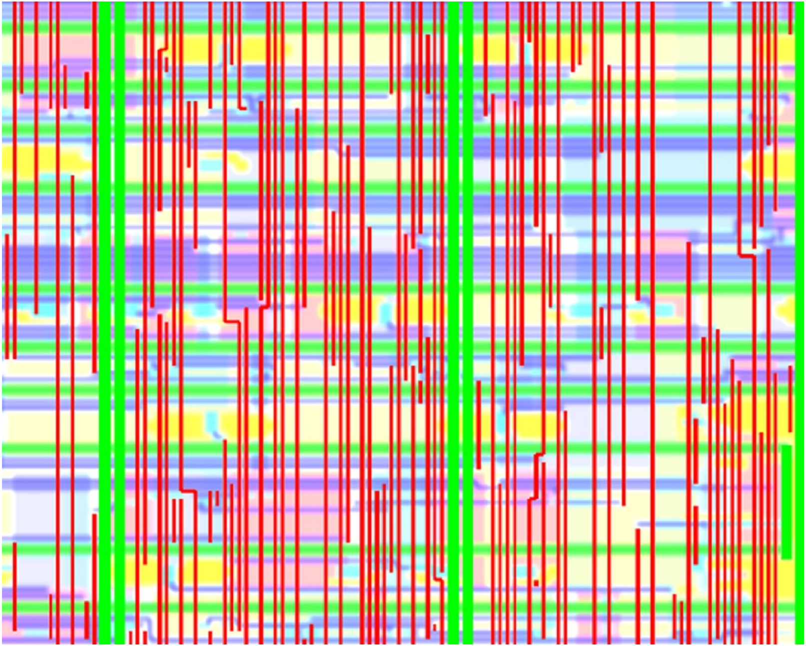
Detailed Routing

Jens Vygen

Hangzhou, March 2009

Detailed routing: example



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, each of which is a rectangle in a routing plane
- a set of blockage shapes
- rules that tell when two shapes are connected and when they are separated
- rules with forbidden patterns (for manufacturability)
- timing constraints, information on power, crosstalk, yield, ...

Task:

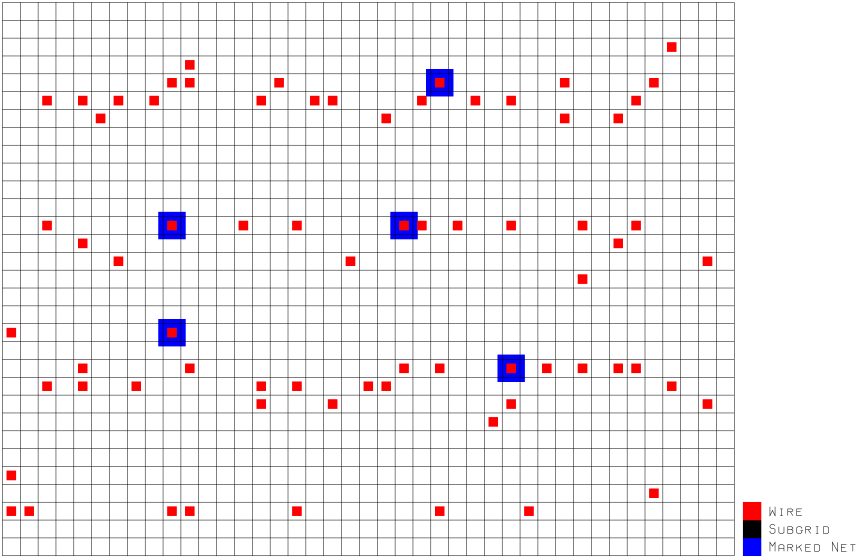
Compute a feasible routing, i.e. a set of wire shapes for each net, connecting the pins, and separate from blockages and shapes of other nets

