Detailed Routing Algorithms for Advanced Technology Nodes

Markus Ahrens, Michael Gester, Niko Klewinghaus, Dirk Müller, Sven Peyer, Christian Schulte, and Gustavo Téllez

***Abstract*—We present algorithms for routing in advanced technology nodes, used by BonnRoute (BR) to obtain efficient and almost design rule clean wire packings and pin access solutions. Designs with dense standard cell libraries in presence of complex industrial design rules, with a special focus on multiple patterning lithography are considered. The key components of this approach are a multilabel interval-based shortest path algorithm for long on-track connections, and a dynamic program for computing packings of pin access paths and short connections between closely spaced pins. The multilabel path search implementation is very general and is driven with different labeling rules, allowing to trade-off runtime against accuracy in terms of obeyed design rules. We combine BR with an industrial router for cleaning up the remaining design rule violations, and demonstrate superior results over that industrial router in our experiments in terms of wire length, number of vias, design rule violations, and runtime.**

***Index Terms*—Algorithms, detailed routing, multiple patterning, very large scale integration.**

I. Introduction

BONNROUTE is the routing solution developed at the University of Bonn in cooperation with IBM. It is part of the BonnTools [1] and has been used by IBM and its customers successfully for the design of more than thousand highly complex chips, for more than two decades.

In this paper, we present those components of BonnRoute (BR) which have been renewed and improved to enable high-quality routing for advanced technology nodes, with special emphasis on multiple patterning. For a more comprehensive overview of BR, see [2].

Multiple patterning is a class of manufacturing technologies for increasing feature density on a chip by assigning objects on a single layer to different manufacturing steps. The first detailed routing algorithms that have been proposed for multiple patterning technologies [3]–[6] have been targeted at the *litho-etch-litho-etch* (LELE) double patterning technology, where objects on each layer are assigned to two different masks that are also referred to as *colors*. These approaches have in common that they assume the availability of stitching and jogs, which is not always realistic. Moreover, they perform coloring greedily during routing, without consideration of convergence issues, and do not consider multiwidth routing and pin access. The prime goal of these approaches is to reduce stitching and coloring conflict counts, which they achieve by various heuristics. Also, triple patterning has been studied as a means to improve on these metrics [7], [8]. Self-aligned double patterning is a multiple patterning technology that does not allow stitching. For this technology, layer assignment has been proposed to resolve conflicts [9], but it is not clear if this is always possible under realistic via spacing constraints, which exceed the track pitch considerably if via layers are done with single patterning. Our approach does not require stitching and hence avoids the yield and routability penalties associated with it. In Section III, we explain how we manage colors and use *track patterns* to achieve convergence in dense routing regions without stitching.

Pin access is becoming more challenging in advanced process nodes, such that simultaneous optimization of pin access and cell layout has been proposed [10]. If standard cells are given, pin escape routing is critical for convergence. In Section IV, we present a new dynamic programming algorithm for simultaneous pin escape and short connection routing. This algorithm is suitable for high pin density routing of small standard cells, and is capable of handling advanced technology design rules and various objectives. Unlike the previous approach applied by BR for pin access [11], [12] and short connection routing [13], our new approach works on whole circuit rows at once. This is a similarity to [14], where an integer multicommodity flow formulation has been proposed for simultaneous pin escape and short connection routing, and a Lagrangian relaxation-based algorithm has been shown to find solutions close to the optimum of the flow formulation in practice. However, it is not clear if realistic requirements such as multiwidth routing, jogs, or via spacing rules of more than one track can be incorporated into the flow formulation, or how well these aspects can be dealt with in their algorithm. All this can be handled by our dynamic programming algorithm, which always finds an optimum solution and allows general objectives, e.g., including spreading of pin access path endpoints in order to improve routability. Our implementation uses only simple routing patterns, although the algorithm does not impose such restrictions. This and the fact that disallowing stitches effectively prohibits most jogs

Manuscript received July 18, 2014; revised October 7, 2014; accepted November 24, 2014. Date of publication December 24, 2014; date of current version March 17, 2015. This paper was recommended by Associate Editor M. Ozdal.

