>
<tr>
<td>Camilla (3 582 559)</td>
<td>0.5</td>
<td>0.950</td>
<td>0.937</td>
<td>4.000</td>
<td>1.000</td>
<td>2.52</td>
<td>2.56</td>
<td>0:51:09</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.950</td>
<td>0.933</td>
<td>4.000</td>
<td>1.000</td>
<td>2.59</td>
<td>2.61</td>
<td>0:33:05</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>0.950</td>
<td>0.918</td>
<td>3.000</td>
<td>1.000</td>
<td>3.34</td>
<td>3.33</td>
<td>0:11:23</td>
</tr>
<tr>
<td>Tomoko (5 340 088)</td>
<td>0.5</td>
<td>1.629</td>
<td>1.449</td>
<td>3.000</td>
<td>1.600</td>
<td>1.12</td>
<td>1.14</td>
<td>0:23:43</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>1.625</td>
<td>1.449</td>
<td>3.000</td>
<td>1.600</td>
<td>1.18</td>
<td>1.20</td>
<td>0:15:21</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>1.667</td>
<td>1.369</td>
<td>2.500</td>
<td>1.600</td>
<td>1.36</td>
<td>1.36</td>
<td>0:07:09</td>
</tr>
<tr>
<td>Andre (7 039 094)</td>
<td>0.5</td>
<td>0.950</td>
<td>0.938</td>
<td>3.000</td>
<td>1.000</td>
<td>2.25</td>
<td>2.38</td>
<td>1:18:54</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>0.950</td>
<td>0.935</td>
<td>3.000</td>
<td>1.000</td>
<td>2.38</td>
<td>2.48</td>
<td>0:50:43</td>
</tr>
<tr>
<td></td>
<td>5.0</td>
<td>0.950</td>
<td>0.923</td>
<td>3.000</td>
<td>1.000</td>
<td>2.92</td>
<td>2.96</td>
<td>0:17:32</td>
</tr>
<tr>
<td>chip (# nets)</td>
<td>optimization objective</td>
<td>ratio to lower bound</td>
<td>relative obj. value</td>
<td>running time (h:mm:ss)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------</td>
<td>----------------------</td>
<td>---------------------</td>
<td>------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Henrik (440 915)</td>
<td>wire length</td>
<td>1.0064</td>
<td>1.000</td>
<td>1.052</td>
<td>1.055</td>
<td>0:01:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>critical area</td>
<td>1.0100</td>
<td>1.030</td>
<td>1.000</td>
<td>1.043</td>
<td>0:04:47</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>power</td>
<td>1.0300</td>
<td>1.146</td>
<td>1.172</td>
<td>1.000</td>
<td>0:07:36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Milena (1 312 592)</td>
<td>wire length</td>
<td>1.0112</td>
<td>1.000</td>
<td>1.103</td>
<td>1.136</td>
<td>0:06:05</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>critical area</td>
<td>1.0134</td>
<td>1.074</td>
<td>1.000</td>
<td>1.087</td>
<td>0:14:10</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>power</td>
<td>1.0298</td>
<td>1.285</td>
<td>1.301</td>
<td>1.000</td>
<td>0:18:08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guido (1 796 234)</td>
<td>wire length</td>
<td>1.0072</td>
<td>1.000</td>
<td>1.138</td>
<td>1.115</td>
<td>0:06:17</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>critical area</td>
<td>1.0149</td>
<td>1.071</td>
<td>1.000</td>
<td>1.100</td>
<td>0:26:28</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>power</td>
<td>1.0287</td>
<td>1.220</td>
<td>1.282</td>
<td>1.000</td>
<td>0:30:39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thilo (2 772 390)</td>
<td>wire length</td>
<td>1.0158</td>
<td>1.000</td>
<td>1.110</td>
<td>1.098</td>
<td>0:23:18</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>critical area</td>
<td>1.0237</td>
<td>1.055</td>
<td>1.000</td>
<td>1.076</td>
<td>0:50:50</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>power</td>
<td>1.0322</td>
<td>1.241</td>
<td>1.292</td>
<td>1.000</td>
<td>0:54:39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Martina (3 783 704)</td>
<td>wire length</td>
<td>1.0097</td>
<td>1.000</td>
<td>1.151</td>
<td>1.134</td>
<td>0:20:14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>critical area</td>
<td>1.0166</td>
<td>1.072</td>
<td>1.000</td>
<td>1.101</td>
<td>1:03:32</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>power</td>
<td>1.0278</td>
<td>1.232</td>
<td>1.297</td>
<td>1.000</td>
<td>1:10:18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Routing buffered wires (and displacing buffers)

