BonnRoute: Algorithms and Data Structures for Fast and Good VLSI Routing

MICHAEL GESTER, DIRK MÜLLER, TIM NIEBERG, CHRISTIAN PANTEN, CHRISTIAN SCHULTE, and JENS VYGEN, University of Bonn

We present the core elements of BonnRoute: advanced data structures and algorithms for fast and high-quality routing in modern technologies. Global routing is based on a combinatorial approximation scheme for min-max resource sharing. Detailed routing uses exact shortest path algorithms, based on a shape-based data structure for pin access and a two-level track-based data structure for long-distance connections. All algorithms are very fast. Compared to an industrial router (on 32 nm and 22 nm chips), BonnRoute is over two times faster, has 5 % less netlength, 20 % less vias, and reduces detours by more than 90 %.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids

General Terms: Algorithms, Design

Additional Key Words and Phrases: Global routing, detailed routing, routing optimization

ACM Reference Format:
Gester, M., Müller, D., Nieberg, T., Panten, C., Schulte, C., and Vygen, J. 2013. BonnRoute: Algorithms and data structures for fast and good VLSI routing ACM Trans. Des. Autom. Electron. Syst. 18, 2, Article 32 (March 2013), 24 pages.
DOI: http://dx.doi.org/10.1145/2442087.2442103

1. INTRODUCTION

BonnRoute is the routing solution of the University of Bonn, developed within the scope of our cooperation with IBM. It is part of the BonnTools [Korte et al. 2007]. BonnRoute has been used by IBM and its customers for the design of more than thousand of the most complex chips. In this article we present its core algorithms, give implementation details, and present results. To the best of our knowledge, this is the first work to describe a full industrial routing solution with all relevant technical details.

1.1. Scope

We restrict ourselves to *Manhattan routing*, meaning that all wires run parallel to the x- or y-axis. On a single routing layer, either almost all wires are horizontal or almost all are vertical (this is called the *preferred direction* of a layer; wires running orthogonally are called *jogs*). Horizontal and vertical layers alternate. This still is common design practice today, although there are design systems that allow wires running in directions that are integer multiples of 45 or 60 degrees [Teig 2002; Ho et al. 2005; Chen et al. 2003].

Routing the most complex chips in the current technologies poses several challenges (cf. also Alpert et al. [2010]).

An extended abstract of this work appeared in the Proceedings of the Design Automation Conference 2012.
Author's address: Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2, D-53113 Bonn; email: {gester, mueller, nieberg, panten, schulte,vygen}@or.uni-bonn.de.

DOI: http://dx.doi.org/10.1145/2442087.2442103

—Finding any feasible solution is often very difficult. Even in a simplified version, this includes the vertex-disjoint Steiner trees problem in graphs, which is among the hardest combinatorial optimization problems [Korte et al. 1990].
—The design rules tend to become more complicated with each new technology generation. While *diff-net rules*, requiring a certain minimum distance of wires that belong to different nets, are most important, *same-net rules*, prohibiting certain configurations of shapes belonging to the same net, also require more and more attention. Ideally, the computed routing should pass all design rule checks (DRC).
—For efficient packing of wires on each layer it is useful to largely restrict to parallel wires, mostly on so-called routing tracks. However, pins are often not perfectly aligned and have many blockages around them. Hence off-track wiring and jogs are needed at least for pin access.
—Mostly for timing reasons, many nets require nonstandard wire widths, special via shapes, larger spacing than normal, or restriction to a subset of routing layers.
—The quality of the overall routing solution is extremely important. Large detours often lead to violated timing constraints. The manufacturing yield is significantly influenced by the routing, in particular by the number of vias and their shapes, and wire length directly impacts the power consumption (for details, see, e.g., Müller [2006] and Vygen [2004]). Poor quality solutions are either useless or lead to inferior chips or require large post-processing effort.
—Huge instance sizes make it difficult to meet expected turn-around times of just a few hours. This requires extremely efficient algorithms and data structures.

1.2. Related Work

Routing is traditionally split into global and detailed routing. This has essentially two reasons. First, detailed routing usually takes most of the running time, and restricting the routing space to be explored for a single net to the solution obtained in global routing means a significant speed-up. Second, we can obtain a globally near-optimum solution in global routing (as we shall see below), and by following this solution in detailed routing we obtain also an excellent detailed routing solution.

Many algorithms that have been applied to global routing are based on heuristics, the most popular approaches being ripup-and-reroute methods. An early and influential ripup-and-reroute algorithm was described by Ting and Tien [1983]. The first to consider global routing as a multicommodity flow problem were Shragowitz and Keel [1987]. They proved that their algorithm, if it terminates, finds a feasible solution if one exists, but they gave no guarantees on termination, running time, or the achieved objective value. Carden et al. [1996] applied the approximation algorithm of Shahrokhi and Matula [1990] for the maximum concurrent flow problem to global routing, providing an approximate optimality guarantee for the first time. [Albrecht 2001] applied the simpler and faster algorithm of Garg and Könemann [2007]. This algorithm was targeted to optimize wiring length. It was extended by Vygen [2004] to incorporate coupling, delay bounds and power consumption, and subsequently used by Müller [2006] to optimize manufacturing yield. However, objective functions and constraints in these works were restricted to be linear.

Hu and Sapatnekar [2001] gave a survey of global routing algorithms published until 2001. The ISPD global routing contests [ISPD 2007, 2008] stimulated development of new global routing approaches recently; see Moffitt et al. [2008] for a survey. The methods employed range from pattern routing over iterative edge shifting and monotonic routing (restricted to stair-case-like paths), to computationally more expensive ripup-and-reroute with history-based congestion costs, often called negotiation-based routing [Chang et al. 2008; Moffitt 2008; Xu et al. 2009; Roy and Markov 2008; Ozdal and Wong 2009; Chen et al. 2009]. Most of these routers exploit the fact that in

these benchmarks all wires running on the same layer have the same width and each net can be routed on all layers. This allows solution of a 2D problem first and mapping wires to layers in a subsequent step, for instance, as described in Lee and Wang [2008]. Moffitt [2009], [ICCAD] proposed a new set of benchmarks based on ISPD [2007, 2008], in which the layers usable by individual nets are restricted by layer directives, and some methods have been proposed to take these constraints into account [Moffitt and Sze 2011; Lee et al. 2011]. Also integer linear programming has been applied to global routing [Cho et al. 2009; Wu et al. 2011a, 2011b], employing decomposition techniques on large instance sizes. Two-dimensional global routers are usually followed by layer assignment. Since our global routing is three-dimensional, we do not need such a step.

Some routers have a step in between global and detailed routing, often called *track assignment* [Batterywala et al. 2002; Li et al. 2011] or *pseudopin assignment* [Chang and Cong 2001]. This computes an ordering of the nets within each global routing channel and a layout of at least the long-distance nets, often not satisfying all design rules. We do not use such a step.

For the core detailed routing part, one usually looks for a shortest feasible path connecting two components of a given net. Common approaches differ mainly in the way to represent routing space and in the algorithms used to find connecting paths. One can roughly distinguish between gridded and gridless (i.e., purely shape-based) algorithms. We use both.

Gridded routers model the routing space by a three-dimensional grid graph and use (a variant of) Dijkstra's algorithm [Dijkstra 1959] to find shortest paths. The main disadvantages are the size of the grid graph (with often more than 10^{11} vertices), which directly impacts the memory consumption and running time of Dijkstra's algorithm, and the inflexibility with respect to off-grid pins, wires of nonstandard width, or nonuniform distance requirements. Variants of Dijkstra's algorithm specialized to grid graphs, also called *maze running*, were proposed in Lee [1961] (for uniform edge costs), Rubin [1974] (using goal-orientation), Hoel [1976], Hadlock [1977] and Johann and Reis [2000]. Hetzel [1998] combined Rubin's algorithm [Rubin 1974], using ℓ_1-distance as lower bound for shortest paths, with merging "congenerous" vertices to so-called intervals. This algorithm was generalized by Peyer et al. [2009] and Humpola [2009] and is the basis for on-track path search in BonnRoute (cf. Section 4.1).

Gridless (aka shape-based) routers model the routing space by a set of rectilinear obstacles and find connections by using an algorithm for the rectilinear obstacle-avoiding shortest path problem, such as Hanan [1966], Lee et al. [1996], and Mitchell [1989]. Based on this, various gridless routers have been developed, for instance, Chen and Chang [2007], Cong et al. [1999, 2001], and Dion and Monier 1995]. Many design rules regarding minimum distances between objects of *different* nets are easily incorporated into these approaches by blowing up the obstacles [Dion and Monier 1995]. In Chen and Chang [2007], a penalty cost based on worst-case scenarios for future wires is introduced to avoid diff-net rule violations. Correcting same-net rule violations is often left for post-processing steps [Cong et al. 1999; Cong et al. 2001]. Cadence's Space Based Router [Cross et al. 2007] is a gridless router with a special emphasis on respecting design rules and manufacturing-aware routing.

The main disadvantage of gridless routing is the unstructured routing space, making it difficult to pack wires efficiently if the nets are considered more or less one at a time, resulting in unnecessary detours or even failure in completing the routing.

1.3. Outline/Our Contributions

Our global router, which we describe in Section 2, is based on an efficient fully polynomial approximation scheme for the min-max resource sharing problem. It produces a

provably near-optimal fractional solution, rounds it, and removes the local congestion caused by rounding. The main subroutine finds approximately shortest Steiner trees in a weighted grid graph very fast.

For detailed routing, we precompute routing tracks that we will use for the majority of the wires in order to pack them efficiently. The available routing space is modeled by an efficient two-level data structure (called fast grid and shape grid). We discuss this and other aspects of representing routing space in Section 3.

A specialized version of Dijkstra's algorithm exploits this data structure to find shortest paths extremely fast even for long distances. Off-track paths are computed by a shape-based shortest path algorithm to access pins, also taking same-net rules into account. We present these routing algorithms in Section 4.

With its near-optimum resource sharing based global routing, efficient data structures representing routing space based on optimally aligned routing tracks, and highly efficient path search routines, BonnRoute can normally find a routing solution even on huge instances if one exists. As BonnRoute's focus is on globally optimum packing of wires rather than DRC-cleanness, we combine it with an industrial router for DRC cleanup. In this way, we obtain competitive numbers of DRC violations while improving considerably netlength, via count, detours, and overall runtime. We present these results in Section 5.

2. GLOBAL ROUTING

2.1. Modeling the Global Routing Problem

Global routing is an abstraction of the actual routing problem to a coarser model of the routing space. The chip area is divided into an array of *tiles*. The size of our tiles is chosen such that approximately 50 to 100 parallel wires (of minimum width) would fit into one tile on each layer.