M. Ahrens, M. Gester, N. Klewinghaus, and D. Müller are with the Research Institute for Discrete Mathematics, University of Bonn, 53113 Bonn, Germany (e-mail: {ahrens, gester, klewinghaus, mueller}@or.uni-bonn.de).

S. Peyer and C. Schulte are with IBM Research and Development Böblingen, 71032 Böblingen, Germany. (e-mail: {cschulte, sven.peyer}@de.ibm.com).

G. Téllez is with IBM Watson Research, Yorktown Heights, NY 10598 USA (e-mail: tellez@us.ibm.com).

Digital Object Identifier 10.1109/TCAD.2014.2385755

is a step toward restricting wiring geometries to more regular shapes [15], [16].

Section V describes how BR computes long connections between pins or wires. We keep color dependencies off the standard shortest path algorithm by using an *automatic coloring* described in Section III. The main contribution of this paper is a new, general *multilabel shortest path* algorithm which is used to compute design rule clean paths or nontrivially colored paths in situations where a standard shortest path algorithm does not find good solutions. This algorithm is of special importance for preventing same-net via spacing rule violations that otherwise often result from vias used instead of jogs, when stitching is prohibited and thus jog usage is limited. Our algorithms are not restricted to double patterning, but also work with three or more masks per layer.

In Section VI, we present results demonstrating that BR produces high-quality routings in short runtime on real-world multiple patterning designs. For these results, we combined BR with an external procedure for cleaning up design rule violations.

II. Design Rules

The lithographic manufacturing process of very large scale integration chips has become more difficult with the continuous shrinking of feature sizes way below the still used wavelength of 193 nm. To ensure manufacturability of this process, there is a huge set of *design rules* which has to be satisfied. Before a design can be released to manufacturing it has to pass all design rule checks (*DRC*). We call any violation of a design rule *DRC-error* and shapes not contributing to any violation *DRC-clean*.

Metal shapes belonging to different nets have to satisfy certain minimum distance requirements (*diff-net rules*). In former technologies, this minimum required distance was a nondecreasing function of the width and parallel runlength of the shapes. In multiple patterning technologies, distances also depend on the color of the shapes (in LELE technology these colors represent the assignment to the masks), and automated routing tools are supposed to create colored shapes. Minimum distances for shapes with same colors are larger than for those with different colors here.

Also for shapes belonging to the same net various design rules have to be satisfied *(same-net rules)*, for example minimum edge length and minimum area requirements. These design rules have grown in complexity at every technology node, requiring that automatic routing tools handle them efficiently.

In BR, we use the design rule model developed in [17] which reduces complexity significantly while not being too restrictive. Wires and vias are represented by 1-D abstractions called *stick figures* (see Fig. 1 for an example). *Wire types* map stick figures to their (colored) shape representation and describe their minimum distance requirements in a form which can be checked efficiently by the *checking oracle* of BR. This oracle serves as an interface for queries whether wires at certain positions are legal with respect to diff-net rules. We assume this oracle as given in this paper, see [2] for more details on the data structures behind this interface. In Section III, we describe how the checking oracle is adapted for multiple patterning.

On multiple patterning layers, we call wire types as sketched above *colored wire types* (since they depend on the color of the shape representation) and group such colored wire types, which only differ in their color to one abstract *chameleon wire type*. If a wire with a chameleon wire type is routed, then it adopts a color depending on its position automatically and becomes a colored wire type (see Section III).

III. Routing Space and Color Management

In this section, we describe how routing space is represented and used in BR (see also Section V-A for more details) and how multiple patterning colors for wires are determined. 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 which is still common design practice today. Connections between different layers are called *vias*.

In previous technology nodes (up to 22 nm) BR precomputed for each layer *routing tracks* (lines in preferred direction of the layer) on which the majority of wires was routed on. Here, all different wire types used the same tracks, and only short connections and access paths were routed *off-track* when necessary. We call a wire or via *on-track* if both endpoints of its stickfigure lie on tracks and *off-track* otherwise.

For advanced technology nodes, we generalize this track concept in two ways: first, we define different track sets for different wire types; second, we define a *preferred color* for each track (that should be used by the majority of wires), both to increase packing density by guiding the sequential detailed routing step used in BR (see [2] for details) to pack wires as dense as possible.

