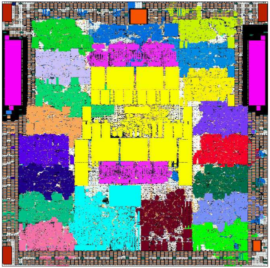
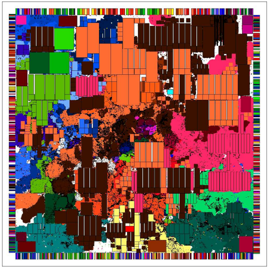
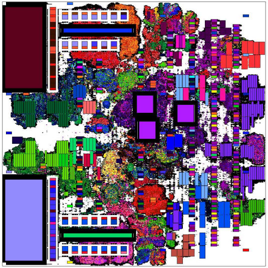
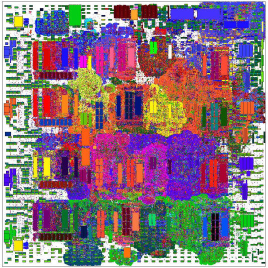
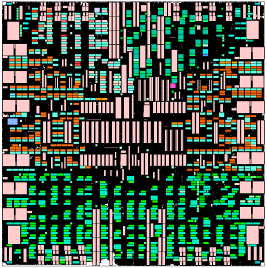
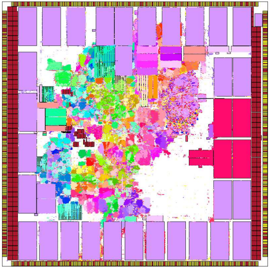
Combinatorial Optimization in Chip Design

Jens Vygen

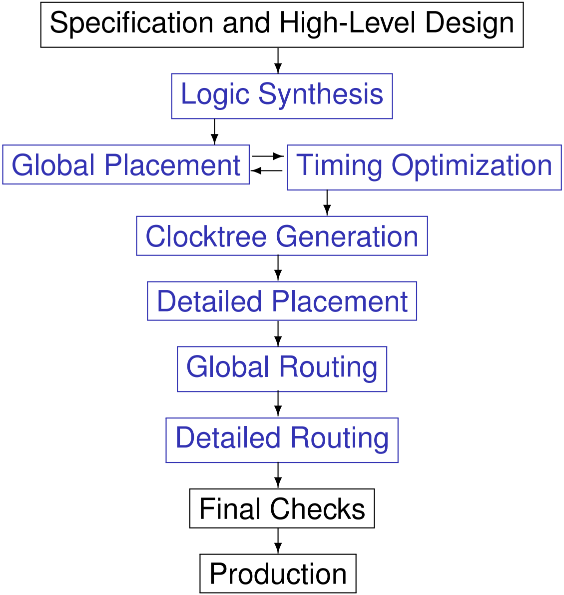
University of Bonn

EURO 2009

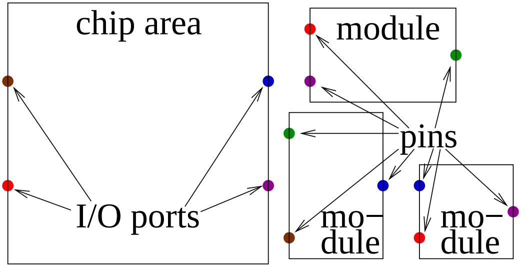
Some recent chips

Simplified design flow

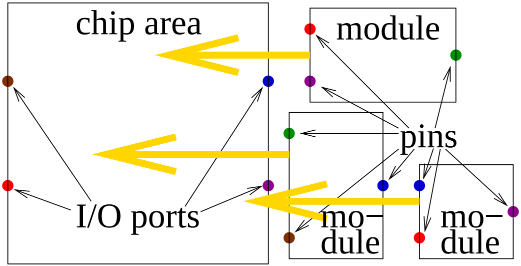


Some key problems



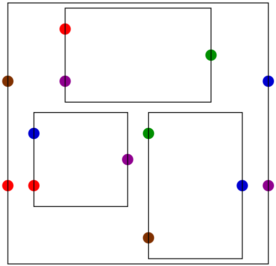
- Place:
- Route:
- Buffer:

Some key problems



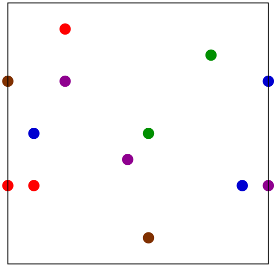
- Place: Given a chip area and rectangular modules with pins, and a partition of all pins into nets, place the modules without overlaps such that the total estimated wirelength is minimum.
- Route:
- Buffer:

Some key problems



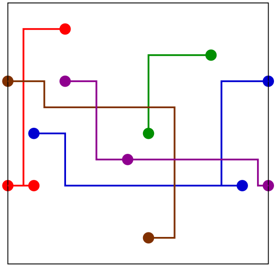
- Place: Given a chip area and rectangular modules with pins, and a partition of all pins into nets, place the modules without overlaps such that the total estimated wirelength is minimum.
- Route:
- Buffer:

Some key problems



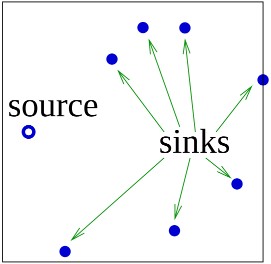
- Place: Given a chip area and rectangular modules with pins, and a partition of all pins into nets, place the modules without overlaps such that the total estimated wirelength is minimum.
- Route: Connect the pins of each net by wires of a given width and vias (connecting adjacent routing planes), such that wires of different nets have at least a given minimum distance.
- Buffer:

Some key problems



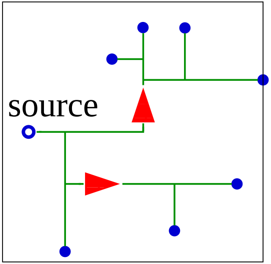
- Place: Given a chip area and rectangular modules with pins, and a partition of all pins into nets, place the modules without overlaps such that the total estimated wirelength is minimum.
- Route: Connect the pins of each net by wires of a given width and vias (connecting adjacent routing planes), such that wires of different nets have at least a given minimum distance.
- Buffer:

Some key problems



- Place: Given a chip area and rectangular modules with pins, and a partition of all pins into nets, place the modules without overlaps such that the total estimated wirelength is minimum.
- Route: Connect the pins of each net by wires of a given width and vias (connecting adjacent routing planes), such that wires of different nets have at least a given minimum distance.
- Buffer: Given a source and a set of sinks, distribute the signal from the source to the sinks by wiring and buffers such that the latest arrival time at a sink is as early as possible.