For each tile and each routing layer we have a vertex. The vertices thus constitute a partition of the routing space. Two vertices (t, l) and (t', l') are connected by an edge if $t = t'$ and $|l - l'| = 1$, or if $l = l'$ and t and t' are two tiles that are adjacent in the preferred direction of routing layer l. This defines an undirected graph G. Unlike in detailed routing, we do not allow routing in non-preferred direction because even with small tile sizes this would block too many routing tracks.

Next we compute edge capacities $u : E(G) \to \mathbb{R}_{\geq 0}$, estimating the number of standard wires that can go from a vertex to a neighboring vertex while obeying the minimum distance requirements. We explain how we compute edge capacities in Section 2.5.

Each pin p has shapes in one or more tiles, and is represented by the respective set V_p of vertices. Each net is a set of pins, and $\mathcal{N}$ denotes the set of nets. Let $\mathcal{T}_n$ denote the set of feasible *Steiner forests* for net n, that is, minimal edge sets $F \subseteq E(G)$ such that $F \cup \bigcup_{p \in n} K(V_p)$ connects all pins of n, where $K(V_p)$ denotes the clique on V_p. Any net n for which $\emptyset \in \mathcal{T}_n$ (e.g., because all pins are in the same tile) can be removed.

For a net n and an edge $e \in E(G)$ let $w(n, e)$ denote the minimum required width of a wire for n along e plus the minimum required distance to any neighbor. This width usually does not depend on the individual edge but only on the layer; however the above notation is simpler and more general.

A major difference to most other global routers is that we allow to allocate extra space $s(n, e) \geq 0$ as this may reduce coupling capacitance and hence delay and power consumption, and may also lead to better yield.

Then the global routing task is to find for each $n \in \mathcal{N}$ a Steiner forest $T_n \in \mathcal{T}_n$ and an extra space assignment $s(n, e) \geq 0$ for $e \in T_n$ such that $\sum_{n \in \mathcal{N}: e \in T_n} (w(n, e) + s(n, e)) \leq u(e)$ for each $e \in E(G)$.

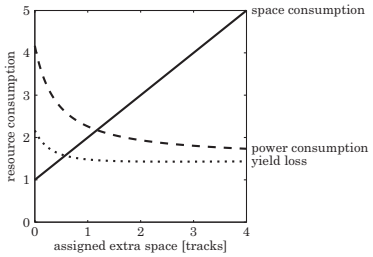

Fig. 1. The resource consumption $\gamma^r_{n,e}(s)$ of a net $n \in \mathcal{N}$ using an edge $e \in E(G)$, depending on the assigned extra space $s \geq 0$, for three resources r: Power consumption (dashed), manufacturing yield loss (dotted), and space consumption on the edge e (solid line).

However, we are not satisfied with an arbitrary feasible solution. While a traditional objective function considers only wire length and number of vias, we can also optimize other objective functions like power consumption or expected yield, and we can also deal with constraints bounding, for instance, detours of certain nets or weighted sums of capacitances on critical paths. The objective function and each of such constraints will be modeled as a *resource*. We denote by $\mathcal{R}$ the set of resources.

Each wire consumes a certain amount of some of the resources. This is modeled by functions $\gamma^r_{n,e} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ for $r \in \mathcal{R}$, $n \in \mathcal{N}$ and $e \in E(G)$. Here, $\gamma^r_{n,e}(s)$ is the estimated use of resource r if e is used by net n with allocated space $w(n,e) + s$. For all relevant resources that we consider, including electrical capacitance (with coupling), power consumption, and estimated wiring yield loss, these functions are convex (cf. Müller [2006] and Vygen [2004]). Figure 1 shows an example.

For each constraint, we have an upper bound $u^r > 0$ of how much we may use of the corresponding resource r. For the objective function, we do not have an upper bound a priori, but we can guess a value that we expect to be achievable and adapt it if needed (we could apply binary search to find approximately the optimum value, but this is usually not needed in practice).

We also model the edge capacities as resources. If $r(e)$ denotes the resource corresponding to edge e, we have $u^{r(e)} = u(e)$, $\gamma^{r(e)}_{n,e}(s) = w(n,e) + s$, and $\gamma^{r(e)}_{n,e'}(s) = 0$ for any other edge $e' \neq e$ and $s \geq 0$.

Now the task is to find for each $n \in \mathcal{N}$ a Steiner forest $T_n \in \mathcal{T}_n$ and an extra space assignment $s(n,e) \geq 0$ for $e \in T_n$ such that $\sum_{n \in \mathcal{N}} \sum_{e \in T_n} \gamma^r_{n,e}(s(n,e)) \leq u^r$ for all $r \in \mathcal{R}$.

2.2. Reduction to Min-Max Resource Sharing

The above problem is NP-hard as it includes the edge-disjoint paths problem and the Steiner tree problem. This is true even in the special case of wire length minimization with edge capacities as the only constraints, which was considered previously by Carden et al. [1996] and Albrecht [2001].

Let $\chi(T) \in \{0,1\}^{E(G)}$ denote the incidence vector of a Steiner forest T (i.e., $(\chi(T))_e = 1$ for $e \in T$ and $(\chi(T))_e = 0$ for $e \notin T$). Then

$$\mathcal{B}^{\text{int}}_n := \{\big(\chi(T), s\big) \mid T \in \mathcal{T}_n,\ s \in \mathbb{R}^{E(G)}_{\geq 0},\ s_e = 0 \text{ for } e \notin T\}.$$

represents the set of feasible solutions for net n. In our relaxation we consider the convex hull $\mathcal{B}_n := \text{conv}\left(\mathcal{B}^{\text{int}}_n\right)$ for each $n \in \mathcal{N}$. For $(x,s) \in \mathcal{B}_n$ we define

$$g^r_n(x,s) := \frac{1}{u^r} \sum_{e \in E(G): x_e > 0} x_e\, \gamma^r_{n,e}(s_e/x_e).$$

Note that for every $(\chi(T), s) \in \mathcal{B}_n^{\text{int}}$ we have $g_n^r(\chi(T), s) = \frac{1}{u^r} \sum_{e \in T} \gamma_{n,e}^r(s_e)$. If each $\gamma_{n,e}^r$ is convex (as in our applications), then each g_n^r is also convex. With this notation, the global routing problem asks for an element $b_n \in \mathcal{B}_n^{\text{int}}$ for each $n \in \mathcal{N}$ such that $\sum_{n \in \mathcal{N}} g_n^r(b_n) \leq 1$ for all $r \in \mathcal{R}$. As in previous works, we now consider a fractional relaxation first and look for $b_n \in \mathcal{B}_n$ $(n \in \mathcal{N})$ approximately attaining

$$\lambda^* := \inf \left\{ \max_{r \in \mathcal{R}} \sum_{n \in \mathcal{N}} g_n^r(b_n) \,\middle|\, b_n \in \mathcal{B}_n \, (n \in \mathcal{N}) \right\}.$$

If the instance is feasible and the guess of an achievable value of the objective function is realistic (see above), then λ^* will be close to 1.

This relaxation is known as the (block-angular) MIN-MAX RESOURCE SHARING PROBLEM. It contains the multi-commodity flow problem as special case (which corresponds to wire length minimization; cf. Carden et al. [1996]; Albrecht [2001]). However, it is much more general.

In the standard formulation of this problem, the sets $\mathcal{B}_n$ are not given explicitly. Rather we assume to have an oracle, denoted by $f_n : \mathbb{R}_{\geq 0}^{\mathcal{R}} \to \mathcal{B}_n$, for $n \in \mathcal{N}$, which for $y \in \mathbb{R}_{\geq 0}^{\mathcal{R}}$ returns an element $b \in \mathcal{B}_n$ with $\sum_{r \in \mathcal{R}} y_r g_n^r(b) \leq \sigma \operatorname{opt}_n(y)$, where $\operatorname{opt}_n(y) := \inf_{b \in \mathcal{B}_n} \sum_{r \in \mathcal{R}} y_r g_n^r(b)$ and $\sigma \geq 1$ is a constant. In other words, we assume that we can optimize linear functions over each $\mathcal{B}_n$ efficiently with an approximation factor σ.

This is indeed true in our case. In Müller et al. [2011], we proved that $\inf_{b \in \mathcal{B}_n^{\text{int}}} \sum_{r \in \mathcal{R}} y_r g_n^r(b) = \inf_{b \in \mathcal{B}_n} \sum_{r \in \mathcal{R}} y_r g_n^r(b)$ holds, that is, optimization over the set $\mathcal{B}_n^{\text{int}}$ gives no worse results than optimizing over its convex hull $\mathcal{B}_n$. Therefore, it suffices to find a $b^* \in \mathcal{B}_n^{\text{int}}$ with $\sum_{r \in \mathcal{R}} y_r g_n^r(b^*) \leq \sigma \inf_{b \in \mathcal{B}_n^{\text{int}}} \sum_{r \in \mathcal{R}} y_r g_n^r(b)$. To this end, we compute a $T^* \in \mathcal{T}_n$ with $\sum_{e \in T^*} \bar{y}_e \leq \sigma \min_{T \in \mathcal{T}_n} \sum_{e \in T} \bar{y}_e$, where

$$\bar{y}_e := \inf_{\substack{s \geq 0: \\ \gamma_{n,e}^r(s) \leq u^r}} \left(\sum_{r \in \mathcal{R}} y_r \gamma_{n,e}^r(s)/u^r \right) \tag{1}$$

is the total cost of using an edge $e \in E(G)$ used by $n \in \mathcal{N}$ with optimum extra space assignment. Then we can set $b^* := (\chi(T^*), s)$ for an appropriate s (which is easy to compute). To find T^*, we can add the edges of the clique $K(V_p)$ for each pin $p \in n$ with zero cost, and use an approximation algorithm for the Steiner tree problem in weighted graphs. We thus obtain the following result:

THEOREM 2.1. [Müller et al. 2011] *The oracle functions f_n can be implemented by an approximation algorithm for the Steiner tree problem in weighted graphs.*

For finding such Steiner trees, we use Algorithm 1 which guarantees an easily provable approximation ratio of $2 - 2/|W|$. In practice, however, we observe much better ratios on almost all nets (cf. Section 5.3).

The main subroutine of Algorithm 1 is Dijkstra's shortest path algorithm, which we implemented with various well-known speed-up techniques, including a variant of goal-orientation with landmarks [Goldberg and Harrelson 2005]. The average time needed by Algorithm 1 is about 0.3 milliseconds on our test instances.