In more detail, we use a set of *track patterns* per layer, where each track pattern consists of a set of tracks distributed over the chip area. Each wire type is assigned to exactly one track pattern per layer (colored wire types having the same pattern as its corresponding chameleon wire types), and we say that a wire is *on-track-pattern* if it is on-track with respect to the tracks of the assigned pattern. Each track of a pattern has a *preferred color* that should be used by the majority of wires routed on this track to increase packing density. If the endpoints of a wire stickfigure lie on tracks with the same preferred color (with respect to the assigned track pattern), we call this color the *preferred color* of the wire. Note that jogs may not have a preferred color. For vias, we have separate preferred colors for each via or wiring layer the via intersects. We call these colors the *layerwise preferred colors* of a via.

These track patterns are chosen in a way such that wires with the same wire type can be packed as dense as possible on-track-pattern when following the preferred colors of the tracks. For this, also repeating blocked areas such as power rails are taken into account. See Fig. 1 for an example.

For mixed $1\times$, $3\times$, and $5\times$ wires (these widths actually appear in our real world designs), we obtain a high-packing

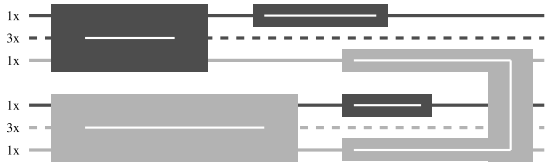


Fig. 1. Example for track patterns and some wires placed and colored according to these track patterns. We have one track pattern for 1× minimum width wires (solid lines) and one for 3× minimum width wires (dashed lines). The two different shades of gray correspond to two multiple patterning colors. The color of a track represents its preferred color. The color of a wire is the preferred color of the track(s) on which the endpoints of the stickfigure (white line) lie(s).

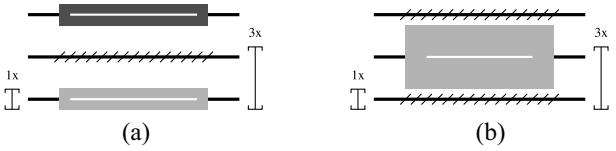


Fig. 2. (a) Two differently colored 1× wires making the crossed out part of the middle track unusable (minimum distance is 1× for diff-color and 3× for same-color). If both wires had the same color, then, the middle track could be used. (b) 3× wire using the same tracks as 1× wires blocks three tracks, while shifting it by 1× distance below or above would only block two tracks for 1× wires (minimum distance is 1× between any wires).

density for on-track-pattern wires colored with the preferred color. Note that in general (for example for 1×, 2×, and 4× wires) a high-packing density in case of mixed width wires is harder to obtain by defining track patterns only. If in such settings the packing density is not satisfying, then an additional track assignment step (which is not yet needed and integrated into BR) could help to obtain better packings (see [18]).

For current technologies, this track pattern concept is essential to obtain dense routing solutions. Without any hint how to place and color wires, the sequential routing step produces unusable gaps [see Fig. 2(a)]. Without separate tracks for different wire types, wider wire types block more tracks than necessary for other wire types [see Fig. 2(b)].

BR uses track patterns in the following way. First, all wires are preferably routed on-track-pattern. Only if some connection cannot be found in this way, off-track-pattern wires are allowed at some penalty cost. Second, the checking oracle is adapted to support queries regarding *chameleon wire types* (see Section II). If the oracle is queried whether an on-track-pattern wire or via with a chameleon wire type is legal, then it actually checks if the wire *colored with the preferred color* (or the via colored with its layerwise preferred colors) is legal with respect to diff-net rules. Here, the color only affects the checking oracle and allows routing algorithms to use the chameleon wire types which adopt the preferred color of the track where a wire is lying. We call this feature of the checking oracle *automatic coloring*.

Jogs which do not have a preferred color (that means, the preferred colors of the tracks where the jog is ending do not coincide) are currently not built by BR, they would either contain a color change (stitch) or would violate a preferred track color on one of the stickfigure endpoints. Both options could be incorporated into the checking oracle if needed.

For the vast majority of routed wires, it is sufficient to use the chameleon wire type, stay on-track-pattern, and use the preferred color, taking color dependency off the routing algorithms. Only if no connection can be found this way, we allow to use other colors (see Section V-D for more details).