Some key problems



- Place: Given a chip area and rectangular modules with pins, and a partition of all pins into nets, place the modules without overlaps such that the total estimated wirelength is minimum.
- Route: Connect the pins of each net by wires of a given width and vias (connecting adjacent routing planes), such that wires of different nets have at least a given minimum distance.
- Buffer: Given a source and a set of sinks, distribute the signal from the source to the sinks by wiring and buffers such that the latest arrival time at a sink is as early as possible.

Challenges

- Very difficult combinatorial problems
  (quadratic assignment problem, packing Steiner trees, ...)
- Huge instance sizes
  (millions of modules and nets, graphs with billions of vertices)
- New technology generations every two years
  (resulting in new problems, constraints and objectives)

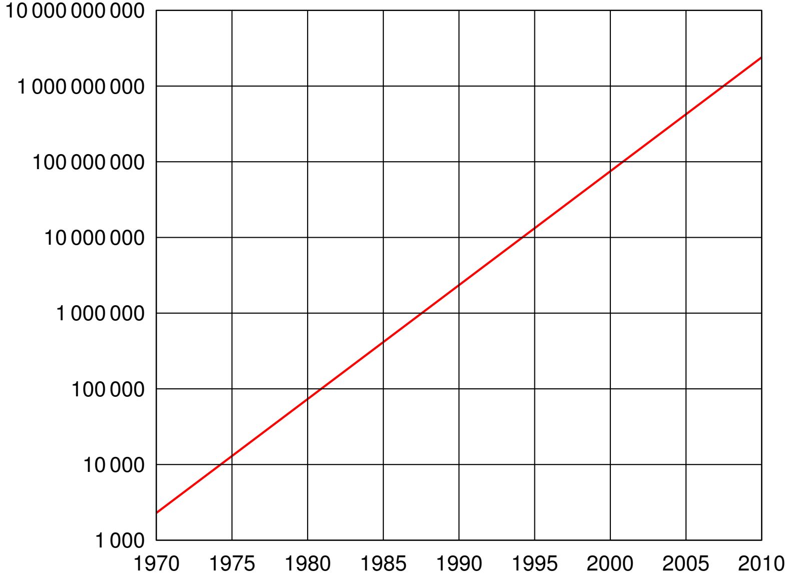
Challenges

- Very difficult combinatorial problems
  (quadratic assignment problem, packing Steiner trees, ...)
- Huge instance sizes
  (millions of modules and nets, graphs with billions of vertices)
- New technology generations every two years
  (resulting in new problems, constraints and objectives)

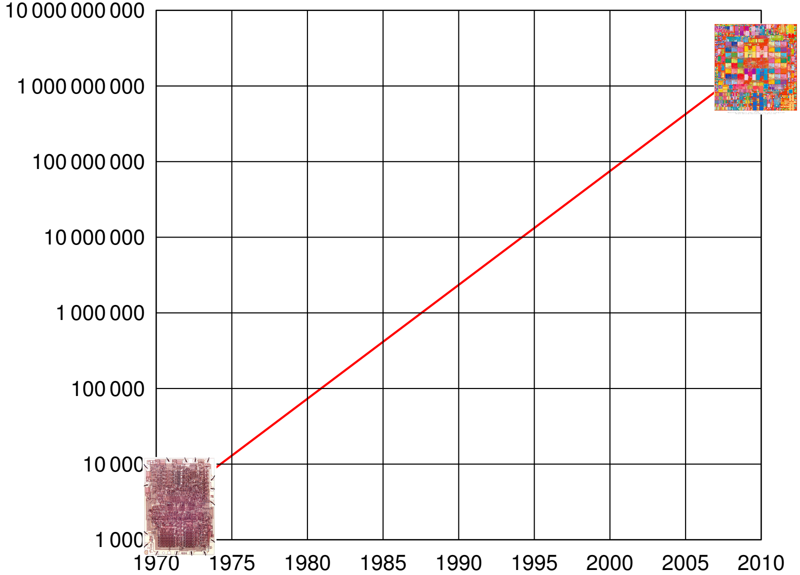
We need:

- New theory
  (existing results and algorithms often insufficient)
- Very fast algorithms and efficient implementations
  (to achieve acceptable turn-around time)
- Fast track from new theory to production-ready software
  (months instead of years)

Moore's law: number of transistors per chip



Moore's law: number of transistors per chip

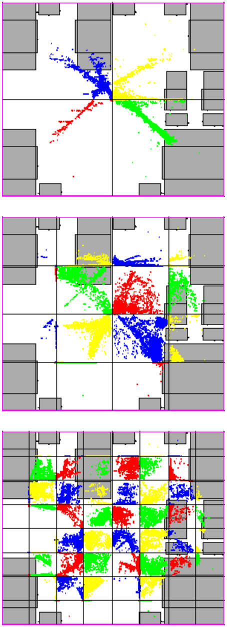


Three examples

- Placement and partitioning
- Routing and resource sharing
- Buffering and sink clustering

Global placement by successive partitioning

All state-of-the-art placement tools use (variants of) quadratic placement and/or partitioning for global placement, followed by legalization
(Brenner, Vygen [2004,2009])



Basic idea:
Successively partition the chip area into smaller and smaller regions and assign the set of modules to these regions

Minimize netlength in quadratic placement
Minimize movement in partitioning

A single partitioning step ("multisection")

Instance: A set X of modules, a size $\text{size}(x)$ for each $x \in X$, and a set R of (sub)regions, a capacity $\text{cap}(r)$ for each $r \in R$.

Task: Find an assignment $f : X \to R$ meeting the capacity constraints

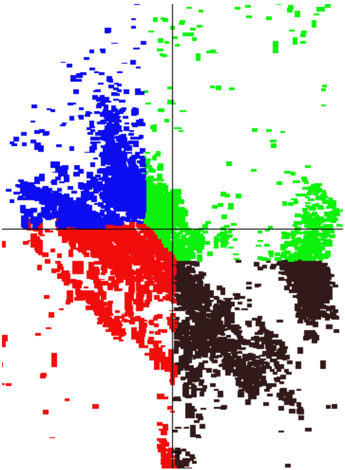
$$\sum_{x \in X : f(x) = r} \text{size}(x) \leq \text{cap}(r) \text{ for all } r \in R$$

such that the total movement

$$\sum_{x \in X} d(x, f(x))$$

is minimum.

Here d denotes, e.g., the ℓ_1-distance.



A single partitioning step ("multisection")

Instance: A set X of modules, a size $\text{size}(x)$ for each $x \in X$, and a set R of (sub)regions, a capacity $\text{cap}(r)$ for each $r \in R$.