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

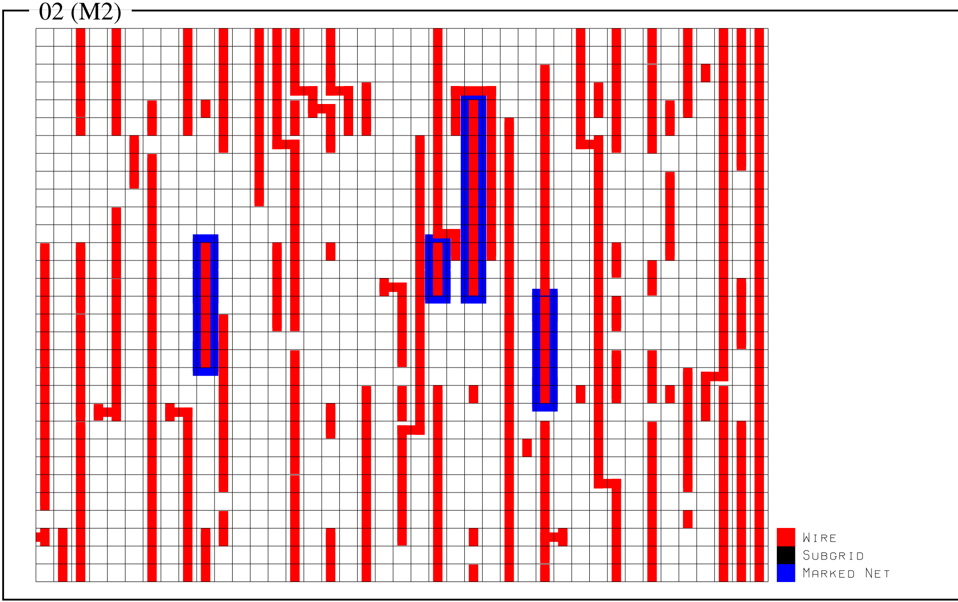
Detailed routing: example

Detailed routing: example

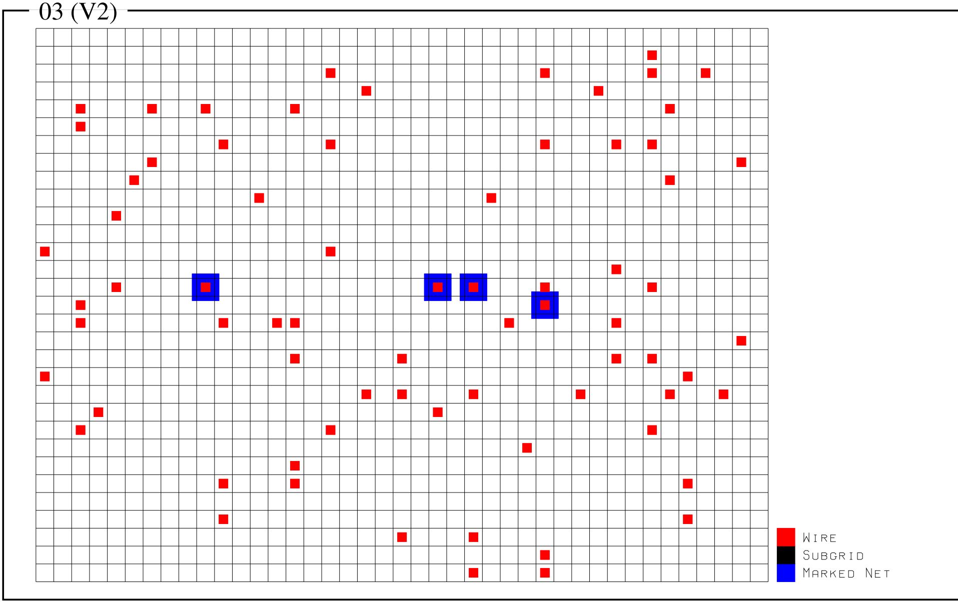
01 (V1)



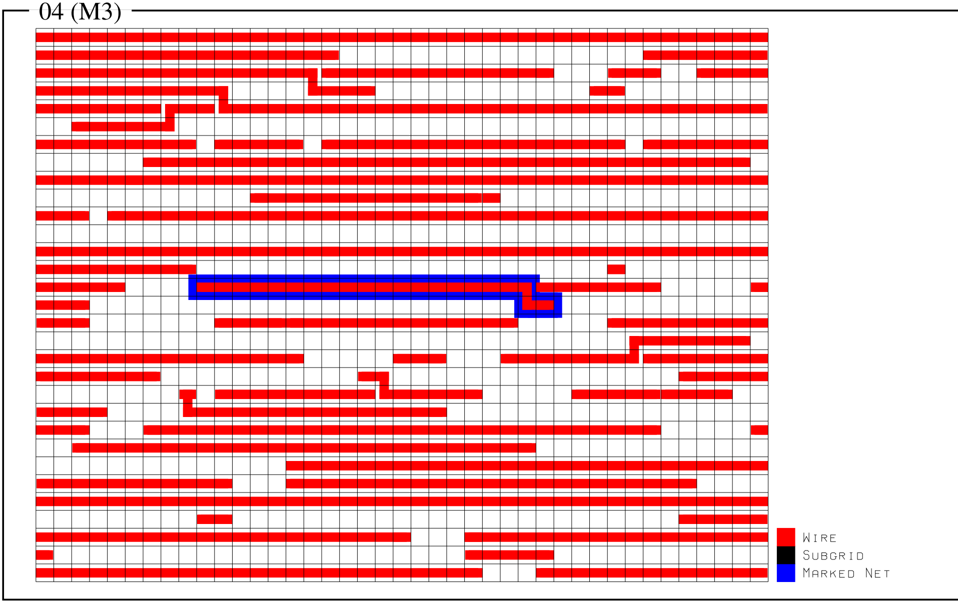
Detailed routing: example



Detailed routing: example



Detailed routing: example



Modelling the routing space by a graph

- Define parallel tracks for each plane, alternatingly horizontally and vertically.
- Distance of tracks is (at least) the minimum space required by a wire
- Via positions where tracks of adjacent planes meet
- Via positions induce vertices on both incident layers

Then a Steiner tree in this graph corresponds to a feasible routing, except that

- pin shapes may not contain any vertex (need special algorithms for local pin access)
- same-net errors may occur (but not often, can usually be repaired at the end)
- in some cases the only feasible routing may be globally off-track (but this is a rare exception)
- special care is needed for wider wires that occupy more than one track (but this can be done)

Routing: simplified view

Find vertex-disjoint Steiner trees connecting given terminal sets in this track graph.

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

$\longrightarrow$ Even linear-time algorithms are too slow!

How to cope with the instance sizes

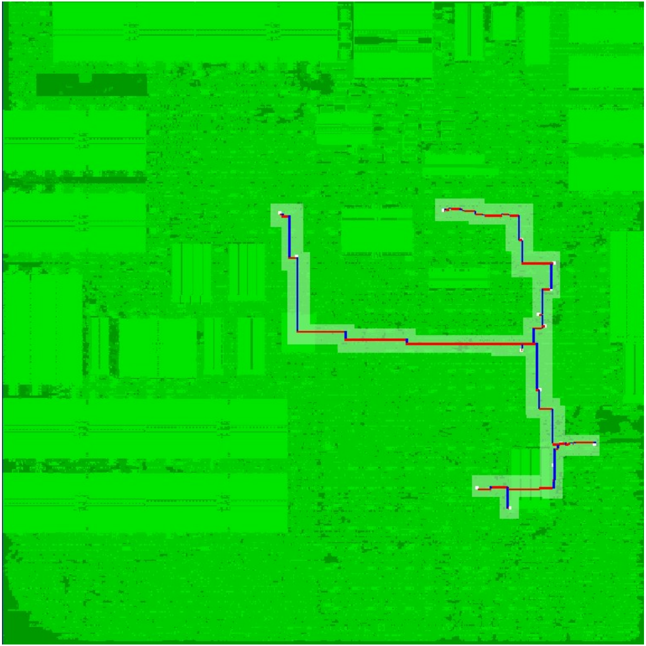
- route nets sequentially (in a good order)
- compose Steiner trees of paths
- main subroutine: find a shortest path (with respect to good edge weights)
- if no path exists, rip-up and re-route

- The order of the (sub)nets should depend on an estimate how close we are to blocking the (sub)nets
- The weights should reflect waste of routing space and electrical capacitance and resistance. Edges on track should be cheapest, orthogonal edges and vias more expensive

The key subroutine: path search

- find a shortest path in a subgraph of the weighted track graph
- restrict each path search to a relatively small area (computed by global routing)
- goal-oriented search
- more later...

Restrict path search to global routing region (corridor)



Goal-oriented search, future cost, feasible potentials

Given a digraph G with arc costs $c : E(G) \to \mathbb{R}_+$.

A function $\pi : V(G) \to \mathbb{R}$ is called a feasible potential if the reduced cost $c_\pi(e) := c(e) + \pi(v) - \pi(w)$ is nonnegative for each $e = (v, w) \in E(G)$.

Let $s, t \in V(G)$. We look for a shortest s-t-path w.r.t. c.

Observation: A shortest s-t-path w.r.t. c is a shortest s-t-path w.r.t. c_π, and vice versa.