This approach allows dense wire packings on the one hand, since (with few exceptions) wires and colors are well aligned by track patterns (see Fig. 1), and faster shortest path computations on the other hand, since the involved algorithms do not need to try different color variants when using the chameleon wire type.

IV. Computing Short Connections and Access Paths

We present a specialized routing approach that can deal with restrictive design rules, high-pin density, and dense wiring, and demonstrate that it significantly improves the routing quality on industrial chip designs in real-world multiple patterning technology nodes. More specifically, we describe how two related problems, the pin access problem and the problem of connecting nearby pins in the same circuit row, can be solved with a single algorithm.

The algorithm exploits the partition of the chip area into circuit rows. This structure was also exploited in coloring algorithms for multiple patterning (see [19], [20]).

In Section IV-A, we give an informal problem definition, and describe our algorithmic solution in Sections IV-B and IV-C. Experimental results on real-world instances are given in Section VI.

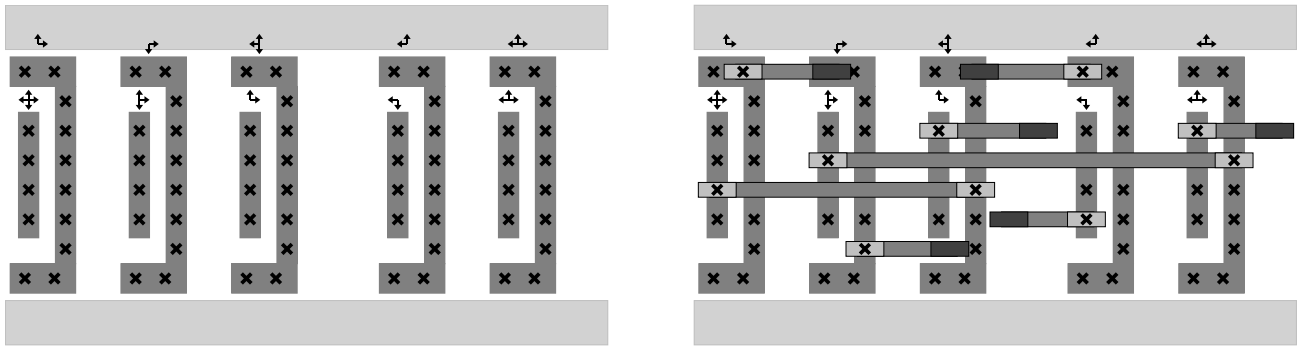
A. Problem Description

Our main goal is to optimize the wiring usage on a single layer called the *access layer*. In our main application, this layer is the second lowest routing layer. The routing layer below the access layer is referred to as the *pin layer*. Without loss of generality, we assume that the preferred routing direction on the access layer is the x-direction. We want to compute short DRC-clean *access paths* for all pins on the access and pin layer such that the endpoints of these paths are well accessible for (long) connections routed later on.

The access paths should optimize certain objectives, for instance there may be a favored direction (e.g., given by global routing corridors and other connected components of the net). Another objective is to increase the flexibility of the later wiring. To achieve this, we prefer solutions in which the (endpoints of) access paths are not too close together (we refer to this objective as *spreading*).

While we concentrate on access paths for pins on the access and pin layer here, there may be also pins on other layers which are not directly accessible for the on-track path search (see Section V) since they do not intersect routing tracks. In such cases, access paths are computed *on the fly* when needed.

The second problem consists of connecting nearby pins (on the access or pin layer) of the same net which are located in the same circuit row.



(a) (b)

Fig. 3. (a) Instance of the GPAP. The two horizontal shapes are power rails that border the circuit row. The other shapes are pins on the pin layer. The arrows above the pins indicate where the other pins of the net are located. The crosses show where it is possible to access the pin with a via. (b) Solution of the instance on the left. Light gray shapes with a cross are vias connecting the pin layer and the access layer, horizontal shapes are wires on the access layer, and black shapes are vias from the access layer to the next higher routing layer.