Task: Find an assignment $f : X \to R$ meeting the capacity constraints

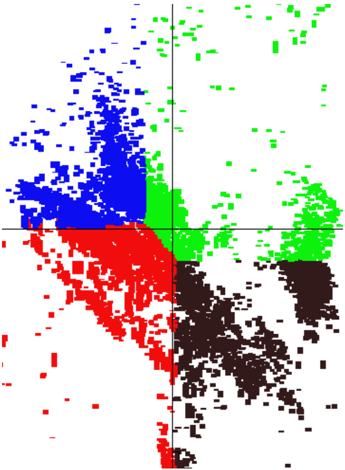
$$\sum_{x \in X : f(x) = r} \text{size}(x) \leq \text{cap}(r) \text{ for all } r \in R$$

such that the total movement

$$\sum_{x \in X} d(x, f(x))$$

is minimum.

Here d denotes, e.g., the ℓ_1-distance.



But: this problem is *NP*-hard (includes PARTITION).

Fractional relaxation

Find $g : X \times R \to \mathbb{R}_+$

with

$$\sum_{r \in R} g(x, r) = \text{size}(x) \text{ for all } x \in X$$

and

$$\sum_{x \in X} g(x, r) \leq \text{cap}(r) \text{ for all } r \in R$$

such that

$$\sum_{x \in X} \sum_{r \in R} g(x, r) d(x, r)$$

is minimum.

Note: $|R| \ll |X|$

Solving the fractional relaxation is sufficient

Theorem (Vygen [2005])

Given any optimum solution to the fractional relaxation, we can compute another optimum solution in $O(|X||R|^2)$ time that is integral except for $|R| - 1$ modules.

Solving the fractional relaxation is sufficient

Theorem (Vygen [2005])

Given any optimum solution to the fractional relaxation, we can compute another optimum solution in $O(|X||R|^2)$ time that is integral except for $|R| - 1$ modules.

Proof:

Define $V(G) := R = \{1, \ldots, |R|\}$ and

$$E(G) := \big\{\{r, r'\} : x \in X,\ g(x, r) > 0,\ g(x, r') > 0,\ g(x, r'') = 0 \text{ for } r'' \in \{1, \ldots, \max\{r, r'\}\} \setminus \{r, r'\}\big\}.$$

Solving the fractional relaxation is sufficient

Theorem (Vygen [2005])

Given any optimum solution to the fractional relaxation, we can compute another optimum solution in $O(|X||R|^2)$ time that is integral except for $|R|-1$ modules.

Proof:

Define $V(G) := R = \{1, \ldots, |R|\}$ and

$$E(G) := \big\{\{r, r'\} : x \in X,\ g(x, r) > 0,\ g(x, r') > 0,\ g(x, r'') = 0 \text{ for } r'' \in \{1, \ldots, \max\{r, r'\}\} \setminus \{r, r'\}\big\}.$$

While G contains a cycle, consider g' and g'' that result from g by moving the same amount of flow around the cycle in each direction.

Solving the fractional relaxation is sufficient

Theorem (Vygen [2005])

Given any optimum solution to the fractional relaxation, we can compute another optimum solution in $O(|X||R|^2)$ *time that is integral except for* $|R| - 1$ *modules.*

Proof:

Define $V(G) := R = \{1, \ldots, |R|\}$ and

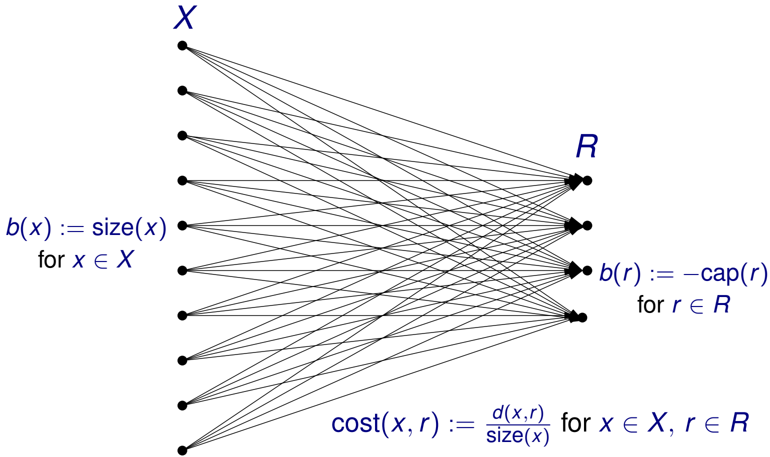
$$E(G) := \big\{\{r, r'\} : x \in X,\ g(x, r) > 0,\ g(x, r') > 0,\\ g(x, r'') = 0 \text{ for } r'' \in \{1, \ldots, \max\{r, r'\}\} \setminus \{r, r'\}\big\}.$$

While G contains a cycle, consider g' and g'' that result from g by moving the same amount of flow around the cycle in each direction.

Both g' and g'' must be optimum solutions.
The number of fractions decreases. Iterate. □

Reformulation as Hitchcock transportation problem

Let G be the digraph with $V(G) := X \,\dot{\cup}\, R$ and $E(G) := X \times R$. Let



Task: Find an uncapacitated b-flow in G of minimum cost.

Algorithms for the Hitchcock problem

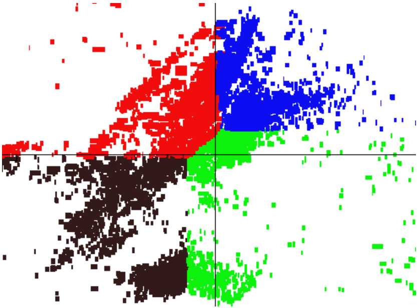
Let $n := |X|$ and $k := |R|$. We assume $n \geq k$.

- $O(n \log n(n \log n + kn))$ general transshipment algorithm: Orlin [1993]
- $O(n\, f(k))$ with exponential functions f, inefficient already for very small k: Dyer [1984], Zemel [1984], Tokuyama, Nakano [1991], Meggido, Tamir [1993], Matsui [1993]
- $O(nk^2 \log^2 n)$: Tokuyama, Nakano [1992, 1995]

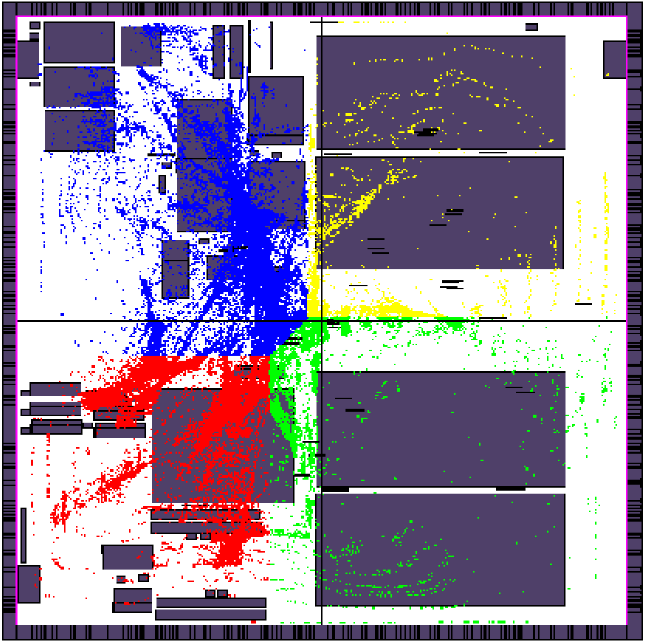
Algorithms for the Hitchcock problem

Let $n := |X|$ and $k := |R|$. We assume $n \geq k$.