2.3. Algorithm for Min-Max Resource Sharing

We use the fastest known algorithm for the MIN-MAX RESOURCE SHARING PROBLEM, due to Müller et al. [2011]. It improves on earlier algorithms by [Grigoriadis and Khachiyan 1994, 1996] (whose algorithms worked only for $\sigma = 1$) and Jansen and Zhang [2008]. The algorithm uses ideas proposed previously for the special case when all functions g_n

ALGORITHM 1: Path Composition Steiner Tree Algorithm

Require: A connected undirected graph G with edge costs $c : E(G) \to \mathbb{R}_{\geq 0}$, and a terminal set $W \subseteq V(G)$ with $|W| \geq 2$.
Ensure: A Steiner tree T for W in G

Set $K := W$ and $F := \emptyset$.
while (K, F) has more than one connected component **do**
  Pick a connected component C of (K, F).
  Find a minimum cost path P from $V(C)$ to $K \setminus V(C)$ in G.
  Set $K := K \cup V(P)$ and $F := F \cup E(P)$.
end while
Return $T := (K, F)$.

ALGORITHM 2: Resource Sharing Algorithm

Require: $\mathcal{R}$ and $\mathcal{N}$, convex functions $g_n^r : \mathcal{B}_n \to \mathbb{R}_{\geq 0}$ $(n \in \mathcal{N}, r \in \mathcal{R})$, and oracle functions $f_n : \mathbb{R}_{\geq 0}^{\mathcal{R}} \to \mathcal{B}_n$ $(n \in \mathcal{N})$ satisfying $\sum_{r \in \mathcal{R}} y_r g_n^r(f_n(y)) \leq \sigma \operatorname{opt}_n(y)$ for all $y \in \mathbb{R}_{\geq 0}^{\mathcal{R}}$ $(\sigma \geq 1)$. $\epsilon > 0$, $t \in \mathbb{N}$.
Ensure: For each $n \in \mathcal{N}$ a convex combination of vectors in $\mathcal{B}_n^{\text{int}}$, given by $\sum_{b \in \mathcal{B}_n^{\text{int}}} x_{n,b} b$. A cost vector $y \in \mathbb{R}_{\geq 0}^{\mathcal{R}}$.

Set $y_r := 1$ for each $r \in \mathcal{R}$.
Set $x_{n,b} := 0$ for each $n \in \mathcal{N}$ and $b \in \mathcal{B}_n^{\text{int}}$.
for $p := 1$ **to** t **do**
  for $n \in \mathcal{N}$ **do**
    Set $b := f_n(y)$.
    Set $x_{n,b} := x_{n,b} + 1$.
    for $r \in \mathcal{R}$ with $g_n^r(b) > 0$ **do** $y_r := y_r \cdot e^{\epsilon g_n^r(b)}$.
  end for
end for
Set $x_{n,b} := \frac{1}{t} x_{n,b}$ for each $n \in \mathcal{N}$ and $b \in \mathcal{B}_n^{\text{int}}$.

are linear, and in particular the multicommodity flow problem [Garg and Könemann 2007; Plotkin et al. 1995; Young 2001].

The algorithm is surprisingly simple, and it is even more simple in our case where we have $g_n^r(b) \leq 1$ for all $b \in \mathcal{B}_n^{\text{int}}$ that we encounter. The core algorithm works in t phases. In each phase, a solution for each net $n \in \mathcal{N}$ is computed by applying Theorem 2.1, using Algorithm 1 to implement the oracle function f_n. Resources become more expensive as they are used. Finally, we take the average of all these solutions. See Algorithm 2 for a detailed description.

If $\frac{1}{2} \leq \lambda^* \leq 1$ and $t = \lceil \frac{96 \ln |\mathcal{R}|}{\omega^2} \rceil$ and $\epsilon = \frac{\omega}{12}$ holds, the algorithm returns a $\sigma(1+\omega)$-approximate solution for any $\omega > 0$. Otherwise we scale all resources, for instance, by binary search (see [Müller et al. 2011] for details, and how to bound λ^*). The result is a fully polynomial-time approximation scheme relative to σ, that is, for any given $\omega > 0$ we obtain a solution $b_n \in \mathcal{B}_n$ $(n \in \mathcal{N})$ with $\max_{r \in \mathcal{R}} \sum_{n \in \mathcal{N}} g_n^r(b_n) \leq \sigma(1+\omega)\lambda^*$.

THEOREM 2.2. [Müller et al. 2011] *If our oracle computes solutions within a factor σ of optimal in time θ, then Algorithm 2 (with appropriate setting of t and ϵ, and a scaling framework if needed) computes a $\sigma(1+\omega)$-approximate solution in $O(\theta \log |\mathcal{R}|(|\mathcal{N}| + |\mathcal{R}|)(\log \log |\mathcal{R}| + \omega^{-2}))$ time.*

In practice we do not need scaling because we know $\frac{1}{2} \leq \lambda^*$ a priori, and we are not interested in a near-optimal solution if λ^* is significantly larger than 1. We found that parameter values $t = 125$ and $\epsilon = 1$ work well. Moreover, there are various speed-up

techniques that improve runtime significantly in practice. For example, we do not need the oracle if we can reuse the previous solution because it is still known to be good. In addition, the algorithm can be parallelized, and a highly scalable parallelized version has been implemented as part of BonnRoute. In Müller [2009] we compared our algorithm to Albrecht [2001] and showed that it gives significantly better results in much shorter runtime.

2.4. Randomized Rounding, Rip-up and Reroute

By Algorithm 2 we obtain for each net n an element of $\mathcal{B}_n$, more precisely a convex combination $\sum_{b\in\mathcal{B}_n^{\text{int}}} x_{n,b}b$ of the elements in $\mathcal{B}_n^{\text{int}}$. Since we eventually need an element of $\mathcal{B}_n^{\text{int}}$ for each net n, it is natural to apply randomized rounding, choosing each $b \in \mathcal{B}_n^{\text{int}}$ with probability $x_{n,b}$, for each n independently. A bound on the degradation of the solution introduced by randomized rounding was given by Raghavan and Thompson [1987] and strengthened later by Müller et al. [2011].

In practice, only few violations occur, that is, if $\hat{b}_n$ is the solution picked by randomized rounding for $n \in \mathcal{N}$, we have $\sum_{n\in\mathcal{N}}(g_n(\hat{b}_n))_r > 1$ only for a small number of resources $r \in \mathcal{R}$. These violations can be eliminated easily by postprocessing ("ripup and reroute") using standard heuristic techniques: first, we repair most of them by rechoosing alternative elements $b \in \mathcal{B}_n^{\text{int}}$ with $x_{n,b} > 0$ for nets $n \in \mathcal{N}$ that use overutilized resources. After this step, to reach feasibility, only for very few nets new routes have to be found that are not part of the output of Algorithm 2. In our experiments, the number of postprocessing route changes was less than 10 % of the number of nets on each chip. Almost all of them were done in the rechoosing step, and at most five new routes were generated on any of our testcases. Moreover, only little time is needed in postprocessing, as our experimental results in Section 5.3 show.

2.5. Global Routing Edge Capacities

Computing edge capacities of the global routing graph G is key to obtaining results that accurately reflect the routability of an instance and that guide the detailed routing step correctly.

For a via edge $e = \{(t, l), (t, l + 1)\}$ of the global routing graph, the capacity $u(e)$ is simply the number of vias from layer l to layer $l + 1$ that can be placed simultaneously in tile t while observing all minimum distance constraints. For edges within one layer it is not so straightforward to determine capacities.

To estimate the capacity $u(e)$ of an edge $e = \{(v, l), (w, l)\} \in E(G)$, we count the usable vertices of the track graph (cf. Section 3.5) in the tile areas of v and w between the coordinates of the tile centers c_v and c_w in preferred direction after extending each blockage by a small constant value in preferred direction. Then we divide the result by the number of vertices on a track between c_v and c_w.

We use two important refinements of this simple capacity estimation. First, connections within a tile have to be accounted for as these are seen as connected when constructing the coarser global routing graph. Therefore prior to global routing capacity estimation we run our detailed router on nets whose pins are contained in a single tile, allowing it to route within a slightly larger area than that of the tile. In addition, intra-tile connection lengths of longer nets are estimated by their Steiner lengths and edge capacities are reduced accordingly, in a similar way as in Wei et al. [2012].

The second refinement considers stacked vias in global routing. A stacked via from layer l to layer $l + 2$ of course also consumes space on layer $l + 1$, and reduces the available space for other nets on this layer. Hence, it contributes to intra-tile resource consumption which cannot be neglected in global routing [Alpert et al. 2010]. Since the expected capacity reduction is sublinear in the number of stacked vias, a preprocessing

step is done in BonnRoute to estimate this value for different numbers $k \in \mathbb{N}$ of stacked vias of size (or minimum area requirement) $p \in \mathbb{N}$ and a normalized region size. This is done by counting the number of possibilities to choose k disjoint sets of p subsequent vertices in x-direction in a two-dimensional lattice such that at most a given number of vertices is chosen in each column. From the resulting numbers, the expected maximum number of selected vertices in a column of the lattice is computed, which is taken as a rough approximation of the reduction of the capacity caused by k disjoint stacked vias placed uniformly at random within the given region.

On wiring layers with via pads extending to neighboring routing tracks, vias block more routing resources. This is accounted for by appropriate scaling.

3. REPRESENTING ROUTING SPACE

An efficient representation of routing space is a key ingredient of BonnRoute. We use a spatial data structure called *shape grid*, described in Section 3.3, for diff-net rule checking. As most of the routing is done on predefined routing tracks, we obtain even faster query times by storing precomputed data for a restricted set of locations based on these tracks. This data is maintained in the *fast grid* data structure presented in Section 3.6. In addition, BonnRoute has a data structure for representing routing space in a way that takes same-net rules into account. We describe it in Section 3.8.

3.1. Diff-net Rules

Wires belonging to different nets have to satisfy certain minimum distance requirements. The required distance between two shapes can be described as a nondecreasing function of their *widths* and common *run-length*. In this context, the width at some point p of a rectilinear polygon is defined by the edge-length of a largest enclosed square covering p. The common run-length of two shapes in x- or y-direction, respectively, is the length of the intersection of the projections of both shapes to the corresponding coordinate axis. Most minimum distance rules are based on the ℓ_2 metric, but there are also rules on minimum horizontal or vertical distance. Some rules apply only to pairs of shapes with positive run-length. Most minimum distance rules affect only shapes on the same layer, but an important exception are *inter-layer via rules* that prescribe minimum distances between vias in adjacent via layers.

Increased spacings are often required if one of the shapes has a so-called *line-end*, which is an edge between two convex vertices closer than some threshold to each other. A particular difficulty with this is that during construction of a route for a net it is often not clear if there will be a line-end at some point or not, as this depends on the shapes subsequently added to the route. In fact, current line-end rules imply that minimum distance rule violations can occur by removing shapes from a DRC-clean routing.

The approach taken in BonnRoute to address this problem is to always assume wires to have line-ends in preferred direction independently of how they are continued. Although this is pessimistic, it does not waste too much routing space in current technologies: if a wire in preferred direction ends without a line-end, then the extra line-end extension that we introduce is almost always a subset of a via or jog shape and hence consumes no additional space.