Solving these problems one-by-one can result in inferior wiring since access paths and short connections use the same routing resources:

1) Access paths can make it impossible to connect nearby pins using the access layer. Then, higher layers must be used and routing resources will be wasted.
2) Short connections may completely block other pins.

For this reason, we define the GENERALIZED PIN ACCESS PROBLEM (GPAP) which consists of computing both short connections and access paths in the same step.

The access layer is partitioned into *circuit rows*, each of which has small width. It suffices to consider the GPAP on circuit rows. Then, a solution for the entire chip can be found by combining the solutions of all circuit rows. This works well because the circuit rows are separated by power rails and thus wires in different circuit rows do not violate minimum distance rules. For an example instance and its solution, see Fig. 3.

Even if wires in neighboring circuit rows can violate minimum distance rules, we can apply this approach. When processing the circuit rows in parallel, we ensure that no two neighboring rows are processed at the same time. Moreover, when checking the legality of a wire, we also consider the access paths and short connections that were placed previously. For such instances, the approach does not necessarily compute optimal solutions, but the solutions should be good in practice.

B. Algorithm

The algorithm (see Fig. 4) works in four main steps. First, we translate the instance into an optimization problem on a graph G (steps 1 and 2). Then in step 3, we compute a path decomposition of bounded width for G, which is used in step 4 to solve the optimization problem optimally.

We next describe the steps of the algorithm in more detail, see also [21] for more specifics. In step 1, a set of legal candidate access paths and candidate short connections (APCandidates and SCCandidates) is computed. In practice, we use mainly simple access paths and short connections that typically consist of a wire on the access layer

Input: The set of pins $\mathcal{P}$ in a circuit row and a partition of $\mathcal{P}$ into a set of nets $\mathcal{N}$.
Output: A pin access for $\mathcal{P}$.

1: $\mathcal{A} = \mathrm{SCCandidates}(\mathcal{N}) \cup \mathrm{APCandidates}(\mathcal{P})$
2: Construct the penalty graph $(G, \mathrm{profit}, \mathrm{penalty})$ for $(\mathcal{P}, \mathcal{A})$.
3: Compute a path decomposition $\mathcal{X}$ for G.
4: Use dynamic programming based on $\mathcal{X}$ to solve the GRAPH PIN ACCESS PROBLEM for $(G, \mathrm{profit}, \mathrm{penalty})$.

Fig. 4. Short connection and pin access algorithm.

that runs in x-direction and vias connecting to the pins (if necessary).

If the access layer and the wiring layer above are multiple patterning layers and the via layer in between is not, then it is favorable to also reserve space for a via on this via layer, since minimum distances on this layer are considerably larger than on the neighboring wiring layers. By this, routing resources on such bottleneck layers are shared early and can be globally optimized (taking all access paths into account), avoiding that in the sequential routing step later on connections compete for the resources left.

Both access paths and short connections must satisfy a set of design rules. We use an oracle that computes how a pin can be accessed legally (see Section V-A for more details). In practice satisfying the most important design rules (e.g., minimum area requirements) is not too complicated since we use very simple paths.

Each access path is assigned a *profit*. The profit can be used to prefer access paths that run in the favored direction (e.g., given by global routing corridors or the relative position of pins in the net).

In step 2, we construct a graph, called *penalty graph*, that completely describes the problem. Conflicting access paths and short connections are connected by an edge with penalty ∞ and pairs that are not illegal, but unfavorable, are assigned a penalty. For an example of penalties, see Fig. 5. Using a graph instead of a hyper-graph is a simplification, since some design