- treat repeaters as part of the wires
- keep distances between repeaters (almost) fixed
- additional resources modeling available placement area
- work in progress
- preliminary results:

<table>
<thead>
<tr>
<th>chip</th>
<th>repeaters</th>
<th>max. rel. cong.</th>
<th>edge overload</th>
<th>wiring length</th>
<th>worst late slack</th>
<th>runtime (h:mm:ss)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charly</td>
<td>fixed</td>
<td>1.167</td>
<td>40</td>
<td>3.64</td>
<td>0.123</td>
<td>0:01:27</td>
</tr>
<tr>
<td></td>
<td>floating</td>
<td>0.994</td>
<td>0</td>
<td>3.64</td>
<td>0.135</td>
<td>0:04:13</td>
</tr>
<tr>
<td>Helen</td>
<td>fixed</td>
<td>5.194</td>
<td>23247</td>
<td>31.99</td>
<td>-1.864</td>
<td>0:10:17</td>
</tr>
<tr>
<td></td>
<td>floating</td>
<td>1.000</td>
<td>0</td>
<td>31.09</td>
<td>-1.868</td>
<td>1:02:01</td>
</tr>
<tr>
<td>Tara</td>
<td>fixed</td>
<td>1.120</td>
<td>74</td>
<td>69.23</td>
<td>0.247</td>
<td>0:11:52</td>
</tr>
<tr>
<td></td>
<td>floating</td>
<td>1.000</td>
<td>0</td>
<td>67.95</td>
<td>0.093</td>
<td>1:34:19</td>
</tr>
<tr>
<td>Chiara</td>
<td>fixed</td>
<td>1.370</td>
<td>30539</td>
<td>10.66</td>
<td>-300.9</td>
<td>0:04:28</td>
</tr>
<tr>
<td></td>
<td>floating</td>
<td>0.974</td>
<td>0</td>
<td>10.62</td>
<td>-298.3</td>
<td>0:51:05</td>
</tr>
</tbody>
</table>
Simplified design flow (old)

1. RTL Description
2. Logic Synthesis
3. Global Placement
4. Timing Optimization
5. Clocktree Generation
6. Detailed Placement
7. Global Routing
8. Detailed Routing
9. Final Checks
10. Production
Simplified design flow (new)

RTL Description

Logic Synthesis

Global Placement

Global Routing

Timing Optimization

Clocktree Generation

Detailed Placement

Global Routing

Detailed Routing

Final Checks

Production
Congestion map of a difficult instance
Comparison to an industrial router

<table>
<thead>
<tr>
<th>chip</th>
<th># nets (K)</th>
<th>wire length (m)</th>
<th># vias (K)</th>
<th># scenic nets</th>
<th>runtime (h:mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bonn</td>
<td>industry</td>
<td>Bonn</td>
<td>industry</td>
<td>Bonn industry</td>
</tr>
<tr>
<td>Guochuan</td>
<td>124</td>
<td>3.72</td>
<td>3.78</td>
<td>889</td>
<td>0</td>
</tr>
<tr>
<td>Ulrike</td>
<td>128</td>
<td>4.48</td>
<td>4.67</td>
<td>951</td>
<td>49</td>
</tr>
<tr>
<td>Ute</td>
<td>190</td>
<td>5.42</td>
<td>5.50</td>
<td>1465</td>
<td>1</td>
</tr>
<tr>
<td>Maxim</td>
<td>222</td>
<td>6.63</td>
<td>7.52</td>
<td>1885</td>
<td>205</td>
</tr>
<tr>
<td>Romy</td>
<td>472</td>
<td>10.69</td>
<td>11.75</td>
<td>3267</td>
<td>686</td>
</tr>
<tr>
<td>Helen</td>
<td>431</td>
<td>35.97</td>
<td>36.56</td>
<td>3188</td>
<td>6</td>
</tr>
<tr>
<td>Milena</td>
<td>1314</td>
<td>47.10</td>
<td>51.25</td>
<td>11144</td>
<td>859</td>
</tr>
<tr>
<td>total</td>
<td>2881</td>
<td>114.01</td>
<td>121.04</td>
<td>22789</td>
<td>1806</td>
</tr>
</tbody>
</table>

Thanks to Michael Gester, Dirk Müller, Karsten Muuss, Tim Nieberg, Christian Panten, Sven Peyer, Christian Schulte, Gustavo Tellez, ...
Conclusion: mathematics leads to better chips!