On the other hand, jogs are not assumed to have line-ends, which is optimistic and can cause minimum distance violations. The rationale behind this is that otherwise a jog would be assumed to interfere with neighboring tracks, which usually is not the case, especially if it is continued with a wire running in preferred direction (see Figure 2).

In current technologies there are many additional minimum distance rules which are also very difficult to handle exactly. For example, rules that only apply to shapes with run-length greater than some specific threshold are problematic because exact

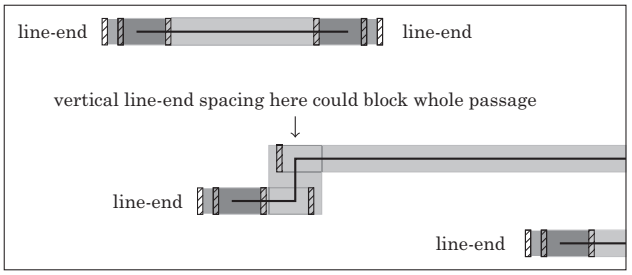

Fig. 2. Set of wire shapes (light gray) and via shapes (dark gray) and their corresponding stick figures (black line segments) on a plane with horizontal preferred direction. We extend *every individual* shape except jog shapes by the additional line-end spacing in preferred direction (hatched rectangles). Note that in the shown situation only the extensions where we actually have a line-end consume additional space. All other extensions are contained in other shapes anyway. For line-ends orthogonal to the preferred direction, however, this would not hold. Therefore, we are optimistic and do not add line-end extensions in this case.

run-lengths are often not known during the construction of a route. Some minimum distance rules even depend on the shapes of more than two nets. In practice, however, such rules also can be handled well by appropriate conservative or optimistic modeling. To clean up eventually remaining DRC violations resulting from this optimism, we use an industry standard router cf; Section 5.3.

3.2. Wire Models, Via Models, and Wire Types

Wire and via models map one-dimensional abstractions of wires, called *stick figures* (see Figure 2), to the metal shapes induced by them, plus their minimum distance requirements to other shapes. This mapping is encoded by an axis-parallel rectangle and a *shape class* used to determine the minimum distance requirements. The metal shape of a wire is the Minkowski sum of the stick-figure and the shape of the wire model. As a via induces shapes in three layers (a bottom and top *pad* in the wiring layers, and a *cut* shape in the intermediate via layer), a via model consists of three axis-parallel rectangles and shape classes. If an inter-layer via rule applies, the projection of the via cut to the next higher via layer is added to the via model. This allows for efficient checking of inter-layer via rules within a via layer. A *wire type* is a function that maps wiring layers to pairs of wire models (for preferred and non-preferred routing direction), and via layers to via models. All wires and vias can thus be represented efficiently by stick-figures and wire types, respecting requirements of varying wire size and spacing over different metal layers.

Note that in this model for efficiency reasons we only consider minimum distance requirements between individual shapes instead of whole rectilinear polygons comprised of multiple shapes. By defining shape classes appropriately, this is a feasible approach in practice [Schulte 2012].

3.3. Shape Grid

The shape grid efficiently stores all relevant data about blockage, wire, and via shapes. The information in the shape grid allows to decide whether a wire can be placed somewhere without violating minimum distance rules. If not, it allows to find out if there is a set of shapes that can be removed such that the answer becomes positive.

The shape grid partitions the chip area on each wiring layer and on the *via layers* into axis-parallel rectangular *cells*, such that each of them has at most one neighbor cell to the left, right, bottom and top, respectively. The size of the cells is small enough such that shapes of different nets cannot be legally present in the same cell.

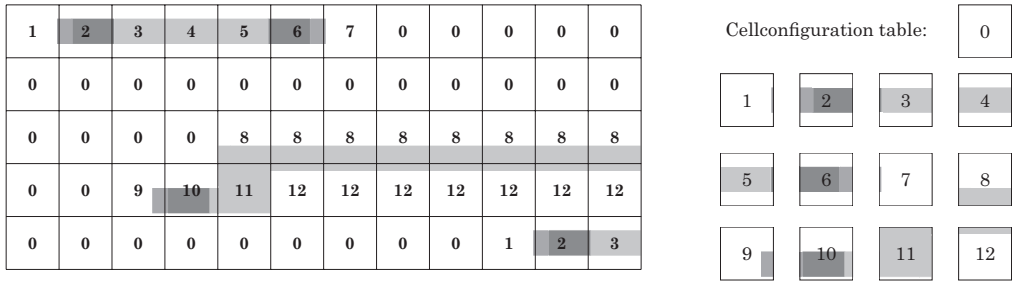

Fig. 3. Shape grid cell configurations and their configuration numbers resulting from the wiring in Fig. 2. For storing them, sequences of identical numbers in preferred direction are merged to intervals. Intervals of cells with configuration 0 are not stored, so there are 15 intervals in the shape grid in this small example. Here only two of them contain more than one cell. The cell configuration table containing the 13 different cell configurations of this example is illustrated in the right part of the figure.

In each cell, all intersections of shapes with its area are stored with coordinates relative to an anchor point (e.g., its center), plus their respective type and minimum distance requirements. Since this *cell configuration* is typically identical in a large number of cells, we store this data indirectly by a *cell configuration number* which is used as an index into a lookup table that stores the actual data.

The data volume is further reduced by grouping neighboring cells in preferred routing direction to *intervals* which are stored in an AVL-tree in each row or column of cells, depending on the preferred routing direction (see Figure 3 for an example).

Moreover, for each nonempty interval we store the net that the shapes of this interval belong to if they are removable. The type and net identifier of a shape are used to assign a *ripup level* to it. The ripup-and-reroute algorithm can then be restricted to rip out shapes of at most some specified ripup level in order to avoid rerouting of critical connections.

3.4. Distance Rule Checking

The distance rule checking module in BonnRoute serves as interface between the shape grid and other components of BonnRoute, most importantly the path search module (cf. Section 4.1). It gets a location l and a set W of wiretypes as input and returns the subset of wire- and via models of wiretypes in W that are allowed to be placed at l. Based on the minimum distance requirements of the wire and via models of wiretypes in W, it computes this information by querying all intervals $\mathcal{I}$ from the shape grid that may contain shapes with distance rule violations to any of the models placed at location l. In addition, it returns a maximal horizontal and a maximal vertical interval of locations, each containing l, in which the same subset of wire and via models is allowed, and for which this can be determined without querying additional intervals $I \notin \mathcal{I}$ from the shape grid. Furthermore, the distance rule checking module can report nets whose (partial) removal increases the set of allowed wire or via models at the given location.

3.5. Routing Tracks

Most of the wiring on a routing layer has the minimum possible width. Such *standard wires* can be packed best with a net-by-net routing approach if they are aligned on *tracks* in preferred direction, whose distance is the minimum wire width plus the minimum required distance of two wires.

As there are usually blockages (e.g., power supply) on a routing layer, a natural objective when defining the tracks is to maximize the total usable track length: Given

a layer L, its minimum wiring pitch p_L, and a set $\mathcal{A}$ of axis-parallel closed rectangles with pairwise disjoint interior in which a standard wire can be routed without minimum distance violations to other objects, the *track optimization problem* is to find a set T of lines in preferred wiring direction with at least distance p_L between each pair of them, and such that $\sum_{t \in T} \left| t \cap \left(\bigcup_{A \in \mathcal{A}} A \right) \right|$ is maximized.

THEOREM 3.1. [Müller 2009] *The track optimization problem can be solved in* $O(|\mathcal{A}| \log |\mathcal{A}|)$ *time.*

The alignment of routing tracks with pins can be taken into account by adding rectangles to $\mathcal{A}$ which model track positions that allow on-track pin access. To obtain a good packing for wires with width larger than the minimum pitch, wire type variants with different stick-figure offsets can be preassigned to tracks.

The intersection points of routing tracks with tracks projected from neighboring wiring layers define the vertices of our *track graph* in a natural way. Two vertices in this graph are connected by an edge if two of their coordinates are equal and the straight line connecting them does not intersect any other vertex or wiring layer. Wire segments or vias are called *on-track* if their stick figures run only on the straight lines corresponding to edges of the track graph.

3.6. Fast Grid

As the vast majority of wires shares a very small set of wire types in practice, BonnRoute stores continuously updated data computed by the distance rule checking module for a restricted set of wire types to be placed at on-track locations. This *fast grid* can look up quickly if a given on-track via or wire segment of one of these types can be used without distance violations to surrounding shapes, which is considerably faster than querying the distance rule checking module each time. For this reason, the on-track path search uses this data structure for the frequently used wire types. Only queries for other, less frequently used, wire types or for off-track locations have to use the distance rule checking module directly.

If there is only on-track wiring, legality of a wire whose stick figure connects two neighboring vertices of the track graph is implied by legality of the zero-dimensional stick figures at each of the vertices. Because of this, the fast grid stores information for vertices, but not for edges. This allows to group longer sequences of vertices with identical data to intervals. If data was stored for vias or jogs, more changes would occur along a track in preferred direction, resulting in a higher interval count. For the checking of cut shapes that is necessary for vias in addition to checking their bottom and top pads, the fast grid also stores intervals in via layers that run in the same direction as the next lower wiring layer. If legality of a wire on an edge $\{v, w\}$ cannot be deduced from legality at v and w because off-track wires or other shapes are present in the vicinity, an extra bit at one of the vertices v and w encodes this. If this bit is set.

Figure 4 illustrates the fast grid data structure. Per wire type, legality information for the wire models used in preferred and non-preferred wiring direction, respectively, has to be stored, plus information for via bottom and top pads, that is, 4 types of shapes. Hence, 12 bits per wire type and interval suffice to encode legality of these shapes in eight different ripup levels. A double word, that is, 64 bits, therefore suffices to store information for five wire types. In via layers fewer bits are required as there are only two types of shapes (via cut shapes and their projections for inter-layer via distance checking).

In our experiments, 97.89% of the queries to the distance rule checking module can be answered using the information stored in the fast grid, resulting in an overall speedup of 5.29 of on-track path search in BonnRoute.