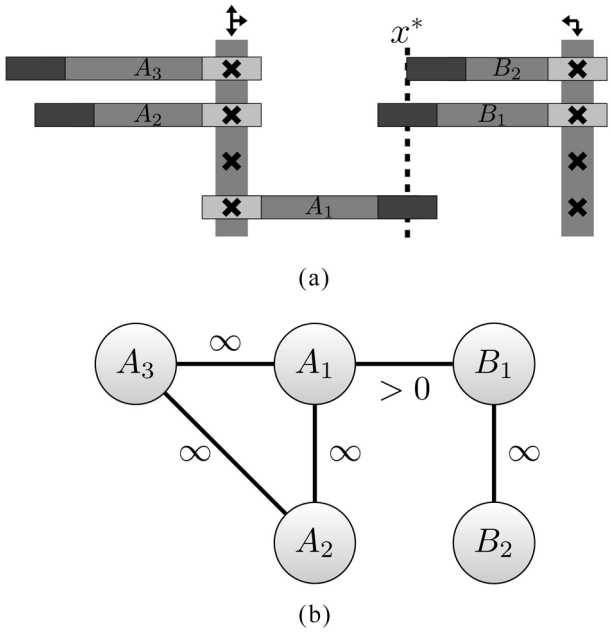


Fig. 5. (a) Access path A_1 runs in the favored direction (indicated by the arrows), while A_2 and A_3 do not. To reflect this, we set profit$(A_1) >$ profit(A_2) and profit$(A_1) >$ profit(A_3). The endpoints of A_1 and B_1 on the routing layer above the access layer are at the same x-coordinate x^*. This restricts further wiring since the endpoint for A_1/B_1 can only be connected to the south/north. To avoid such restrictions, we connect A_1 and B_1 by an edge and set penalty$(\{A_1, B_1\}) > 0$. (b) Edges in the penalty graph for (a) and their penalties.

rules depend on three or more metal shapes. This presentation ignores such rules but the algorithm can easily be extended to handle them.

Definition 1 (Penalty Graph): Let $\mathcal{P}$ be a set of pins and $\mathcal{A}$ be a set of feasible access paths for $\mathcal{P}$. A *penalty graph* for $(\mathcal{P}, \mathcal{A})$ is a triple $(G, \text{profit}, \text{penalty})$, where $G = (\mathcal{A}, E)$ is an undirected graph with node profits profit: $\mathcal{A} \to \mathbb{N}$ and edge penalties penalty: $E \to \mathbb{N} \cup \{\infty\}$, that contains an edge $\{a_1, a_2\} \in E$ with penalty$(\{a_1, a_2\}) = \infty$ if one of the following is true.

1) Both a_1 and a_2 connect to a common pin.
2) Using both a_1 and a_2 violates a design rule.

An independent set in

$$G_\infty = (\mathcal{A}, E_\infty = \{e \in E \mid \text{penalty}(e) = \infty\})$$

is called a *pin access*. The *profit* of a pin access S is

$$\text{profit}(S) = \sum_{a \in S} \text{profit}(a) - \sum_{\{a,b\} \in E(G[S])} \text{penalty}(\{a, b\}).$$

The task is to find a pin access of maximum profit.

GRAPH PIN ACCESS PROBLEM	
Instance:	A penalty graph $(G, \text{profit}, \text{penalty})$ for $(\mathcal{P}, \mathcal{A})$.
Task:	Find a pin access of maximum profit.

The problem generalizes the MAXIMUM WEIGHT INDEPENDENT SET PROBLEM (MWISP): a pin access can include two nodes that are joined by an edge $\{v, w\} \in E \setminus E_\infty$, but this reduces the profit by penalty$(\{v, w\})$. Note that

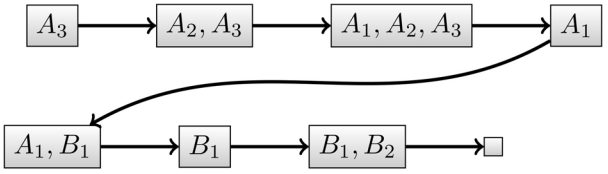


Fig. 6. Path decomposition with width 2 for the graph in Fig. 5(b). Each rectangle depicts a bag.

an instance of the GRAPH PIN ACCESS PROBLEM can be transformed into an equivalent instance of the MWISP by replacing each edge of penalty $p \neq \infty$ by a path of length 3, where both internal nodes have weight p.

In step 3, we compute a path decomposition (defined below) of low width of the penalty graph. Many problems, including the independent set problem, can be solved efficiently on graphs with bounded treewidth, which generalizes pathwidth. For an overview on treewidth and pathwidth, we refer to [22].

Definition 2: Let G be a graph. A *path decomposition* of G is a sequence of subsets of vertices, called *bags*, $(X_1, \ldots, X_r)$, such that the following properties are satisfied.