Suppose $\mathcal{L}(x)$ is a lower bound on the distance from x to t, and $\mathcal{L}(v) \leq c(e) + \mathcal{L}(w)$ for each $e = (v, w) \in E(G)$.
Then $\pi(x) := -\mathcal{L}(x)$ is a feasible potential.
$\mathcal{L}(x)$ is also called the future cost at x.

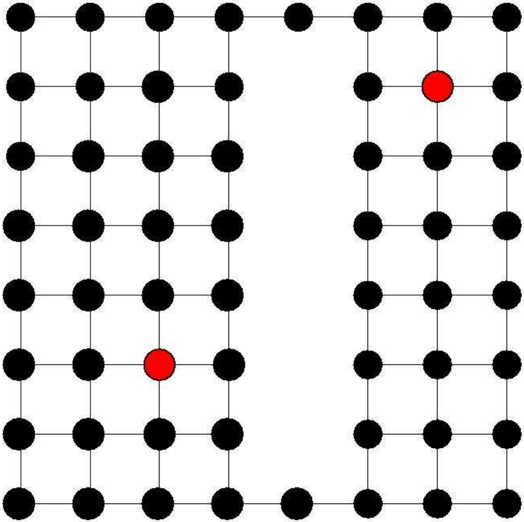
How to compute $\mathcal{L}$

Set $\mathcal{L}(v)$ to the length of a shortest path from v to T in (G', c') where G' is a supergraph of G and $c'(e) \leq c(e)$ for all $e \in E(G)$.

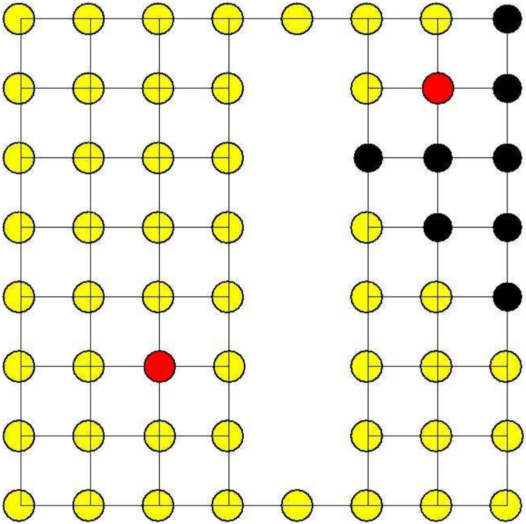
Choose (G', c') such that $\mathcal{L}$ is a good lower bound which can be computed fast.
A lower bound is good if it is close to the actual distance.

- ℓ_1-distance to target. But: the target is not necessarily a point. Need a Voronoi diagram first ($O(n \log n)$ preprocessing), then constant time.
- Choose (G', c') as a suitable subgraph of the track graph (with a simple structure) and, at the same time, supergraph of the current instance (details follow). Let G' be defined by the global routing corridor.

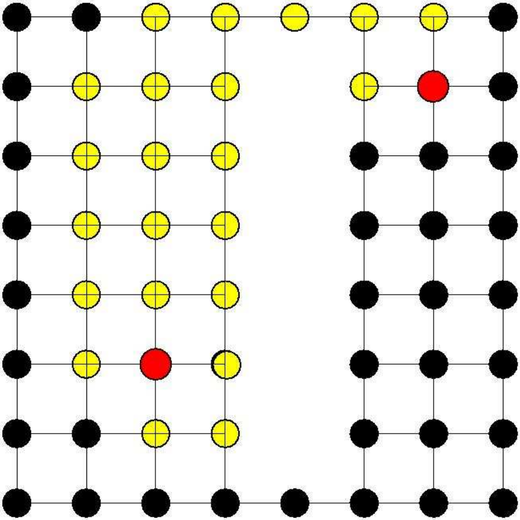
Future cost: example



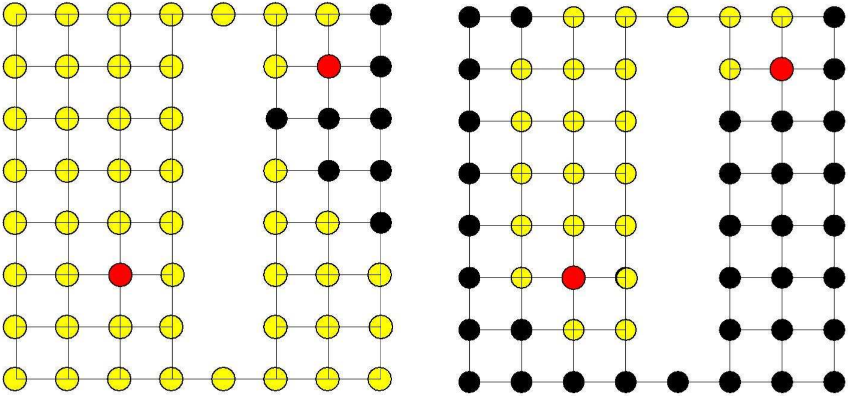
Dijkstra without future cost



Dijkstra with future cost (ℓ_1-distance)



Comparison with and without future cost



50 points labelled

24 points labelled

Generalizing Dijkstra's algorithm

Given

- a digraph G with edge lengths $c : E(G) \to \mathbb{R}_+$
- a set $T \subseteq V(G)$
- sets $V_1, V_2, \ldots, V_l \subseteq V(G)$ and $1 \leq k \leq l$ such that $T = \bigcup_{i=1}^{k} V_i$ and $V(G) = \bigcup_{i=1}^{l} V_i$.

we want to determine

$$d(v) := \mathrm{dist}_{(G,c)}(v, T)$$

for all $v \in V(G)$.

We label the sets V_i instead of single vertices, by functions $d_i : V_i \to \mathbb{R}_+ \cup \{\infty\}$ with $d_i(u) \geq d(u)$ for all $u \in V_i$.

Initially, $d_i(u) := 0$ for $1 \leq i \leq k$ and $u \in V_i$, and $d_i(u) := \infty$ for $k < i \leq l$ and $u \in V_i$. Then we repeatedly apply:

UPDATE(V_i, V_j):

Replace $d_j(u)$ by

$$\min\{d_j(u), \min\{d_i(v) + \mathrm{dist}_{(G[V_i \cup V_j],c)}(u, v) : v \in V_i\}\}$$

for all $u \in V_j$.

Generalizing Dijkstra's algorithm: optimality conditions

Theorem

Suppose that we have functions $d_1, d_2, \ldots, d_l$ with:

- $d_i(u) = 0$ *for all* $u \in V_i$ *and* $i = 1, \ldots, k$.
- $d_i(u) \geq d(u)$ *for all* $u \in V_i$ *and* $i = 1, \ldots, l$.
- *For each edge* $e = (v, w) \in E(G)$ *and each* $i \in \{1, \ldots, l\}$ *with* $w \in V_i$ *there exists a* $j \in \{1, \ldots, l\}$ *with* $v \in V_j$ *and* $d_j(v) \leq c(e) + d_i(w)$.