- $O(n \log n(n \log n + kn))$ general transshipment algorithm: Orlin [1993]
- $O(n\, f(k))$ with exponential functions f, inefficient already for very small k: Dyer [1984], Zemel [1984], Tokuyama, Nakano [1991], Meggido, Tamir [1993], Matsui [1993]
- $O(nk^2 \log^2 n)$: Tokuyama, Nakano [1992, 1995]
- Structure theorem and very efficient $O(n)$-algorithm for $k = 4$ and $d = \ell_1$-distance (quadrisection): Vygen [2005]
- $O(nk^2(\log n + k \log k))$: Brenner [2008]



Quadrisection based on quadratic placement



General multisection algorithm

- Sort $X = \{x_1, \ldots, x_n\}$ such that $\text{size}(x_1) \geq \text{size}(x_2) \geq \cdots \geq \text{size}(x_n)$.
- Start with zero flow.
- **For** $i := 1$ **to** n **do**:

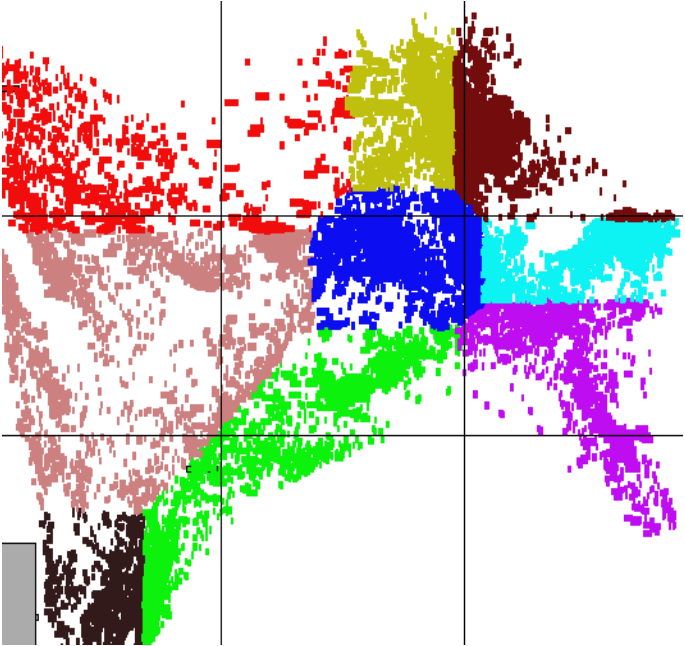
  augment flow by an optimum flow from x_i to R
  of value $\text{size}(x_i)$ in the residual graph
  transform flow to an almost integral one

- Key idea:

  In each iteration we have to consider only $O(k^2)$ arcs.
- Overall running time: $O(nk^2(\log n + k \log k))$

(Brenner [2008])

Multisection example

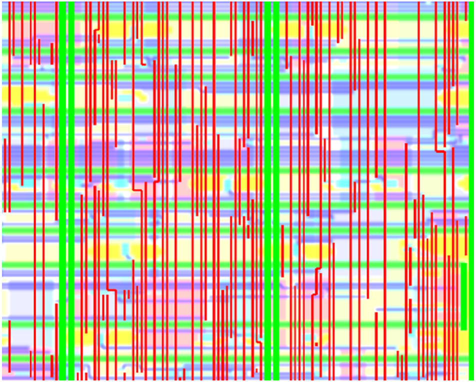


Three examples

- Placement and partitioning
- Routing and resource sharing
- Buffering and sink clustering

Global routing

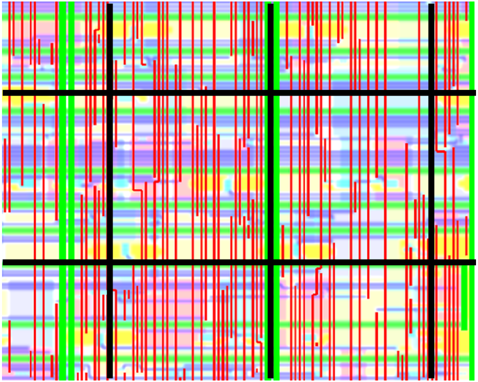
Due to its complexity and the huge instances, routing is split into global and detailed routing



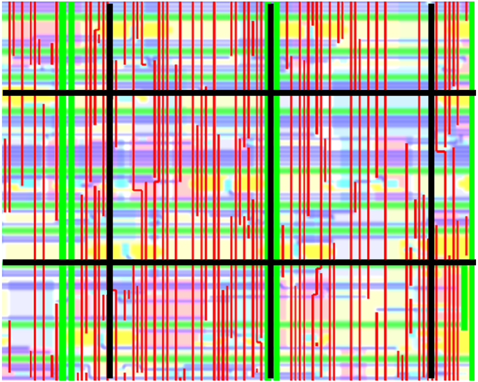
Global routing

Due to its complexity and the huge instances, routing is split into global and detailed routing

In each routing plane: contract regions of approximately 50x50 tracks to a single vertex



Global routing



Due to its complexity and the huge instances, routing is split into global and detailed routing

In each routing plane: contract regions of approximately 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 detailed routing area for each net
- Detailed routing computes the detailed wires in these areas by a very fast goal-oriented interval-labeling variant of Dijkstra's algorithm (Peyer, Rautenbach, Vygen [2009])

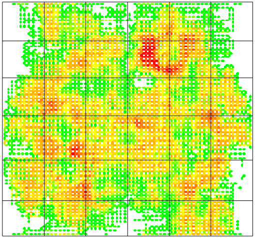
Global routing: classical problem formulation

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.



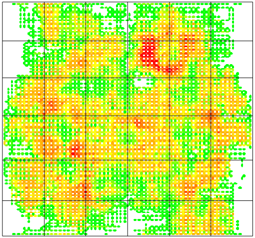
Global routing: classical problem formulation

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!

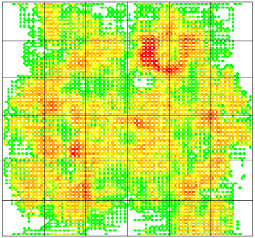
Global routing: classical problem formulation

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!