1) Each edge is contained in some X_i, that is

$$\forall e \in E(G)\ \exists i : e \subseteq X_i.$$

2) For each vertex v, the set of bags that contain v is a nonempty contiguous subsequence of $\mathcal{X}$, that is

$$\exists i < j \in \mathbb{N} : \{X_k \mid v \in X_k\} = \{X_i, \ldots, X_j\}.$$

The *width* of a path decomposition $\mathcal{X}$ is

$$\text{width}(\mathcal{X}) = \max_{X \in \mathcal{X}} |X| - 1.$$

An example is given in Fig. 6.

We sort the elements of $\mathcal{A} = \{a_0, \ldots, a_{s-1}\}$ according to their minimum x-coordinate and define the path decomposition $\mathcal{X} = \{X_0, \ldots, X_{2s-1}\}$ in the following way:

$$\begin{aligned} X_{-1} &= \emptyset \\ X_{2i} &= X_{2i-1} \cup \{a_i\} && i \in \{0, \ldots, s-1\} \\ X_{2i+1} &= \{a_j \in X_{2i} \mid \exists k > i : \{a_j, a_k\} \in E\} && i \in \{0, \ldots, s-1\}. \end{aligned}$$

By this definition, an access path a_i is first inserted into the bag X_{2i}. It is removed one step after its last neighbor (or itself if it is inserted after all of its neighbors) is inserted. Fig. 6 shows the path decomposition computed for the example in Fig. 5. Note that the above procedure can lead to several consecutive identical bags. In Fig. 6, such copies were removed.

In practice, each pin is accessed by a bounded number of short connections and access paths. Moreover, the length of each access paths/short connections is bounded by a constant and the number of tracks in a circuit row is constant. Under these assumptions, the path decomposition $\mathcal{X}$ has bounded width.

Finally in step 4, we use dynamic programming on $\mathcal{X}$ to solve the GRAPH PIN ACCESS PROBLEM optimally. The dynamic program is similar to the one that solves the MWISP

in bounded pathwidth graphs (see [22]). More specifically, we process the bags $X_1, \ldots, X_{2s}$ one-by-one. Throughout the dynamic program after processing the bag X_i, we maintain the following.

1) Every pin access that uses only nodes in X_i, i.e., all independent sets in $G_\infty[X_i]$.
2) For each such pin access S a (super) pin access $S_{\text{ext}} \supseteq S$ that uses only access paths in $\cup_{j\in\{1,\ldots,i\}} X_j$ and is equal when restricted to X_i, i.e., $S \cap X_i = S_{\text{ext}} \cap X_i$. Additionally, we require that S_{ext} is optimal among the pin accesses that have this property.

If we have this data for $i \in \{-1, \ldots, 2s-2\}$, we can compute the same data for $i+1$. After processing the last bag $X_{2s-1} = \emptyset$ an optimal pin access is given by $\emptyset_{\text{ext}}$. If $\mathcal{X}$ has bounded width the runtime of the dynamic program is linear in the number of access paths and short connections. In practice, the average number of combinations that are enumerated in one step is between 200 and 1000 for typical instances.

C. Speeding up the Algorithm

A naive implementation of the algorithm in Fig. 4 has poor runtime in practice. To achieve acceptable runtimes, we use several speedup techniques. We mention a few examples.

For any pin P, the set $\mathcal{A}_P$ of access paths and short connections that access P forms a clique. In a path decomposition an element of $\mathcal{A}_P$ can be removed only after every element has been inserted. For large pins with many access paths, the above approach can lead to path decompositions of high width and to a high number of combinations that need to be enumerated by the dynamic program. To overcome this difficulty, we simply remove large cliques C from the penalty graph. To prevent the algorithm from using two elements of C, we need to enumerate whether an element of C was already used. This is only necessary after the first element of C was inserted and before the last element of C was removed.

Another possibility to speed-up the algorithm is to improve the path decomposition $\mathcal{X} = (X_1, \ldots, X_l)$. Our above procedure for constructing the path decomposition proceeds as follows.

1) First, determine the order in which the elements are inserted into bags.
2) Then, we get a path decomposition by defining that an access path is removed as soon as all its neighbors and itself were inserted.