Then $d(v) = \min\{d_i(v) : i = 1, \ldots, l,\ v \in V_i\}$ *for all* $v \in V(G)$.

Proof: Suppose that $d(v) < \min\{d_j(v) : j = 1, \ldots, l,\ v \in V_j\}$; choose v such that $d(v)$ is minimum; in case of ties the shortest v-T-path P shall have minimum number of edges. Let w be the neighbour of v on P.
By the choice of v, there exists an $i \in \{1, \ldots, l\}$ with $w \in V_i$ and $c((v, w)) + d_i(w) = c((v, w)) + d(u) = d(v) < \min\{d_j(v) : j = 1, \ldots, l,\ v \in V_j\}$. This is a contradiction. $\square$

(Peyer, Rautenbach, V. [2006])

GENERALIZED DIJKSTRA

Set $d_i(u) := 0$ for $1 \leq i \leq k$ and $u \in V_i$.
Set $d_i(u) := \infty$ for $k < i \leq l$ and $u \in V_i$.
Set $Q := \{1, \ldots, k\}$ and $\text{key}(i) := 0$ for $i = 1, \ldots, k$.
WHILE $Q \neq \emptyset$ DO:
    Choose $i \in Q$ with $\text{key}(i)$ minimum. Set $Q := Q \setminus \{i\}$.
    PROJECT(i).

PROJECT(i):
    Choose $J \subseteq \{1, \ldots, l\} \setminus \{i\}$ such that $\bigcup_{j \in \{i\} \cup J} V_j$ contains all neighbours of V_i.
    FOR $j \in J$:
        UPDATE(V_i, V_j).
        IF $d_j(v)$ changes for some $v \in V_j$,
        THEN let $\text{key}(j)$ be the minimum changed $d_j(v)$, $v \in V_j$, and set $Q := Q \cup \{j\}$.

GENERALIZED DIJKSTRA: optimality

Theorem

This algorithm produces functions $d_1, d_2, \ldots, d_l$ satisfying the optimality conditions.

Proof: The statement is obvious for the first two conditions. Therefore, suppose, for a contradiction, that there exists an edge $e = \{u, v\} \in E(G)$ and an index $i \in \{1, \ldots, l\}$ such that $d_j(v) > d_i(u) + c(e)$ for all $j \in \{1, \ldots, l\}$ with $v \in V_j$.

Then $v \notin V_i$. Since $d_i(u) < \infty$, we have $i \in Q$ at some moment.

Consider the last time that the algorithm executes PROJECT(i). Note that d_i does not change after this moment.

As v is a neighbour of $u \in V_i$, there is some $j \in J$ with $v \in V_j$ and UPDATE(V_i, V_j) ensures

$$d_j(v) \leq d_i(u) + \mathrm{dist}_{(G[V_i \cup V_j],c)}(u, v) \leq d_i(u) + c(e).$$

As $d_j(v)$ never increases, this is a contradiction. $\square$

(Peyer, Rautenbach, V. [2006])

GENERALIZED DIJKSTRA: running time

- If we implement Q by a Fibonacci heap, the running time is is $O(n(\log l + p))$, where p is the time for one PROJECT operation and n is the number of iterations.
- Since every $i \in \{1, \dots, k\}$ enters Q exactly once and every $i \in \{k+1, \dots, l\}$ enters Q at most $|V_i|$ times, we only have the bound $n \leq k + \sum_{i=k+1}^{l} |V_i|$ in general.
- If $V_1, \dots, V_l$ is a partition of $V(G)$ into one-element sets, then this is the standard algorithm with running time $O(m + n \log n)$, where $n = |V(G)|$ and $m = |E(G)|$.
- Much faster for special graphs, in particular grid graphs
- Sorting $V_{k+1}, \dots, V_l$ such that $c((u, v)) > 0$ for $(u, v) \in E(G) \cap ((V_i \times (V_j \setminus V_i)) \cup ((V_i \setminus V_j) \times V_j))$ and $i < j$ gives that each $i \in \{k+1, \dots, l\}$ enters Q at most once for each key.

Modeling the routing space by a grid graph

Let G_0 be the infinite 3-dimensional grid graph, i.e. $V(G_0) = \mathbb{Z}^3$, and
$E(G_0) = \{\{(x,y,z),(x',y',z')\} : |x-x'| + |y-y'| + |z-z'| = 1\}$.
We assume that for each $z \in \mathbb{Z}$ there are three constants $c_{z,1}, c_{z,2}, c_z \in \mathbb{R}$ such that

$$\begin{aligned} c(\{(x,y,z),(x+1,y,z)\}) &= c_{z,1}, \\ c(\{(x,y,z),(x,y+1,z)\}) &= c_{z,2}, \text{ and} \\ c(\{(x,y,z),(x,y,z+1)\}) &= c_z \end{aligned}$$

for all $x, y \in \mathbb{Z}$, This reflects higher costs for vias and jogs and in access planes.
We look for shortest paths w.r.t. c in induced subgraphs of G_0.

GENERALIZED DIJKSTRA on grids

Let G be an induced subgraph of the infinite 3-dimensional grid. Write $V(G)$ as the union of rectangles $V_1, \ldots, V_l$ such that each has $O(\log l)$ neighbours.
Assume that the number of different edge weights is constant.
Then:

- the number of iterations is $O(l)$
- the functions d_i can be stored in constant space
- an UPDATE operation takes constant time
- the cardinality of the set J to be considered in the PROJECT operation is $O(\log l)$

$\Rightarrow$ Running time of $O(l \log l)$

(Peyer, Rautenbach, V. [2006])