Timing, yield, power consumption, etc. ignored!

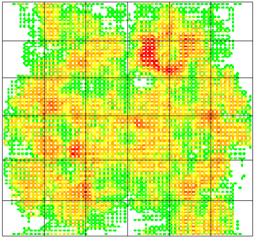
Global routing: classical problem formulation

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!

Timing, yield, power consumption, etc. ignored!

Special case of two-terminal nets: integer multi-commodity flows

Relaxation: fractional multi-commodity flows

- Can be solved by linear programming (but too slow)
- Combinatorial fully polynomial approximation schemes: Sharokhi, Matula [1990], Leighton, Makedon, Plotkin, Stein, Tardos, Tragoudas [1991], Plotkin, Shmoys, Tardos [1991], Radzik [1995], Young [1995], Grigoriadis, Khachiyan [1996], Garg, Könemann [1998], Fleischer [2000], Karakostas [2002]

Relaxation: fractional multi-commodity flows

- Can be solved by linear programming (but too slow)
- Combinatorial fully polynomial approximation schemes: Sharokhi, Matula [1990], Leighton, Makedon, Plotkin, Stein, Tardos, Tragoudas [1991], Plotkin, Shmoys, Tardos [1991], Radzik [1995], Young [1995], Grigoriadis, Khachiyan [1996], Garg, Könemann [1998], Fleischer [2000], Karakostas [2002]
- If edges have sufficient capacity, randomized rounding yields an integral solution violating capacity constraints only slightly (Raghavan, Thompson [1987,1991], Raghavan [1988])

Relaxation: fractional multi-commodity flows

- Can be solved by linear programming (but too slow)
- Combinatorial fully polynomial approximation schemes: Sharokhi, Matula [1990], Leighton, Makedon, Plotkin, Stein, Tardos, Tragoudas [1991], Plotkin, Shmoys, Tardos [1991], Radzik [1995], Young [1995], Grigoriadis, Khachiyan [1996], Garg, Könemann [1998], Fleischer [2000], Karakostas [2002]
- If edges have sufficient capacity, randomized rounding yields an integral solution violating capacity constraints only slightly (Raghavan, Thompson [1987,1991], Raghavan [1988])
- This can be applied to Steiner trees instead of paths, works efficiently for large global routing instances (Albrecht [2001])

Relaxation: fractional multi-commodity flows

- Can be solved by linear programming (but too slow)
- Combinatorial fully polynomial approximation schemes: Sharokhi, Matula [1990], Leighton, Makedon, Plotkin, Stein, Tardos, Tragoudas [1991], Plotkin, Shmoys, Tardos [1991], Radzik [1995], Young [1995], Grigoriadis, Khachiyan [1996], Garg, Könemann [1998], Fleischer [2000], Karakostas [2002]
- If edges have sufficient capacity, randomized rounding yields an integral solution violating capacity constraints only slightly (Raghavan, Thompson [1987,1991], Raghavan [1988])
- This can be applied to Steiner trees instead of paths, works efficiently for large global routing instances (Albrecht [2001])

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

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!)

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

General idea

Compute for each net n

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

General idea

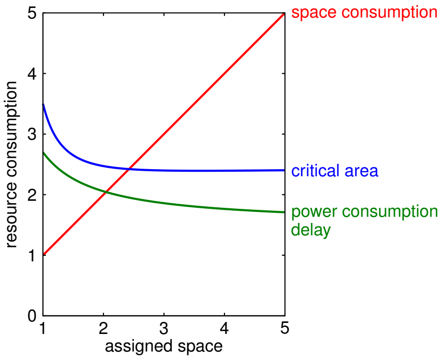
Compute for each net *n*

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

The contribution of (*n*, *e*) to

- power consumption
- wiring yield loss ("critical area")
- delay

depends on whether *e* is used and how much space is assigned. These functions are convex.



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** $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}}$

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** $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}}$

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** $g_c : \mathcal{B}_c \to \mathbb{R}_+^{\mathcal{R}}$
- given by an **oracle function** $f_c : \mathbb{R}_+^{\mathcal{R}} \to \mathcal{B}_c$ with

$$\omega^\top g_c(f_c(\omega)) \leq (1 + \epsilon_0) \inf_{b \in \mathcal{B}_c} \omega^\top g_c(b)$$

for all $\omega \in \mathbb{R}_+^{\mathcal{R}}$ and some $\epsilon_0 \in \mathbb{R}_+$ (a **block solver**).

Task

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

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

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 transformed into a constraint
- **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)

Algorithm

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

For at most $\lceil \log |\mathcal{R}| \log(1 + \epsilon_0) \rceil$ iterations **do**:

- Scale all resource consumptions and compute t.
- Initialize all resource prizes: $\omega_r := 1$ $(r \in \mathcal{R})$.
- **For** $p := 1$ **to** t **do**:
  - **For each** $c \in \mathcal{C}$:
    - Find an approximately cheapest solution $f_c(\omega)$.
    - Update prizes: ω_r depends exponentially on the total usage of $r \in \mathcal{R}$.
- Take the arithmetic mean of the t solutions.

Main result

Theorem (Müller, Vygen [2008])

Our algorithm computes a $(1 + \epsilon_0 + \epsilon)$*-approximate solution in* $O(|\mathcal{C}|\theta\rho(1 + \epsilon_0)^2 \log |\mathcal{R}|(\log |\mathcal{R}| + \epsilon^{-2}(1 + \epsilon_0)))$ *time, where* ρ *is the "width" (usually 1) and* θ *is the time for an oracle call, for any* $\epsilon > 0$.

Main result

Theorem (Müller, Vygen [2008])

Our algorithm computes a $(1 + \epsilon_0 + \epsilon)$*-approximate solution in* $O(|\mathcal{C}|\theta\rho(1 + \epsilon_0)^2 \log |\mathcal{R}|(\log |\mathcal{R}| + \epsilon^{-2}(1 + \epsilon_0)))$ *time, where* ρ *is the "width" (usually 1) and* θ *is the time for an oracle call, for any* $\epsilon > 0$*.*

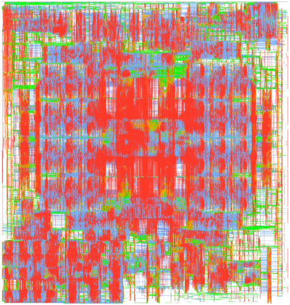
All previous algorithms (Grigoriadis, Khachiyan [1994,1996], Khandekar [2004], Jansen, Zhang [2008]) depend at least linearly on $|\mathcal{R}|$ or quadratically on $|\mathcal{C}|$!