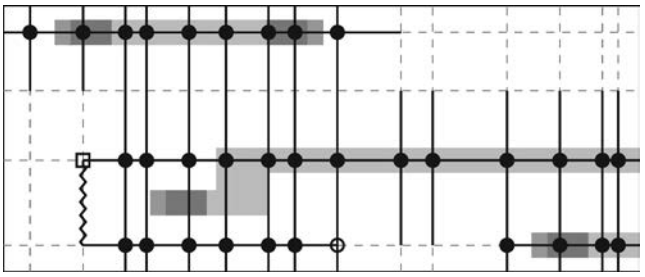

Fig. 4. The fast grid: dashed lines represent x- and y-edges of the track graph, with vertices at their intersections. For some given wiretype, circles mark vertices at which no jog can start (unless ripup is allowed); if a circle is filled, then no wire in preferred direction can start at this vertex either. Thick black edges represent edges that are unusable with the given wiretype. Usability of an edge can be deduced from the vertex information in all cases except the zigzag edge: For this edge, we set a bit at one of its incident vertices (marked by a square in the picture) to indicate that it has an incident edge whose usability cannot be deduced from the vertices it connects, so the shape grid must be queried here. Storing sequences of equally marked vertices in preferred direction yields 6 intervals in the fast grid in this example.

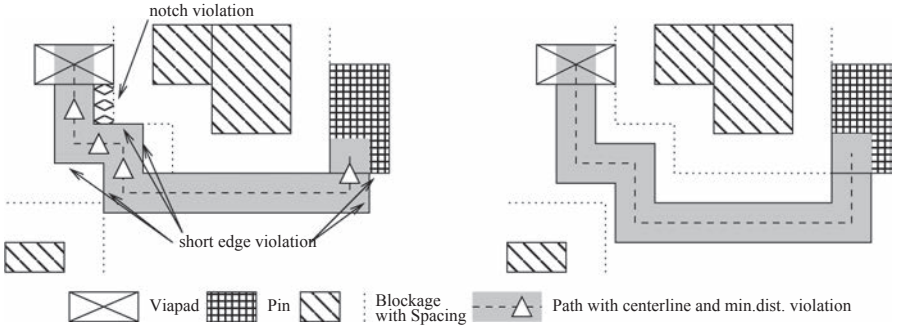

Fig. 5. Example showing geometric design rule violations for geometric shortest paths without minimum segment length (left). The path is sought to connect the viapad to the pin. Taking minimum segment lengths into account, a feasible path (right) obeys these rules.

3.7. Same-Net Rules

Most same-net design rules are in place to avoid geometric configurations with features below the lithographic capabilities and resolution, and to ensure space for optical proximity correction (OPC). Additionally, there are rules stemming from yield considerations.

The list of same-net design rules is usually very long, we briefly discuss the most important examples.

—Notch rules: Even within the same path, non-adjacent segments have to obey distance requirements.
—Short edge rules: For any pair of adjacent edges in the boundary polygon of any metal area, at least one of the two edges must have a given minimum length.
—Minimum area rules: Every connected metalized polygon on a layer must have a certain minimum area, independent of its actual shape.

These rules are often violated in geometric shortest paths and Steiner trees; see also Figure 5. It is particularly important to obey them in pin access because of the often irregular geometries of pins and limited space available for fixing errors by postprocessing.

ALGORITHM 3: Blockage_Grid_Vertical

Require: blockage border coordinates $\underline{x}_j, \overline{x}_j$ $(j = 1, \ldots, n)$, $s, t \in \mathbb{R}^2$, $\tau > 0$
Ensure: list $\overline{\mathcal{V}}$ of horizontal coordinates
1: $\mathcal{V} := \{x_s, x_t, -\infty, \infty\}$; $\overline{\mathcal{V}} := \varnothing$
2: **for** $j = 1$ **to** n **do** add $\underline{x}_j, \overline{x}_j$ to $\mathcal{V}$
3: sort $\mathcal{V}$ and remove duplicates
4: **for** $i = 1$ **to** $|\mathcal{V}| = |\{x_1, \ldots, x_{n_v}\}|$ **do**
5: $c_{\min} := i - \min\{k \in \mathbb{N} \mid x_{i-k} - x_{i-k-1} \geq 4\tau\}$
6: $c_{\max} := i + \min\{k \in \mathbb{N} \mid x_{i+k+1} - x_{i+k} \geq 4\tau\}$
7: **for** $\lambda \in \mathbb{Z}$ with $x_i + \lambda \cdot u \in [x_{c_{\min}} - 2\tau, x_{c_{\max}} + 2\tau]$ **do**
8: add $x_i + \lambda \cdot \tau$ to $\overline{\mathcal{V}}$
9: **end for**
10: **end for**
11: sort $\overline{\mathcal{V}}$ and remove duplicates

3.8. Blockage Grid for Off-Track Wiring

The main goal of the blockage grid is to support fast search for shortest (off-track) paths that avoid the most important classes of same-net rule violations. Most same-net rules can be mapped to requirements on minimum segment lengths in the constructed path [Nieberg 2011]. The BonnRoute approach to DRC-clean off-track wiring hence is as follows.

Suppose that we only have a single requirement stating that each segment of a wire must have length at least $\tau > 0$. Call a rectilinear path *τ-feasible* if each of its segments has length at least τ and does not intersect the interior of any obstacle. We construct a data structure called *blockage grid*, which allows for finding shortest τ-feasible paths.

Starting with original lines of a Hanan grid we add additional vertical and horizontal lines at distances that are multiples of τ (cf. Algorithm 3). However, since we only add lines as long as two consecutive lines of the original Hanan grid are closer than 4τ to one another, we can bound the total number of vertices, that is, crossing points of lines, by $O(n^4)$,where n denotes the number of rectangles representing the blockages. The following result then shows that these vertices suffice.

THEOREM 3.2. [Maßberg and Nieberg 2012] *Consider a set of n rectilinear obstacles in the plane, a distance $\tau > 0$, and $s, t \in \mathbb{R}^2$. If there exists a τ-feasible path from s to t, then there is also a shortest τ-feasible path from s to t all whose segments have vertices of the blockage grid as endpoints.*

In order to respect the minimum segment length τ during the path search, we construct a path-preserving digraph G on which we can run a regular shortest path algorithm. We construct G from the blockage grid as follows. G contains up to four vertices for each vertex of the blockage grid; one collecting incoming arcs of each direction. Arcs of G connect neighboring vertices of the blockage grid where this does not induce a bend together with the incoming direction. Additional arcs connect each vertex to the (at most two) nearest vertices at distance $\geq \tau$ perpendicular to the incoming direction. This way, each bend is followed by a long arc as G does not contain short arcs after bends. More sophisticated rules like short edge avoidance can be added as well by introducing corresponding edges in G or removing edges that induce a violation.

Storing the blockage grid coordinates only takes $O(n^2)$ space. For a multilayer path search, we have a blockage grid for each wiring and via layer. Coordinates are also induced by blockages of neighboring layers. Vertices of two adjacent wiring layers are connected if a via is allowed here.

4. DETAILED ROUTING

The core task of connecting two components of a net is solved in BonnRoute by an on-track shortest path search operating on intervals of the track graph described in Section 4.1. If the source or target component cannot be connected on-track, it is combined with off-track pin access, which we present in Section 4.3. Finally, Section 4.4 discusses the overall procedure to connect an entire net.

4.1. On-Track Path Search

The on-track path search finds a shortest path between two sets of vertices in the track graph corresponding to connected components of a net.

We use a directed version G_T of the track graph, where each undirected edge is replaced by two oppositely directed edges. We identify each vertex with its coordinates (i.e., $V(G_T) \subseteq \mathbb{R}^3$), and define edge costs

$$c((v,v')) = \begin{cases} \gamma_{\{z,z'\}} & \text{if } (v,v') \text{ is a via} \\ \beta_z \cdot ||v - v'||_1 & \text{if } (v,v') \text{ is a jog} \\ ||v - v'||_1 & \text{otherwise} \end{cases}$$

for two neighboring vertices $v := (x,y,z)$ and $v' := (x',y',z')$ of G_T, where $\beta_z, \gamma_{\{z,z'\}} \in \mathbb{Z}_{>0}$ are parameters that encode penalty costs for wires running in non-preferred direction and for vias, respectively.

The input for one call of the on-track path search is a *source set* $S \subseteq V(G_T)$, a *target set* $T \subseteq V(G_T)$, a *routing area* $R \subseteq V(G_T)$ with $R \cap S \neq \emptyset$ and $R \cap T \neq \emptyset$, and a wiretype w. The restriction to one wiretype is just a simplification, more wiretypes can also be handled in a straightforward way.

The routing area R induces a graph $G_T[R]$ to which the on-track path search is restricted. Let G be the subgraph of $G_T[R]$ that results from removing all edges whose usage with wiretype w would introduce a diff-net violation. We do not store G explicitly, but rather query the distance rule checking module (cf. Section 3.4) for usable edges as needed.

The output of the on-track path search is a shortest S-T-path P in G (with respect to c) which by definition of G corresponds to a shortest on-track wiring connection without diff-net violations from S to T with wiretype w.

When routing a modern chip we have to perform millions of on-track path searches. Thus even a runtime linear in $|V(G)|$ would be too slow. Our on-track path search uses a Dijkstra-based algorithm [Dijkstra 1959] with two major speed-up features: a *future cost* (similar to the A* heuristic [Hart et al. 1968]) to reduce the number of labeling steps, and merging vertices to so-called *intervals* to speed up sequences of labeling steps. In the following we describe these features.

A *future cost* is a function $\pi : V(G) \to \mathbb{N}$ satisfying $c_\pi((v,w)) := c((v,w)) - \pi(v) + \pi(w) \geq 0$ for all $(v,w) \in E(G)$ and $\pi(t) = 0$ for all $t \in T$. One easily observes that $\pi(v)$ is a lower bound on the distance from v to T in (G,c) for each $v \in V(G)$. Let G' result from G by adding a vertex s and an edge (s,s') with $c_\pi((s,s')) := \pi(s')$ for $s' \in S$. Then all shortest S-T-paths in (G,c) are also shortest s-T-paths in (G',c_π) and vice versa. In practice, better lower bounds π result in fewer labeling steps if we apply Dijkstra's algorithm to (G',c_π).

We introduce two future costs. The simpler one is given by $\pi_H((x,y,z)) := lb_{\text{wire}}(x,y) + lb_{\text{via}}(z)$, where $lb_{\text{wire}}(x,y) := \min_{(x_t,y_t,z_t)\in T}(|x - x_t| + |y - y_t|)$ and $lb_{\text{via}}(z)$ is the minimum cost for a via connection from plane z to a plane containing a target location [Hetzel 1998].

Let T_{rect} be a set of axis-parallel rectangles covering exactly the vertices in T projected to one plane. Then $lb_{\text{wire}}(x,y)$ can be queried for each vertex (x,y,z) in time

$O(\log |T_{\text{rect}}|)$ by point location ([Preparata and Shamos 1985]) applied to the ℓ_1 Voronoi diagram of the rectangles in T_{rect} by spending $O(|T_{\text{rect}}| \log |T_{\text{rect}}|)$ preprocessing time [Papadopoulou and Lee 2001; Gester 2009].