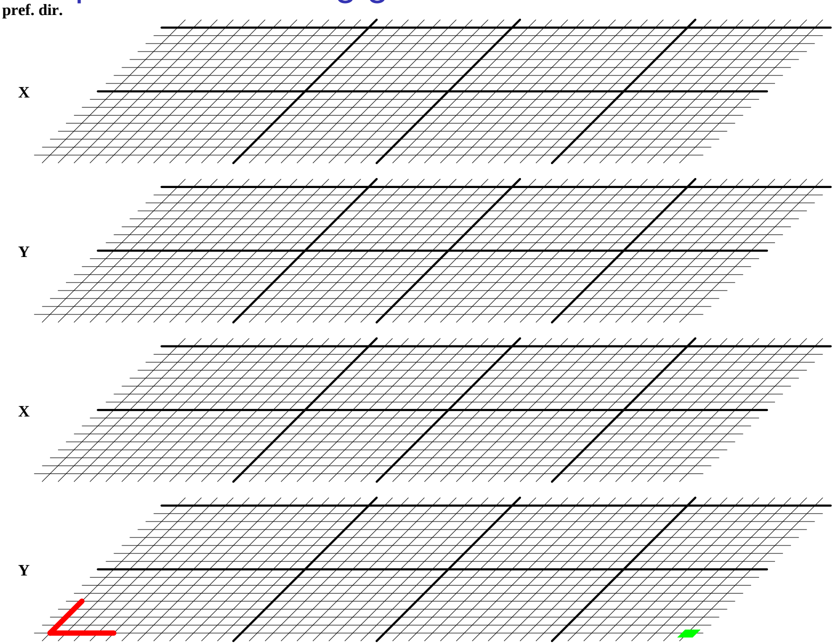
GENERALIZED DIJKSTRA for accurate future costs

- Consider a supergraph G' of the graph G representing the routing area, such that G' can be decomposed into few rectangles (and in which distances are not much shorter).
- Apply GENERALIZED DIJKSTRA to G', labeling these rectangles.
- As $d(v) = \mathrm{dist}_{(G',c)}(v, T) \leq \mathrm{dist}_{(G,c)}(v, T)$, the numbers $d(v)$ serve as future cost for shortest path computation in G.

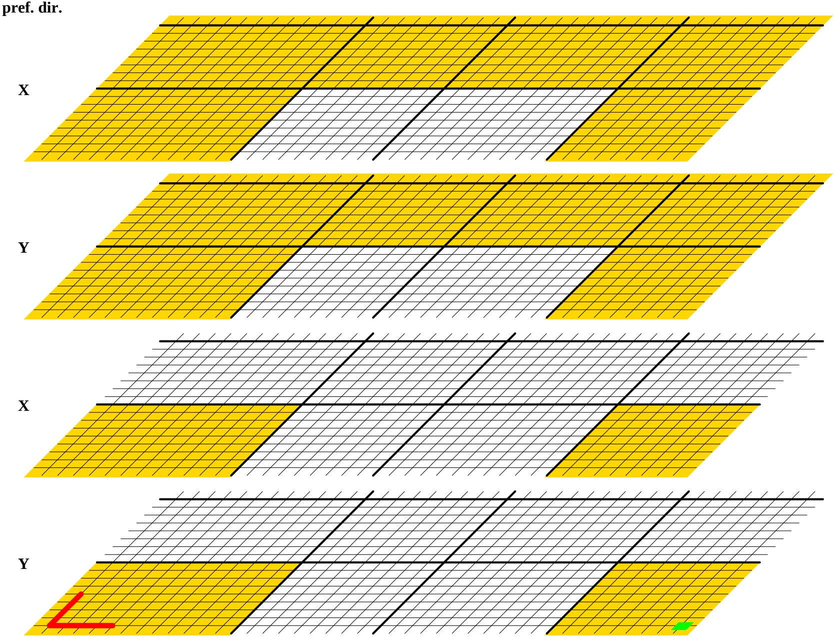
Example for accurate future costs

- four layers
- alternating preference directions
- we look for a path from a (green) source to a (red) target
- edge cost 1 in preference direction
- edge cost 4 in orthogonal direction
- edge cost 7 for vias