Main result

Theorem (Müller, Vygen [2008])

Our algorithm computes a $(1+\epsilon_0+\epsilon)$*-approximate solution in* $O(|\mathcal{C}|\theta\rho(1+\epsilon_0)^2\log|\mathcal{R}|(\log|\mathcal{R}|+\epsilon^{-2}(1+\epsilon_0)))$ *time, where* ρ *is the "width" (usually 1) and* θ *is the time for an oracle call, for any* $\epsilon > 0$*.*

All previous algorithms (Grigoriadis, Khachiyan [1994,1996], Khandekar [2004], Jansen, Zhang [2008]) depend at least linearly on $|\mathcal{R}|$ or quadratically on $|\mathcal{C}|$!



Extensions for practical application:

- Most oracle calls not necessary;
  reuse previous result if still good enough.
  Use lower bounds to decide
- Speed-up heuristics
- Efficient parallelization
- Fast approximate block solvers

The algorithm in practice

- In practice, results are much better than theory guarantees. Usually 10–20 iterations suffice.
- Only few upper bounds are violated by randomized rounding; these are corrected locally by re-choose, rip-up and re-route.
- Detailed routing can realize the solution well, due to excellent capacity estimations.
- Small integrality gap and approximate dual solution implies an infeasibility proof for most infeasible instances.

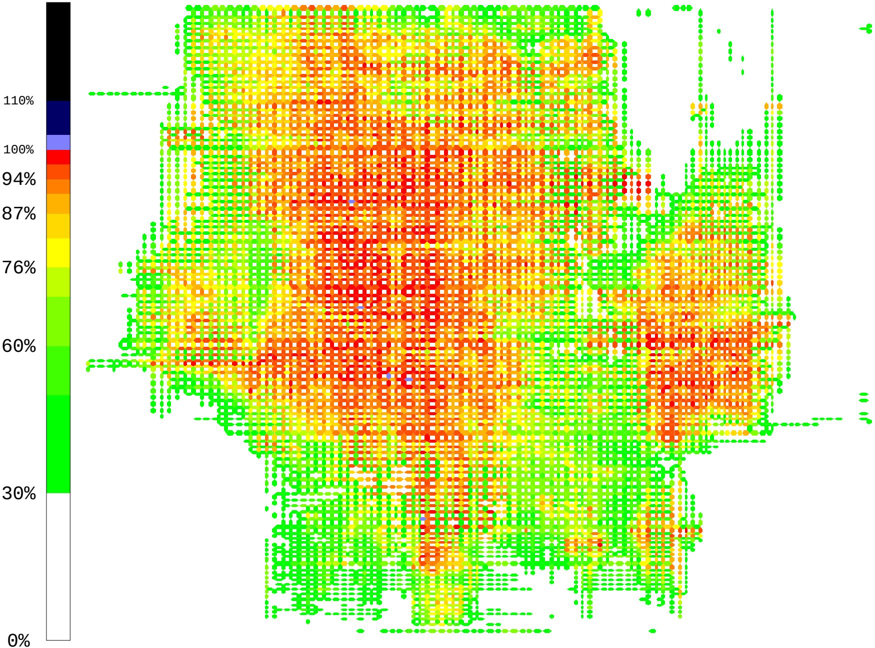
The algorithm in practice

- In practice, results are much better than theory guarantees. Usually 10–20 iterations suffice.
- Only few upper bounds are violated by randomized rounding; these are corrected locally by re-choose, rip-up and re-route.
- Detailed routing can realize the solution well, due to excellent capacity estimations.
- Small integrality gap and approximate dual solution implies an infeasibility proof for most infeasible instances.

Running times in practice (h:mm:ss):

Chip	$\lvert\mathcal{C}\rvert$	$\lvert\mathcal{R}\rvert$	1 thread	4 threads	8 threads
A	478 946	894 377	0:15:49	0:04:25	0:02:37
B	786 368	1 949 245	1:18:13	0:23:09	0:14:29
C	529 966	1 091 339	0:48:40	0:13:19	0:08:20
D	959 163	2 794 166	1:12:26	0:21:00	0:10:49
E	3 590 647	20 392 657	1:16:07	0:23:27	0:15:09
F	5 340 123	23 606 915	0:33:25	0:12:22	0:08:51
G	7 039 094	22 891 145	2:32:48	0:46:12	0:29:08

Congestion map of a difficult instance



Critical area after detailed routing

Critical area measures the expected percentage of manufactured chips that will *not* work due to opens or shorts

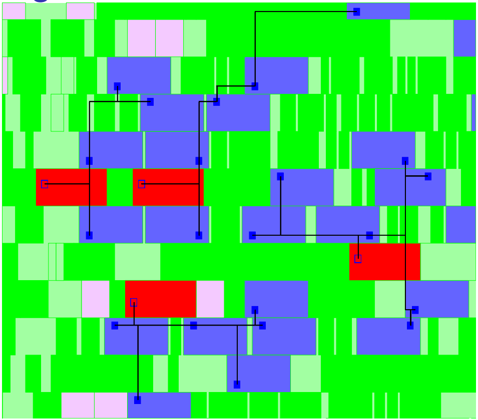
Chip	**# nets**	**old (netlength)**	**new (yield optimization)**	
Bill	11 287	0.00833	0.00376	(–54.9%)
Ingo	58 765	0.00505	0.00392	(–22.4%)
Paul	68 277	0.00568	0.00402	(–29.2%)
Lotti	132 986	0.00688	0.00575	(–16.4%)
Hanne	140 413	0.01543	0.01027	(–33.4%)
Elena	421 402	0.03314	0.02966	(–10.5%)
Edgar	772 245	0.10493	0.08586	(–18.2%)
Heidi	777 166	0.05804	0.04965	(–14.5%)
Garry	827 569	0.08017	0.06714	(–16.3%)
Monika	1 502 512	0.09505	0.08055	(–15.3%)

(Müller [2006])

Three examples

- Placement and partitioning
- Routing and resource sharing
- Buffering and sink clustering

Buffering and sink clustering



Problem: a signal must be distributed to a set of sinks.

If the number of sinks is large (as in clocktree design), the sink clustering problem is key

blue: sinks
red: facilities

Other important problems for distributing signals include

- topology generation: constructing short and fast Steiner trees
- buffering (dynamic programming)

(Bartoschek, Held, Rautenbach, Vygen [2006,2009])

Sink clustering

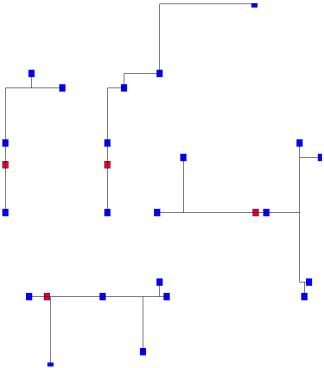
Instance:

- metric space (V, c),
- finite set $\mathcal{D} \subseteq V$ (of sinks),
- demands $d : \mathcal{D} \rightarrow \mathbb{R}_+$,
- facility opening cost $f \in \mathbb{R}_+$,
- capacity $u \in \mathbb{R}_+$.