The main advantage of π_H is its simplicity and fast computability, since the runtime only depends on $|T_{\text{rect}}|$ and not on the size of G. However, not considering the structure of G in π_H can lead to a big gap between $\pi_H(v)$ and the length of a shortest v-T-path.

To avoid this problem, Peyer et al. [2009] proposed a blockage-aware future cost π_P which computes shortest paths to T from all nodes in a supergraph G' of G that captures the structure of G quite well by ignoring small blockages and keeping larger blockages. In particular, G' is chosen such that $\pi_P \geq \pi_H$. This results in considerably fewer labeling steps in Dijkstra's algorithm, but the runtime for computing π_P is usually higher than for computing π_H. Therefore we use π_P only if the global routing for this connection already contains a large detour, and π_H otherwise. The future cost used in path search is denoted π in the following.

The generalization of Dijkstra's algorithm to intervals of vertices was proposed by Hetzel [1998] for the special case of equidistant routing tracks which match in all layers with the same preferred direction. We partition $V(G)$ into a set of maximal intervals $\mathcal{I}$ such that for each $I \in \mathcal{I}$ all vertices in I are on the same track (and hence on the same layer z), $G[I]$ is connected, and the following properties are satisfied for all $v, v' \in I$:

(i) $|\pi(v) - \pi(v')| = ||v - v'||_1$.
(ii) If $(v, v'') \in E(G)$ is a jog or via, then $(v', v'' + v' - v) \in E(G)$.

We do not store $\mathcal{I}$ explicitly, but create intervals on the fly as needed. Instead of storing a label at each vertex of G, we maintain a set of labels $\Delta(I) \subset I \times \mathbb{N}$ for each interval I. In practice we often have $|\Delta(I)| \ll |I|$. Each label $(v, \delta) \in I \times \mathbb{N}$ carries the information that there is an s-v-path of length δ which we already found, just as in the classical Dijkstra algorithm. Moreover, this induces an s-v'-path of length $d_{(v,\delta)}(v') := \delta + ||v - v'||_1 - \pi(v) + \pi(v')$ for all $v' \in I$. We do not add a label (v', δ') to $\Delta(I)$ if it is *redundant*, that is, $\delta' \geq d_{(v,\delta)}(v')$ for some $(v, \delta) \in \Delta(I)$. The length of a shortest s-v'-path known so far for $v' \in I$ is $d(v') := \begin{cases} \min_{(v,\delta)\in\Delta(I)} d_{(v,\delta)}(v') & \text{if } \Delta(I) \neq \emptyset \\ \infty & \text{else} \end{cases}$. By our definitions, this function is monotone and consists of alternating pieces of slope 0 and 2, or slope 0 and -2. Thus $J_I(\delta) := \{v \in I \mid d(v) = \delta\}$ is either empty or a single interval.

See Algorithm 4 for a sketch of the interval-based shortest path algorithm with future cost and Figure 6 for an illustration of a path search. The algorithm was generalized by Peyer et al. [2009] to more general vertex sets. We get the following result.

THEOREM 4.1. [Hetzel 1998; Humpola 2009; Peyer et al. 2009] *A shortest S-T-path P in a track graph G with edge costs $c : E(G) \to \mathbb{N}$ as above and future cost $\pi : V(G) \to \mathbb{N}$ can be found in time $O(\min\{(\Lambda + 1)|\mathcal{I}| \log |\mathcal{I}|, |V(G)| \log |V(G)|\})$, where Λ is the cost of P w.r.t. c_π and $\mathcal{I}$ is the set of intervals representing G.*

According to recent experiments on 22 nm chips, labeling intervals instead of single nodes speeds up the path search by at least a factor of 6.

4.2. Extensions

To support *ripup-and-reroute*, the on-track path search can be driven in a mode where it can use some vertices of the track graph that are normally unusable. This can apply to vertices with minimum distance violations to shapes that are allowed to be modified. However, high extra costs are imposed on intervals containing such vertices. These costs increase over time at the same vertex to avoid cyclic rip-up sequences (see Section 4.4).

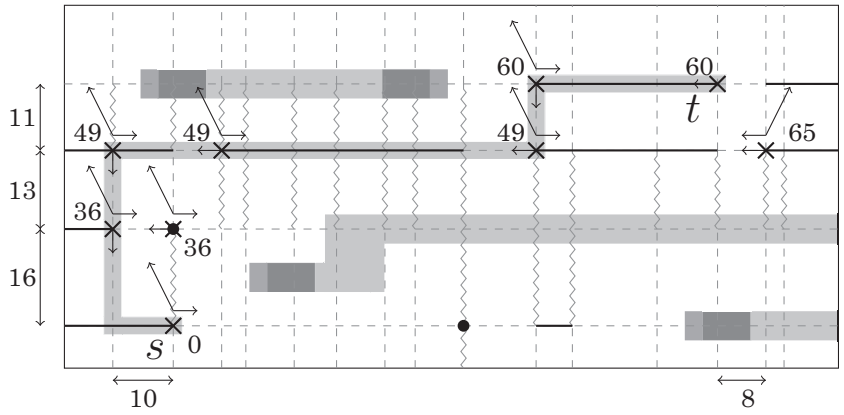

Fig. 6. Completed path search from source s to target t with cost function c_{π_H}. We assume $\beta_z = 2$. Dashed lines represent x- and y-edges of the track graph, the preferred routing direction is horizontal. Thick black lines and black circles represent all usable intervals and zigzag lines represent unusable edges in non-preferred routing direction adjacent to them. A label (v, δ) is represented by a cross and a number giving the function value at the breakpoint of the sketched label function d. The directions from which the labels were induced are illustrated by arrows. The intervals on the top right are split because we need a monotone future cost on each interval (property i)). The splitting of other intervals is due to property ii) because of different allowed jog directions (cf. zigzag lines). A shortest s-t-path corresponding to the labeling is shown in gray.

If there is unused space in a region, *wire spreading* can improve timing and manufacturing yield (both by reducing the probability of extra material defects [Müller 2006], and by increasing the number of vias that can be replaced by larger, that is, more robust, vias in postprocessing). To do this in BonnRoute, the on-track path search imposes extra costs on intervals that should be kept free, based on congestion observed by global routing.

4.3. Off-Track Pin Access

As most pins are not aligned with the grid, they require off-track pin access [Nieberg 2011] which is based on the off-track path search and blockage grid presented in Section 3.8.

For each such pin, we construct a *catalogue* of several DRC clean paths connecting it to on-track points within a small radius. Among these, we compute one primary access path for each pin such that the set of these paths for the pins of each circuit forms a *conflict-free solution* (see also Figure 7), that is, is DRC-clean also with respect to diff-net rules. We add the primary access paths of the conflict-free solution to the routing space as reservations before actually starting to route: this way, newly added wires do not invalidate the precomputed conflict-free solution. We do all this in a preprocessing step, but also have the functionality to construct additional paths on-the-fly when needed.

Although there may be millions of circuits placed on the chip, containing the vast majority of pins, there are only a few thousand prototype circuits in a library. Furthermore, non-circuit blockages and wires, for instance, stemming from power supply or pre-designed clock nets, often have a regular structure. The preprocessing step of constructing the catalogues for the pins exploits this.

We partition the set of circuits into *circuit classes*. Expanding the bounding area of an instance of a circuit and taking all shapes influencing this area and the track coordinates into account, we can identify geometrically equal situations on the chip area

ALGORITHM 4: Interval-based shortest path algorithm

Require: S, T, and implicitly given G, $\mathcal{I}$ (with $\Delta(I) := \emptyset\ \forall I \in \mathcal{I}$ initially), c, and π as described above
Ensure: A shortest S-T-path in G wrt. c_π

for all $I \in \mathcal{I}$ with $I \cap S \neq \emptyset$ **do**
  $v_{\max} := \operatorname{argmax}_{v \in I \cap S}(\pi(v))$
  $\Delta(I) := \{(v_{\max}, \pi(v_{\max}))\}$
end for
repeat
  $\delta := \min\{d(v) \mid v \in V(G)\}$
  if $\delta = \infty$ **then**
    return no S-T-path existing
  end if
  $\mathcal{F} := \{I \in \mathcal{I} \mid J_I(\delta) \neq \emptyset\}$
  while $\mathcal{F} \neq \emptyset$ **do**
    Choose $I_1 \in \operatorname{argmax}_{I \in \mathcal{F}}(\max_{v \in J_I(\delta)}(\pi(v)))$
    for all $I_2 \in \mathcal{I}$ with $(v_1, v_2) \in E(G)$ for $v_1 \in J_{I_1}(\delta)$, $v_2 \in I_2$ **do**
      Choose such a v_1 with $\pi(v_1)$ maximum
      $\delta_2 := \delta + c_\pi(v_1, v_2)$
      if (v_2, δ_2) is not redundant for $\Delta(I_2)$ **then**
        $\Delta(I_2) := \Delta(I_2) \cup (v_2, \delta_2)$
        if $\delta_2 = \delta$ **then**
          $\mathcal{F} := \mathcal{F} \cup I_2$
        end if
      end if
    end for
    $\mathcal{F} := \mathcal{F} \setminus I_1$
  end while
until $\exists t \in T : d(t) = \delta$
return shortest path by backtracking from t to S

(up to translation, mirroring, and rotation), and we thus collect these configurations into equivalence classes. The preprocessing is then done based on these.

In order to anticipate the effects of local congestion, we are not interested in feasible conflict-free solutions alone, but also take (local) spreading of the paths into account. Knowing that the ongrid path-search adds wiring around the endpoints of the access paths, we evaluate a conflict-free solution based on spreading of endpoints, number of blocked tracks, directions of feasible on-track continuation, and length.

We use a branch-and-bound based enumeration technique called destructive bounding to compute a good conflict-free solution for each circuit class.

4.4. Connecting Nets

The core components of BonnRoute are used to connect the pins of a net as follows. We iteratively pick a connected component of a not yet fully routed net, call it the source component, and construct the source vertex set S passed as input to the on-track path search by

1) adding all track graph vertices at which the component can be connected by on-track wires and vias without same-net rule violations;
2) adding vertices that can be connected to the component by a short off-track path without same-net rule violations.

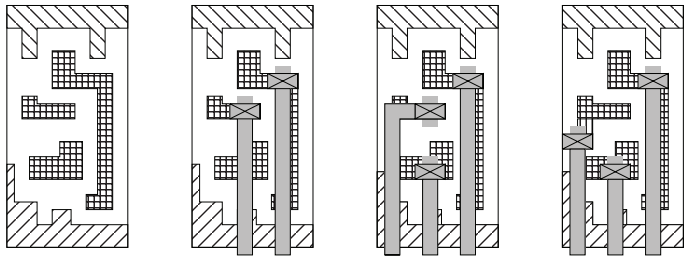

Fig. 7. Examples of different pin access solutions. On the left, a circuit is given with its pins (on layer 1) and blockage structure. The following three figures show three different situations for pin access wiring (grey, on layer 2). First, a situation that may occur by a greedy approach is shown: after connecting the first two pins, the third pin is blocked. The two figures on the right show two conflict-free solutions (using on-track via positions on metal 2 as endpoints). The one on the right is superior and will be chosen by our algorithm.