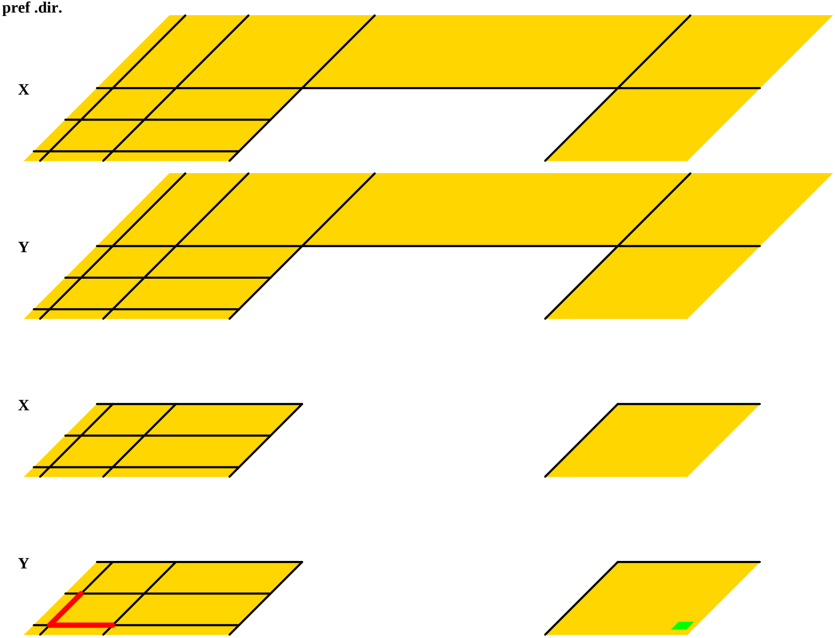
Example: local routing grid



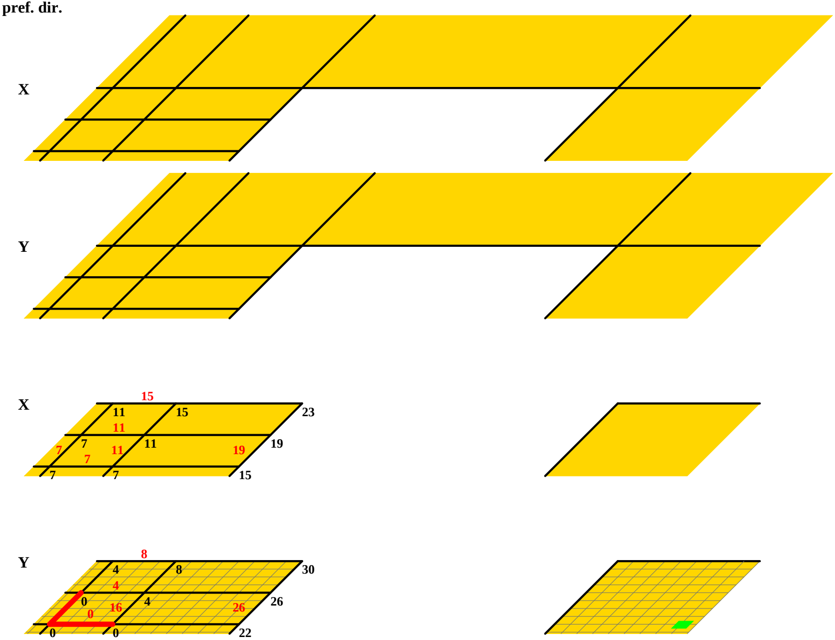
Example: global routing corrdidors



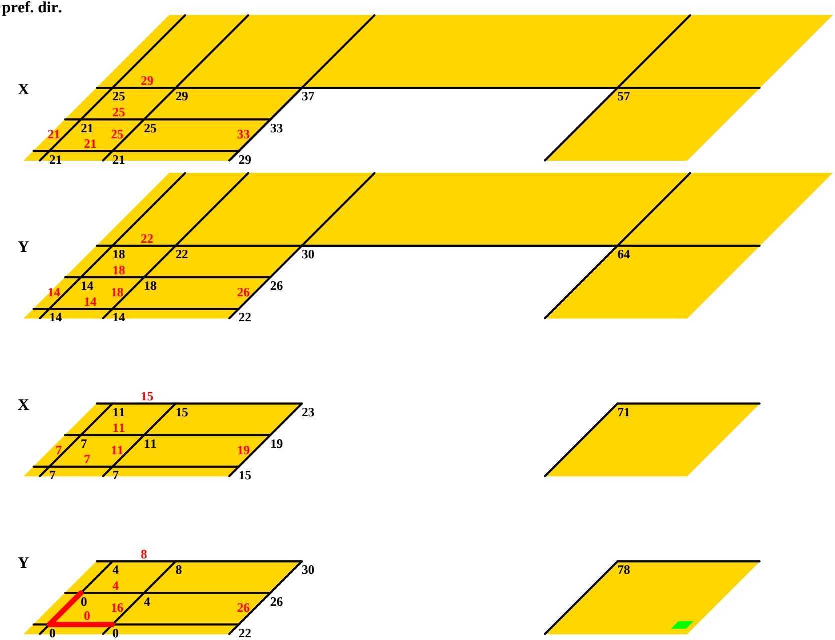
Example: Hanan grid



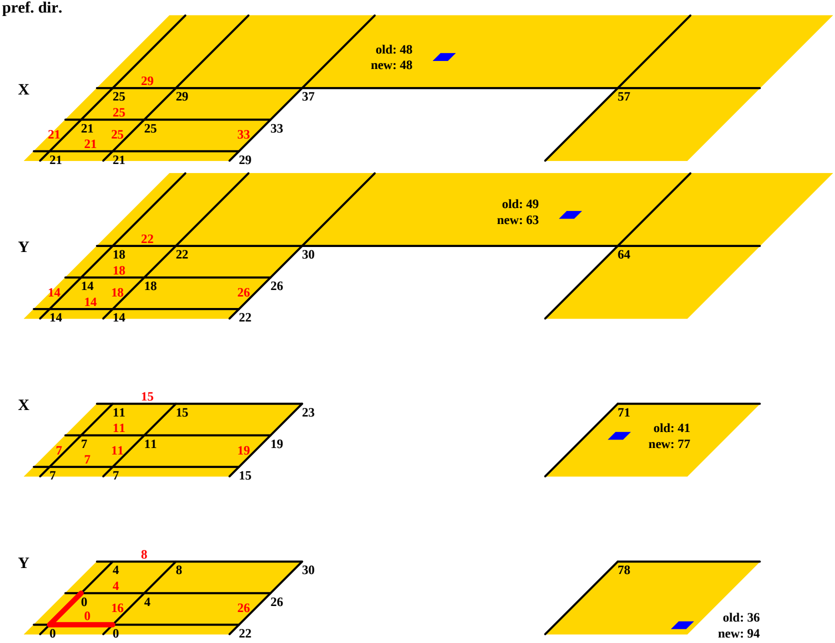
Example: GENERALIZED DIJKSTRA



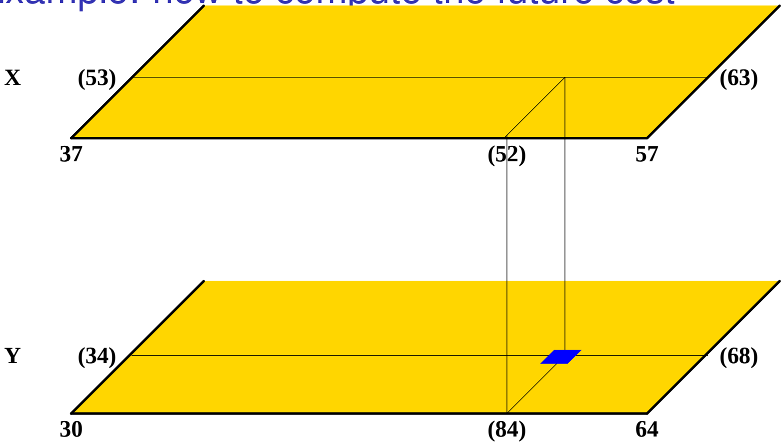
Example: GENERALIZED DIJKSTRA



Example: old (ℓ_1-distance) versus new future cost



Example: how to compute the future cost

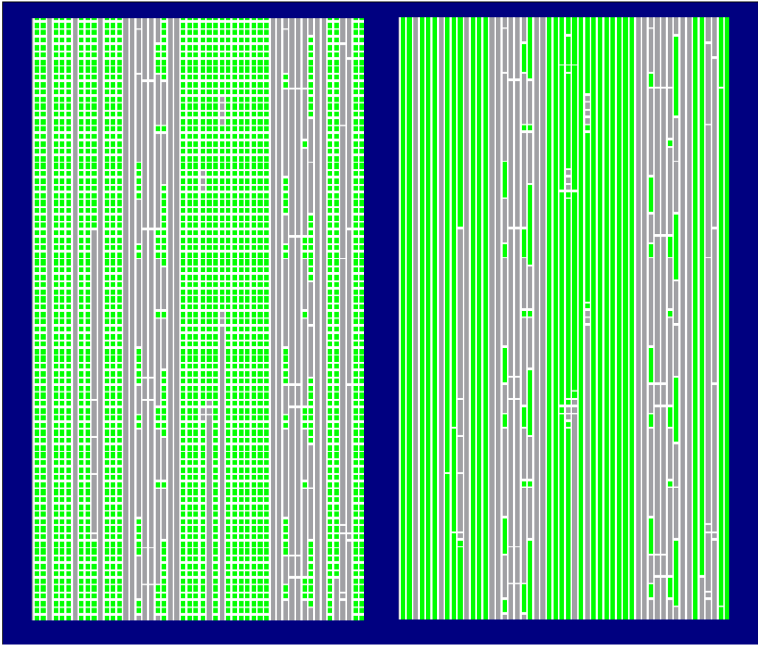


$$\color{blue}{63} = \min(\ 34 + 15 \times 4,\ 68 + 5 \times 4,\ 84 + 4 \times 1,\ 53 + 15 \times 1 + 7,\ 63 + 5 \times 1 + 7,\ \color{blue}{52 + 4 \times 1 + 7}\ )$$