Sink clustering

Instance:

- metric space (V, c),
- finite set $\mathcal{D} \subseteq V$ (of sinks),
- demands $d : \mathcal{D} \to \mathbb{R}_+$,
- facility opening cost $f \in \mathbb{R}_+$,
- capacity $u \in \mathbb{R}_+$.



Task:

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

$$c(E(T_i)) + d(D_i) \leq u$$

for $i = 1, \ldots, k$ such that

$$\sum_{i=1}^{k} c(E(T_i)) + kf$$

is minimum.

Approximability of sink clustering

Proposition

- *There is no* $(1.5 - \epsilon)$*-approximation algorithm (for any* $\epsilon > 0$*) unless* $P = NP$*.*
- *There is no* $(2 - \epsilon)$*-approximation algorithm (for any* $\epsilon > 0$*) for any class of metrics where the Steiner tree problem cannot be solved exactly in polynomial time.*

Approximability of sink clustering

Proposition

- *There is no* $(1.5 - \epsilon)$*-approximation algorithm (for any* $\epsilon > 0$*) unless* $P = NP$*.*
- *There is no* $(2 - \epsilon)$*-approximation algorithm (for any* $\epsilon > 0$*) for any class of metrics where the Steiner tree problem cannot be solved exactly in polynomial time.*

Theorem (Maßberg, Vygen [2008])

Let $n := |\mathcal{D}|$*. There is*

- *a polynomial-time* 4.099*-approximation and*
- *an* $O(n^2)$*-time* 5*-approximation.*

For the rectilinear plane there is

- *a polynomial-time* $(3 + \epsilon)$*-approximation for any* $\epsilon > 0$ *and*
- *an* $O(n \log n)$*-time* 4*-approximation.*

Lower bound: spanning forests

Let F_1 be a minimum spanning tree for $(\mathcal{D}, c)$.
Let $e_1, \ldots, e_{n-1}$ be the edges of F_1 so that $c(e_1) \geq \ldots \geq c(e_{n-1})$.
Set $F_k := F_{k-1} \setminus \{e_{k-1}\}$ for $k = 2, \ldots, n$.

Lower bound: spanning forests

Let F_1 be a minimum spanning tree for $(\mathcal{D}, c)$.
Let $e_1, \ldots, e_{n-1}$ be the edges of F_1 so that $c(e_1) \geq \ldots \geq c(e_{n-1})$.
Set $F_k := F_{k-1} \setminus \{e_{k-1}\}$ for $k = 2, \ldots, n$.

Lemma

F_k is a minimum weight spanning forest in $(\mathcal{D}, c)$ with exactly k connected components.

Lower bound: spanning forests

Let F_1 be a minimum spanning tree for $(\mathcal{D}, c)$.
Let $e_1, \ldots, e_{n-1}$ be the edges of F_1 so that $c(e_1) \geq \ldots \geq c(e_{n-1})$.
Set $F_k := F_{k-1} \setminus \{e_{k-1}\}$ for $k = 2, \ldots, n$.

Lemma

F_k is a minimum weight spanning forest in $(\mathcal{D}, c)$ with exactly k connected components.

Proof: By induction on k. Trivial for $k = 1$. Let $k > 1$.
Let F^* be a minimum weight k-spanning forest.
Let $e \in F_{k-1}$ such that $F^* \cup \{e\}$ is a forest (matroid property).

Lower bound: spanning forests

Let F_1 be a minimum spanning tree for $(\mathcal{D}, c)$.
Let $e_1, \ldots, e_{n-1}$ be the edges of F_1 so that $c(e_1) \geq \ldots \geq c(e_{n-1})$.
Set $F_k := F_{k-1} \setminus \{e_{k-1}\}$ for $k = 2, \ldots, n$.

Lemma

F_k is a minimum weight spanning forest in $(\mathcal{D}, c)$ with exactly k connected components.

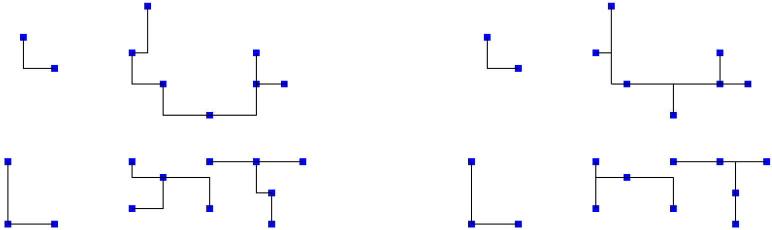
Proof: By induction on k. Trivial for $k = 1$. Let $k > 1$.
Let F^* be a minimum weight k-spanning forest.
Let $e \in F_{k-1}$ such that $F^* \cup \{e\}$ is a forest (matroid property).
Then

$$c(F_k) + c(e_{k-1}) = c(F_{k-1}) \leq c(F^*) + c(e) \leq c(F^*) + c(e_{k-1}).$$

□

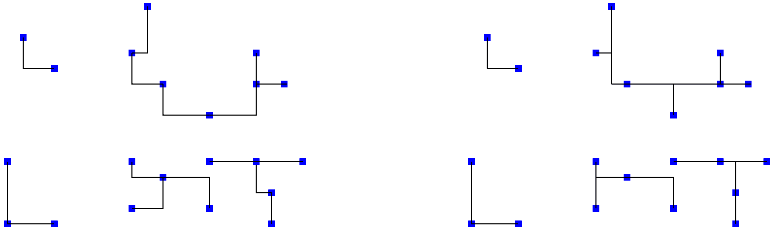
Lower bound: Steiner forests

A k-Steiner forest is a forest F with $\mathcal{D} \subseteq V(F)$ and exactly k connected components.



Lower bound: Steiner forests

A k-Steiner forest is a forest F with $\mathcal{D} \subseteq V(F)$ and exactly k connected components.



Lemma

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

Lower bound: number of facilities

Let t' be the smallest integer such that

$$\frac{1}{\alpha} c(F_{t'}) + d(\mathcal{D}) \leq t' \cdot u$$

Lemma

t' is a lower bound for the number of facilities of any solution. □

Lower bound: number of facilities

Let t' be the smallest integer such that

$$\frac{1}{\alpha}c(F_{t'}) + d(\mathcal{D}) \leq t' \cdot u$$

Lemma

t' is a lower bound for the number of facilities of any solution. □

Let t'' be an integer in $\{t', \ldots, n\}$ minimizing

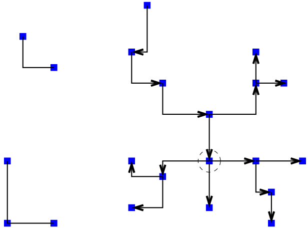
$$\frac{1}{\alpha}c(F_{t''}) + t'' \cdot f.$$

Theorem

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

Algorithm

1. Compute a minimum spanning tree on $(\mathcal{D}, c)$.
2. Compute t'' and spanning forest $F_{t''}$ as above.
3. Split up overloaded components by a bin packing algorithm.