The vertices added in 2) are in most cases endpoints of off-track pin access paths computed as described in Section 4.3. Of course, these paths are first checked for diff-net rule violations to earlier routed nets. If no or only a small number of access paths remains feasible, we dynamically generate new access paths.

To determine the target vertex set T, we find a shortest path from the selected source component to a different component of the net along the wires obtained from global routing, and generate the routing area R for on-track path search as the union of all global routing tiles intersected by that path (adding tiles in neighboring layers to allow crossing of already existing wiring). For all non-source components that can be connected at vertices in R we add those to T analogously to the construction of S.

The shapes of the components that contributed to S and T, including reserved pin access paths (cf. Section 4.3), are removed temporarily from routing space in order not to block access to these vertices. If an on-track path has been found, we combine it with the corresponding off-track path if present, and then perform a postprocessing step on the resulting path to clean up same-net rule violations which occur particularly where on-track and off-track paths meet, but also within on-track paths because no attention is given to same-net rules there. The resulting path is then added to the net and to the routing space. Also temporarily removed shapes are reinserted into the routing space, with the exception of resource reservations in the components that have been connected.

If no path has been found, we start a ripup sequence allowing both on-track and off-track path searches to violate minimum distances to wires of other nets which then have to be rerouted. If we need to rip out wires, the corresponding components are connected next, again possibly using ripup. A parameter bounds the depth of ripup sequences. If the sequence fails, the original situation is restored. Nets that are still not connected are reconsidered later with higher ripup effort and extended routing area.

5. PRACTICAL ISSUES AND EXPERIMENTAL RESULTS

5.1. Parallelization

The parallelization approaches we use in global and detailed routing are fundamentally different, although both are shared-memory approaches. While our global router allows different threads to work in the same region, in detailed routing we partition the chip area into regions assigned to threads. The main reason for this is that while in global routing *volatility-tolerant* block solvers [Müller et al. 2011] can be used to guarantee the desired approximation ratio even with concurrent access to the same resources by

multiple threads, we want to obey minimum distance rules strictly throughout detailed routing.

Hence, in detailed routing, each thread is allowed to make only changes that do not affect regions assigned to other threads. For instance, if a wire of a certain type has large minimum distance requirements to other objects, it may not be put close to the border of the region assigned to a thread. Because this allows only a subset of connections to be closed, a sequence of partitions is defined such that in each of them the estimated workload in each region is balanced. The number of regions, and hence the number of threads used, is reduced in later partitions to allow longer connections to be closed. In addition, there is a critical net routing step at the beginning of BonnRoute. Nets can be considered critical here from a timing perspective, or because they use wide wires that would be hard to route when the majority of nets has already been routed.

5.2. Combination with External DRC Cleanup

Due to the ever increasing complexity of design rules, the philosophy of BonnRoute is to focus on near-optimum packing of wires (with respect to wiring length, detours, power, or manufacturing yield), and leave DRC cleanup to an external tool. As explained above, BonnRoute does however try to avoid two classes of DRC violations:

1) It leaves almost no violations of diff-net spacing rules; there are very few exceptions to this usually caused by line ends and by combinations of at least three shapes, any two of which do not conflict.
2) DRC violations that need additional space for fixing, for instance, minimum area errors, are avoided as much as possible during BonnRoute.

In addition, many violations of other types of rules can be avoided by preprocessing or fixed by postprocessing. Preprocessing includes, for instance, computation of conflict-free DRC-clean pin access paths, and postprocessing is done immediately after each path search.

The combination of BonnRoute with an industry standard router for DRC cleanup results in competitive numbers of design rule violations, but with much better results in other metrics such as runtime, detours, and number of vias, as the results in Section 5.3 show.

5.3. Experimental results

In this section we compare routing results of an industry standard router (ISR) with our combined flow ("BR + ISR") that we described in Section 5.2, using the same industry standard router for DRC cleanup after BonnRoute. ISR is a current router which is used in practice on many industry designs of former and recent technologies. In contrast to BonnRoute, it uses a track assignment step to cover long distances and then completes the routing in purely gridless fashion.

All experiments were done on a 3.47GHz Intel Xeon machine with 192GB RAM. Both routers used 12 threads. The instances are current IBM designs with different wiretypes and complete rule sets.

Table I reports runtime, memory consumption, netlength, via counts, scenic nets, and error counts. These error counts are composed of the numbers of DRC violations and opens (i.e., number of connected components minus number of nets).

For our combined flow, we also show the runtime needed by BonnRoute itself. Interestingly, the DRC cleanup takes longer than BonnRoute although only local changes are made. Nevertheless, the total runtime of our "BR+ISR" flow is less than half compared to "ISR". The memory consumption of the two flows is about the same on most instances, only on chip 8 "BR+ISR" consumes about twice as much memory as "ISR," yet still much less than available. Although we have a small increase in total number

Table I.
Runtime, memory consumption, netlength, via counts, scenic nets, and number of errors for an industry standard router (ISR), and for our flow of BonnRoute followed by ISR for DRC cleanup (BR+ISR)

Chip	Tech	Time (hh:mm:ss)		Memory	Netlength	#Vias	#Scenic	#Scenic	Errors
	# Nets	BR	Total	(GB)	(m)		($\geq$ 25%)	($\geq$ 50%)	
1	22 nm		2:20:10	6.45	3.08	1,265,662	587	103	18
	120,704 nets	0:17:35	0:35:53	4.97	2.97	932,441	7		28
2	22 nm		2:09:32	6.76	3.56	1,112,724	1,545	1,071	124
	126,239 nets	0:20:08	1:28:38	6.20	3.40	903,760	16		194
3	22 nm		2:04:40	6.24	3.07	1,240,196	695	286	14
	129,653 nets	0:17:19	0:35:23	6.13	2.97	925,221	8	1	25
4	22 nm		2:28:17	5.90	3.37	1,247,778	502	232	15
	135,275 nets	0:26:17	0:42:19	4.98	3.24	944,671	5		3
5	32 nm		2:49:58	9.70	9.91	2,936,452	2,401	1,157	50
	383,940 nets	1:00:35	1:46:37	14.74	9.57	2,399,268	105	6	61
6	22 nm		12:42:42	19.06	14.01	4,252,972	14,829	10,668	459
	438,326 nets	1:09:05	5:28:50	17.55	12.47	3,233,924	1,568	520	506
7	22 nm		8:51:05	13.30	12.31	4,038,686	3,853	2,310	154
	466,155 nets	1:03:56	2:31:47	14.79	11.87	3,184,733	350	193	183
8	32 nm		14:44:44	18.87	38.87	7,769,348	11,516	6,539	111
	961,593 nets	2:37:46	9:59:18	37.36	37.31	6,238,891	2,619	1,285	117
Sum			48:11:08	86.28	88.18	23,863,818	35,928	22,366	945
	2,761,885 nets	7:12:41	23:08:45	106.7	83.80	18,762,909	4,678	2,005	1117

Table II.
Total netlength obtained by BonnRoute's global router over all nets with specified number of terminals on all chips, and the ratio above Steiner length for each class

2 terminals	3 terminals	4 terminals	5–10 terminals	11–20 terminals	>20 terminals
52.48m (1.037x)	8.85m (1.078x)	5.75m (1.101x)	10.81m (1.145x)	1.58m (1.181x)	0.08m (1.182x)

of errors in the "BR+ISR" results (which is not significant in practice), the table shows that the overall quality of the routing obtained by BonnRoute is far superior.

We call a net *scenic* if it has routed wiring length of at least 100 μm and a detour of at least 25% or 50%, respectively, over the length of a Steiner tree with minimum length (for nets with at most 9 terminals [Chu and Wong 2008]) or approximately minimum for nets with 10 or more terminals (obtained by heuristic Steiner tree algorithms). The Steiner trees used as baseline for defining scenic nets are of course identical in the "ISR" and "BR+ISR" rows.

With respect to detours made by our global router, Table II breaks down the achieved netlength by the number of terminals, that is, connected components of a net in its input, and compares it to the Steiner length in each class. As Algorithm 1 is optimal for two-terminal-nets, the detours in the first column are due to congestion mitigation. The other columns also show that Algorithm 1 gives much better approximation ratios than the theoretical bound for most nets in practice. This performance is one reason for the superior scenic net statistic in Table I.

Table III shows that our global router achieves better runtime, netlength, and via counts than the global router that is part of ISR. Moreover, the difference between global routing netlength and the detailed routing netlength reported in Table I is considerably smaller with BonnRoute. Our global router spends less than 5% of its runtime in rip-up and rerouting. Chip 8 is the only chip on which rip-up and reroute does not eliminate all edge capacity violations after randomized rounding. However, only a single edge of the global routing graph is over-utilized in this solution by one

Table III.
Global routing results of BonnRoute's global router and the ISR global router. In addition to the total runtime of our global router, we report how much of it is spent in Algorithm 2 and in rip-up and reroute ("R&R"). Netlength and via counts include detailed wiring in the input, which for BR-global also comprises the wiring added during prerouting of short nets (cf. Section 2.5)

Chip	Time (mm:ss)			Netlength (m)			Number of vias	
	BR-global	(Alg. 2 / R&R)	ISR-global	Steiner	BR-global	ISR-global	BR-global	ISR-global
1	00:37	(00:11 / 00:01)	02:09	2.786	2.931	2.968	773,231	812,403
2	00:46	(00:18 / 00:01)	01:54	3.254	3.401	3.503	786,839	896,627
3	00:39	(00:12 / 00:01)	02:18	2.802	2.944	2.974	798,370	801,443
4	00:38	(00:09 / 00:01)	01:54	3.076	3.230	3.233	832,925	790,567
5	02:05	(00:53 / 00:05)	05:00	9.089	9.661	9.788	2,128,821	2,431,503
6	04:51	(03:04 / 00:11)	15:15	11.775	12.408	13.806	2,715,991	3,371,812
7	03:05	(01:24 / 00:07)	05:58	11.317	11.921	12.124	2,797,863	3,142,063
8	13:43	(09:34 / 00:27)	14:25	35.635	37.502	38.532	5,700,827	5,710,111
Sum	26:24	(15:45 / 00:54)	48:53	79.734	83.998	86.928	16,534,867	17,956,529

standard-wide wire, which is probably due to an under-estimation of routing capacities in presence of complex blockage structures.