The key subroutine: path search

- find a shortest path in a subgraph of the weighted track graph
- restrict each path search to a relatively small area (computed by global routing)
- goal-oriented search
- represent the routing area by a set of intervals (with constant properties)
- label intervals rather than single points

Detailed routing: intervals



Path search on intervals

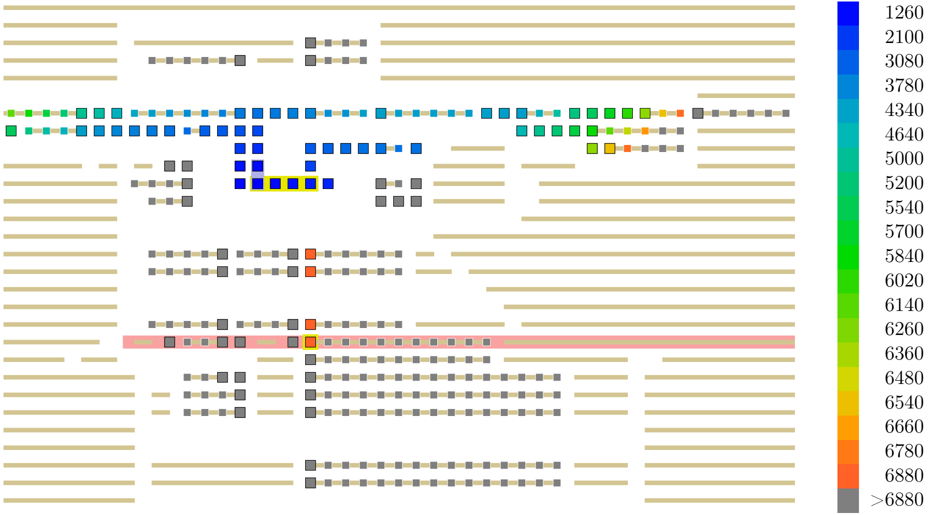
- goal-oriented Dijkstra
- label intervals rather than single vertices
- vertices are not stored anywhere!
- efficient data structure for managing intervals and labels
- algorithm can be viewed again as a special case of GENERALIZEDDIJKSTRA.

Theorem

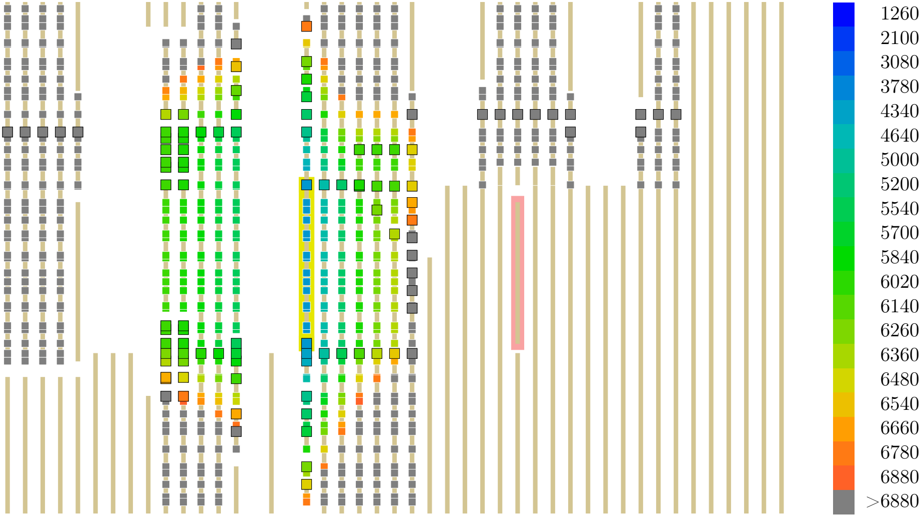
We can find a shortest path in $O((d+1)I \log I)$ time, where d is the detour (actual length minus lower bound), and I is the number of intervals in the search space.

Hetzel [1995,1998], Peyer Rautenbach, V. [2007], Humpola [2009]

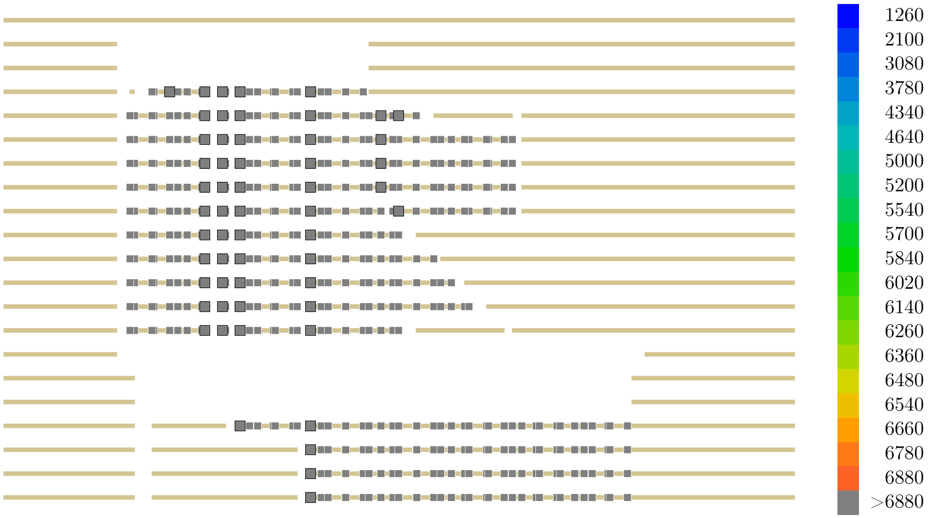
Labeling intervals: old future cost (ℓ_1-distance), layer 1