It can be guaranteed that for each new component at least $\frac{u}{2}$ of the load will be removed from the initial forest.

Analysis of the algorithm

Recall: $\frac{1}{\alpha}c(F_{t''}) + t'' \cdot f$ is a lower bound for the optimum.

We set $L_r := \frac{1}{\alpha}c(F_{t''})$ and $L_f := t'' \cdot f$.

Analysis of the algorithm

Recall: $\frac{1}{\alpha}c(F_{t''}) + t'' \cdot f$ is a lower bound for the optimum.

We set $L_r := \frac{1}{\alpha}c(F_{t''})$ and $L_f := t'' \cdot f$.

Observe: $L_r + d(\mathcal{D}) \leq \frac{u}{f} L_f$.

Analysis of the algorithm

Recall: $\frac{1}{\alpha}c(F_{t''}) + t'' \cdot f$ is a lower bound for the optimum.

We set $L_r := \frac{1}{\alpha}c(F_{t''})$ and $L_f := t'' \cdot f$.

Observe: $L_r + d(\mathcal{D}) \leq \frac{u}{f} L_f$.

The cost of the final solution is at most

$$c(F_{t''}) + t''f + \frac{2}{u}\Big(c(F_{t''}) + d(\mathcal{D})\Big)f$$

Analysis of the algorithm

Recall: $\frac{1}{\alpha}c(F_{t''}) + t'' \cdot f$ is a lower bound for the optimum.

We set $L_r := \frac{1}{\alpha}c(F_{t''})$ and $L_f := t'' \cdot f$.

Observe: $L_r + d(\mathcal{D}) \leq \frac{u}{f}L_f$.

The cost of the final solution is at most

$$c(F_{t''}) + t''f + \frac{2}{u}\Big(c(F_{t''}) + d(\mathcal{D})\Big)f$$

$$= \alpha L_r + L_f + \frac{2f}{u}\left(\alpha L_r + d(\mathcal{D})\right)$$

Analysis of the algorithm

Recall: $\frac{1}{\alpha}c(F_{t''}) + t'' \cdot f$ is a lower bound for the optimum.

We set $L_r := \frac{1}{\alpha}c(F_{t''})$ and $L_f := t'' \cdot f$.

Observe: $L_r + d(\mathcal{D}) \leq \frac{u}{f} L_f$.

The cost of the final solution is at most

$$c(F_{t''}) + t''f + \frac{2}{u}\Big(c(F_{t''}) + d(\mathcal{D})\Big)f$$

$$= \alpha L_r + L_f + \frac{2f}{u}(\alpha L_r + d(\mathcal{D}))$$

$$\leq \alpha L_r + L_f + 2\alpha L_f$$

Theorem (Maßberg, Vygen [2008])

We have a $(2\alpha + 1)$*-approximation algorithm.* □

Computing the initial spanning tree dominates the running time.

Experimental results on real-world instances

instance	A	B	C	D	E	F
# sinks	3 675	17 140	45 606	54 831	109 224	119 461
MST length	13.72	60.35	134.24	183.37	260.36	314.48
t'	117	638	1 475	2 051	3 116	3 998
L_r	8.21	31.68	63.73	102.80	135.32	181.45
$L_r + L_f$	23.07	112.70	251.06	363.28	531.05	689.19
# facilities	161	947	2 171	2 922	4 156	5 525
service cost	12.08	54.23	101.57	159.93	234.34	279.93
total cost	32.52	174.50	377.29	531.03	762.15	981.61
gap (factor)	1.41	1.55	1.59	1.46	1.44	1.42

Experimental results on real-world instances

instance	A	B	C	D	E	F
# sinks	3 675	17 140	45 606	54 831	109 224	119 461
MST length	13.72	60.35	134.24	183.37	260.36	314.48
t'	117	638	1 475	2 051	3 116	3 998
L_r	8.21	31.68	63.73	102.80	135.32	181.45
$L_r + L_f$	23.07	112.70	251.06	363.28	531.05	689.19
# facilities	161	947	2 171	2 922	4 156	5 525
service cost	12.08	54.23	101.57	159.93	234.34	279.93
total cost	32.52	174.50	377.29	531.03	762.15	981.61
gap (factor)	1.41	1.55	1.59	1.46	1.44	1.42

Reduction of power consumption:

chip	Jens	Katrin	Bert	Alex
total # sinks	3 805	137 265	40 298	189 341
largest instance	375	119 461	16 260	35 305
power (W, heuristic)	0.100	0.329	0.306	2.097
power (W, new algorithm)	0.088	0.287	0.283	1.946
gain	−11.1%	−12.8%	−7.5%	−7.2%

Conclusion

We discussed three examples:

- Placement and partitioning
- Routing and resource sharing
- Buffering and sink clustering

These algorithms—and many others—are part of the BonnTools.

The BonnTools

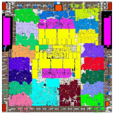
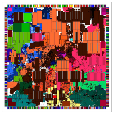
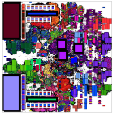
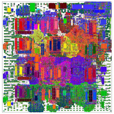
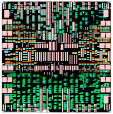
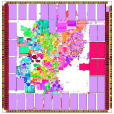
- developed by the University of Bonn
  (Research Institute for Discrete Mathematics)
- cover all major areas of layout and timing optimization,

The BonnTools

- developed by the University of Bonn
  (Research Institute for Discrete Mathematics)
- 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 in C and C++,

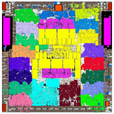
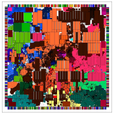
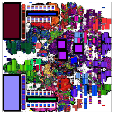
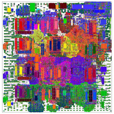
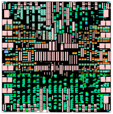
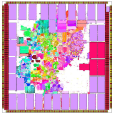
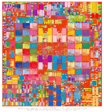
The BonnTools

- developed by the University of Bonn
  (Research Institute for Discrete Mathematics)
- 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 in C and 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.

The BonnTools

- developed by the University of Bonn
  (Research Institute for Discrete Mathematics)
- 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 in C and 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.

Thanks to all my colleagues and students!

Thank you!