ACKNOWLEDGMENTS

The authors would like to thank our cooperation partners at IBM, in particular Karsten Muuss, Sven Peyer, and Gustavo Tellez. Moreover, we would like to acknowledge the contributions of former members and students of the BonnRoute team, in particular Asmus Hetzel, Christoph Albrecht, André Rohe, Sven Peyer, Jesco Humpola, Lars Bellinghausen, Corinna Gottschalk, Daniel Joachimi, Niko Klewinghaus, Felix Nohn, Thomas Petig, and Rudolf Scheifele. We also thank the anonymous referees for many useful remarks that helped us to improve the presentation.

REFERENCES

ALBRECHT, C. 2001. Global routing by new approximation algorithms for multicommodity flow. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 20*, 622–632.

ALPERT, C., LI, Z., MOFFITT, M., NAM, G., ROY, J., AND TELLEZ, G. 2010. What makes a design difficult to route. In *Proceedings of the International Symposium on Physical Design*. 7–12.

BATTERYWALA, S., SHENOY, N., NICHOLLS, W., AND ZHOU, H. 2002. Track assignment: a desirable intermediate step between global routing and detailed routing. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 59–66.

CARDEN, R., LI, J., AND CHENG, C.-K. 1996. A global router with a theoretical bound on the optimal solution. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 15*, 208–216.

CHANG, C.-C. AND CONG, J. 2001. Pseudopin assignment with crosstalk noise control. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 20*, 5, 598–611.

CHANG, Y.-J., LEE, Y.-T., AND WANG, T.-C. 2008. NTHU-route 2.0: a fast and stable global router. In *Proceedings of the International Conference on Computer-Aided Design*. 338–343.

CHEN, H., CHENG, C.-K., KAHNG, A., MANDOIU, I., WANG, Q., AND YAO, B. 2003. The Y-architecture for on-chip interconnect: Analysis and methodology. In *Proceedings of the IEEE/ACM Design Automation Conference*. 13–19.

CHEN, H.-Y., HSU, C.-H., AND CHANG, Y.-W. 2009. High-performance global routing with fast overflow reduction. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 582–587.

CHEN, T. AND CHANG, Y. 2007. Multilevel full-chip gridless routing with applications to optical-proximity correction. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 26*, 1041–1053.

CHO, M., LU, K., YUAN, K., AND PAN, D. Z. 2009. Boxrouter 2.0: A hybrid and robust global router with layer assignment for routability. *ACM Trans. Des. Autom. Electron. Syst. 14*, 32, 1–21.

CHU, C. AND WONG, Y.-C. 2008. FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 27*, 70–83.

CONG, J., FANG, J., AND KHOO, K. 1999. An implicit connection graph maze routing algorithm for ECO routing. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 163–167.

Cong, J., Fang, J., and Zhang, Y. 2001. Multilevel approach to full-chip gridless routing. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 396–403.

Cross, D., Nequist, E., and Scheffer, L. 2007. A DFM aware, space based router. In *Proceedings of the International Symposium on Physical Design*. 171–172.

Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik 1*, 269–271.

Dion, J. and Monier, L. 1995. Contour: A tile-based gridless router. WRL Res. Rep. 95/3, Western Research Laboratory, Palo Alto, CA.

Garg, N. and Könemann, J. 2007. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput. 37*, 630–652.

Gester, M. 2009. Voronoi-Diagramme von Achtecken in der Maximum-Metrik. Diploma Thesis, University of Bonn.

Goldberg, A. and Harrelson, C. 2005. Computing the shortest path: A* search meets graph theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*. 156–165.

Grigoriadis, M. and Khachiyan, L. 1994. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM J. Optim. 4*, 86–107.

Grigoriadis, M. and Khachiyan, L. 1996. Coordination complexity of parallel price-directive decomposition. *Math. Operations Res. 21*, 321–340.

Hadlock, F. O. 1977. A shortest path algorithm for grid graph. *Networks 7,* 323–334

Hanan, M. 1966. On Steiner's problem with rectilinear distance. *SIAM J. Appl. Math. 14*, 255–265.

Hart, P. E., Nilsson, N. J., and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernetics 4*, 2, 100–107.

Hetzel, A. 1998. A sequential detailed router for huge grid graphs. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe*. 332–339.

Ho, T.-Y., Chang, C.-F., Cheang, Y.-W., and Chen, S.-J. 2005. Multilevel full-chip routing for the Xbased architecture. In *Proceedings of the Design Automation Conference*. 597–602.

Hoel, J. H. 1976. Some variations of Lee's algorithm. *IEEE Trans. Comput. 25*, 19–24.

Hu, J. and Sapatnekar, S. 2001. A survey on multi-net global routing for integrated circuits. *Integration VLSI J. 31*, 1–49.

Humpola, J. 2009. Schneller Algorithmus für kürzeste Wege in irregulären Gittergraphen. Diploma Thesis, University of Bonn.

ICCAD. 2009 global routing benchmarks. http://www.eecs.umich.edu/~mmoffitt/iccad2009.html.

ISPD. global routing contest and benchmark suite (2007). http://www.sigda.org/ispd2007/contest.html.

ISPD. global routing contest and benchmark suite (2008). http://www.sigda.org/ispd2008/contests/ispd08rc.html.

Jansen, K. and Zhang, H. 2008. Approximation algorithms for general packing problems and their application to the multicast congestion problem. *Math. Prog. A*, 183–206.

Johann, M. and Reis, R. 2000. Net by net routing with a new path search algorithm. In *Proceedings of the 13th Symposium on Integrated Circuits and Systems Design*. 144–149.

Korte, B., Prömel, H., and Steger, A. 1990. Steiner trees in VLSI-layout. In *Paths, Flows, and VLSI-Layout*, B. Korte, L. Lovász, H. Prömel, and A. Schrijver, Eds. Springern.

Korte, B., Rautenbach, D., and Vygen, J. 2007. BonnTools: Mathematical innovation for layout and timing closure of systems on a chip. *Proc. IEEE 95*, 555–572.

Lee, C. Y. 1961. An algorithm for path connection and its applications. *IRE Trans. Electron. Comput. 10*, 346–365.

Lee, D. T., Yang, C. D., and Wong, C. K. 1996. Rectilinear paths among rectilinear obstacles. *Discrete Appl. Math. 70*, 185–215.

Lee, T.-H., Chang, Y.-J., and Wang, T.-C. 2011. An enhanced global router with consideration of general layer directives. In *Proceedings of the International Symposium on Physical Design*. 53–60.

Lee, T.-H. and Wang, T.-C 2008. Congestion-constrained layer assignment for via minimization in global routing. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 27*, 1643–1656.

Li, Y.-L., Chang, Y.-N., and Cheng, W.-N. 2011. A gridless routing system with nonslicing floorplanning-based crosstalk reduction on gridless track assignment. *ACM Trans. Des. Autom. Electron. Syst. 16*, 19:1–19:25.

Massberg, J. and Nieberg, T. 2012. Rectilinear paths with minimum segment lengths. *Discrete Appl. Math.*, to appear.

Mitchell, J. 1989. An optimal algorithm for shortest rectilinear paths among obstacles. In *Proceedings of the 1st Canadian Conference on Computational Geometry*.

MOFFITT, M. 2008. Maizerouter: Engineering an effective global router. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 27*, 2017–2026.

MOFFITT, M., ROY, J. A., AND MARKOV, I. L. 2008. The coming of age of (academic) global routing. In *Proceedings of the International Symposium on Physical Design*. 148–155.

MOFFITT, M. D. 2009. Global routing revisited. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 805–808.

MOFFITT, M. D. AND SZE, C. N. 2011. Wire synthesizable global routing for timing closure. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 545–550.

MÜLLER, D. 2006. Optimizing yield in global routing. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 480–486.

MÜLLER, D. 2009. Fast resource sharing in VLSI routing. Ph.D. thesis, University of Bonn.

MÜLLER, D., RADKE, K., AND VYGEN, J. 2011. Faster min-max resource sharing in theory and practice. *Math. Prog. Comp. 3*, 1–35.

NIEBERG, T. 2011. Gridless pin access in detailed routing. In *Proceedings of the IEEE/ACM Design Automation Conference*. 170–175.

OZDAL, M. AND WONG, M. 2009. Archer: A history-based global routing algorithm. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 28*, 528–540.

PAPADOPOULOU, E. AND LEE, D. 2001. The L1 Voronoi diagram of segments and VLSI applications. *Int. J. Computat. Geometry Appl. 11*, 503–528.

PEYER, S., RAUTENBACH, D., AND VYGEN, J. 2009. A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing. *J. Discrete Algorithms 7*, 377–390.

PLOTKIN, S., SHMOYS, D., AND TARDOS, E. 1995. Fast approximation algorithms for fractional packing and covering problems. *Math. Operations Res. 2*, 257–301.

PREPARATA, F. P. AND SHAMOS, M. I. 1985. *Computational Geometry*. Springer.

RAGHAVAN, P. AND THOMPSON, C. 1987. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica 7*, 365–374.

ROY, J. AND MARKOV, I. 2008. High-performance routing at the nanometer scale. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 27*, 1066–1077.

RUBIN, F. 1974. The Lee path connection algorithm. *IEEE Trans. Comput. 23*, 907–914.

SCHULTE, C. 2012. Design rules in VLSI routing. Ph.D. thesis, University of Bonn.

SHAHROKHI, F. AND MATULA, D. 1990. The maximum concurrent flow problem. J. ACM 37, 318–334.

SHRAGOWITZ, E. AND KEEL, S. 1987. A global router based on a multicommodity flow model. *Integration VLSI J. 5,* 3–16.

TEIG, S. 2002. The X architecture: Not your father's diagonal wiring. In *Proceedings of the International Workshop on System-Level Interconnect Prediction*. 33–37.

TING, B. AND TIEN, B. N. 1983. Routing techniques for gate array. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 2*, 301–312.

VYGEN, J. 2004. Near-optimum global routing with coupling, delay bounds, and power consumption. In *Proceedings of the 10th Conference on Integer Programming and Combinatorial Optimization*. Springer, 308–324.

WEI, Y., SZE, C., ET AL. 2012. GLARE: Global and local wiring aware routability evaluation. In *Proceedings of the IEEE/ACM Design Automation Conference*. 768–773.

WU, T.-H., DAVOODI, A., AND LINDEROTH, J. 2011a. GRIP: Global routing via integer programming. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 30*, 72–84.

WU, T.-H., DAVOODI, A., AND LINDEROTH, J. 2011b. Power-driven global routing for multi-supply voltage domains. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe*. 1–6.

XU, Y., ZHANG, Y., AND CHU, C. 2009. Fastroute 4.0: global router with efficient via minimization. In *Proceedings of the Asia and South Pacific Design Automation Conference*. 576–581.

YOUNG, N. 2001. Sequential and parallel algorithms for mixed packing and covering. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*. 538–546.

Received March 2012; revised August 2012; accepted November 2012