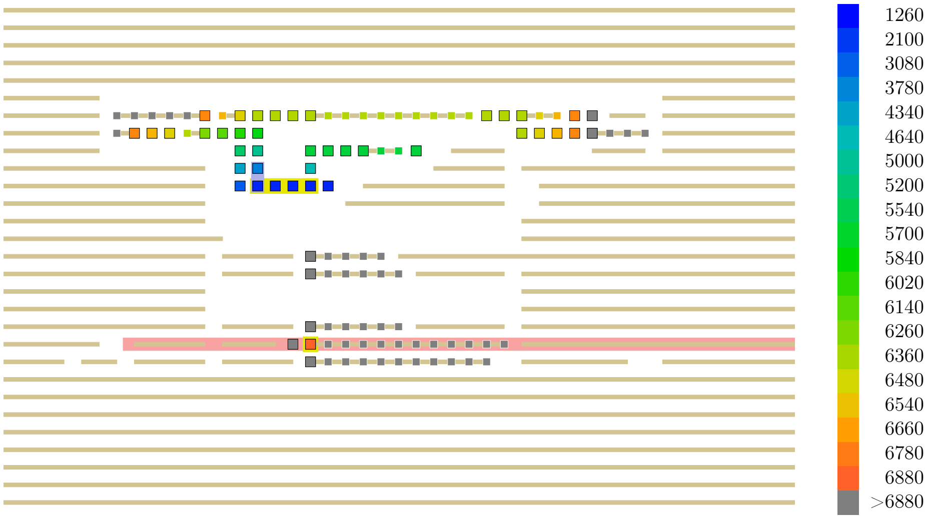
Labeling intervals: old future cost (ℓ_1-distance), layer 2



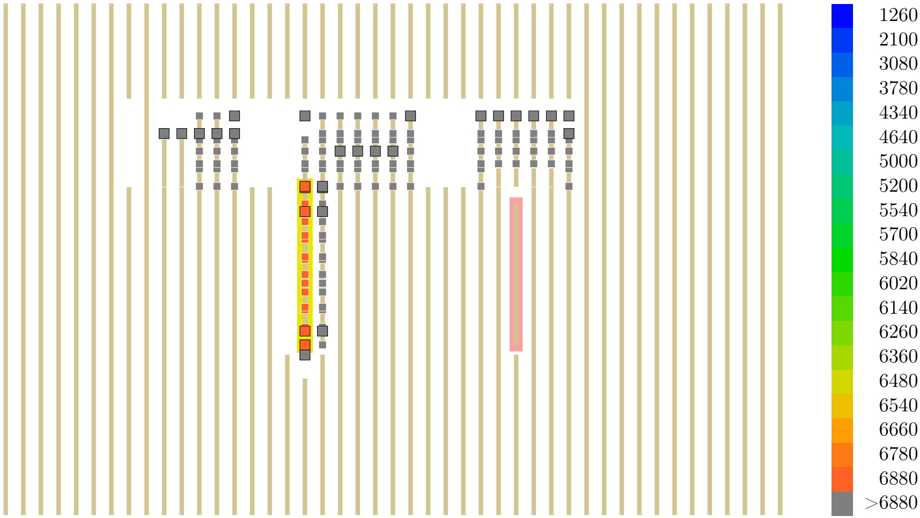
Labeling intervals: old future cost (ℓ_1-distance), layer 3



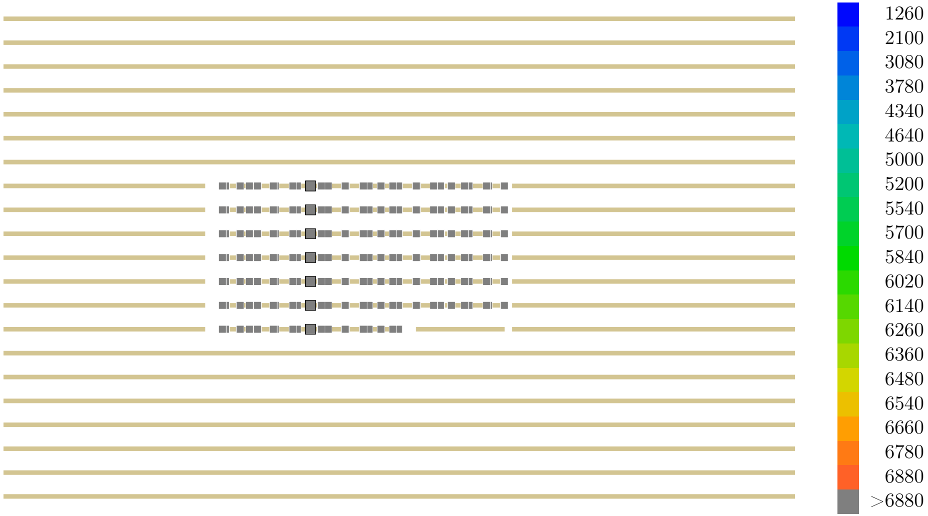
Labeling intervals: new future cost, layer 1



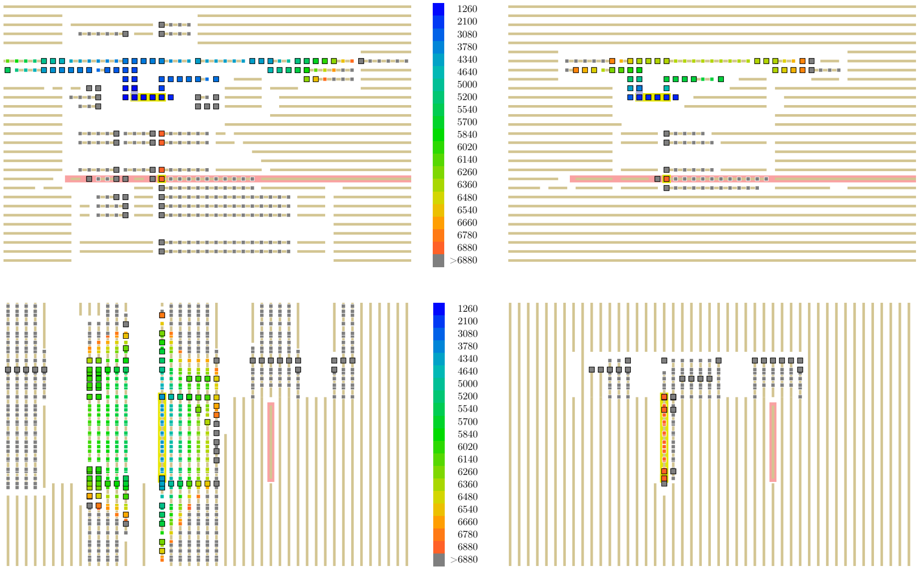
Labeling intervals: new future cost, layer 2



Labeling intervals: new future cost, layer 3



Labeling intervals: old versus new future cost



Detailed routing: summary

- huge instances, complicated rules
- model routing space by track graph
- the track graph can have more than 10^{11} vertices
- route nets sequentially, subnets by a variant of Dijkstra's algorithm
- restrict path search to small areas (computed by global routing)
- goal-oriented Dijkstra: use accurate future cost
- label intervals rather than single points
- special algorithms for local pin access
- postprocessing for same-net errors, design for manufacturability