The extension in 2) optimizes the width among all path decompositions that insert the elements according to the same order. If we are given an order of the removals instead of the insertions, we can use 2) on $(X_l, \ldots, X_1)$ to optimize the width among all path decompositions that remove the elements in the same order.

Since a path decomposition $\mathcal{X}$ contains both an order of the insertions and an order of the deletions (after breaking ties arbitrarily), we can improve $\mathcal{X}$ by alternatingly performing both optimizations. In practice, using both optimizations reduces the runtime of the dynamic program by a factor of approximately four.

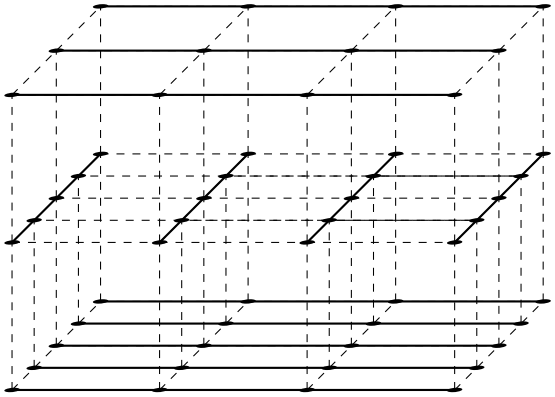


Fig. 7. Example for a track graph. Solid lines are tracks and black dots are vertices of the track graph. Two neighboring vertices connected by a solid or dashed line are connected by an edge in the track graph.

In Section VI, we present experimental results demonstrating the effectiveness of our algorithm on multiple patterning designs.

V. Computing Long Connections

A. Representation of Search Space

We now describe how the search space for the computation of long on-track connections is represented in BR. The *track graph* contains a vertex for each intersection point of tracks on neighboring wiring layers. 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. See Fig. 7 for an example.

We use a directed version $\bar{G}$ of the track graph in the following, where each undirected edge is replaced by two oppositely directed edges. We assume that we are given one source and one target component of a net and a *routing corridor* $A \subseteq V(\bar{G})$ computed in the global routing step of BR (see [2], [23]). Our goal is to connect the two components by on-track wires within $\bar{G}[A]$ (this restriction is relaxed gradually if no connection is found).

The source and target components (consisting of pins or wires) are transformed to *access objects* by an access area oracle (see [17] for details). Each such object contains one vertex of $\bar{G}[A]$ where the source or target component can be accessed (called the *access vertex*) together with one additional vertex of $\bar{G}[A]$ for each possible routing direction (called *direction vertex*). The access objects are build in such a way that when accessing an access vertex with a wire in a certain direction, this wire must run at least to the corresponding direction vertex to be legal with respect to same-net rules.

Now, let $S \subseteq V(\bar{G})$ be the set of access vertices of the source component, $T \subseteq V(\bar{G})$ the set of access vertices of the target component, and w a given chameleon wire type which adopts a color depending on the location of a wire (see Section II).

We assume $A \cap S \neq \emptyset$, $A \cap T \neq \emptyset$ and A connected. The restriction to one wire type is just a simplification, more wire types can also be handled in a straightforward way. Let G be the graph resulting from $\bar{G}[A]$ by the following.

1) Removing all edges whose usage with wire type w is not legal due to the checking oracle (either because the auto-colored version of w would introduce a diff-net violation or the track does not belong to the track pattern assigned to w).
2) Replacing each edge of the form $(s, w) \in E(\bar{G}[A]), s \in S$ by an edge (s, w'), where w' is the direction vertex of access vertex s for the direction of edge (s, w).
3) Replacing each edge of the form $(v, t) \in E(\bar{G}[A]), t \in T$ by an edge (v', t), where v' is the direction vertex of access vertex t for the direction of edge (v, t).

Now, by construction S-T-paths in G do not violate diff-net rules and the two end segments of the path do not produce same-net errors in combination with the pins or wires they access. Note that G is not a 3-D grid graph in general (in contrast to $\bar{G}$), since edges incident to S or T may cross other vertices. However, the algorithms we describe in the following are able to handle such graphs as well. We do not store G explicitly, but rather query the checking oracle (see Sections II and III) and the access area oracle for usable edges as needed.

For a vertex $v \in V(G)$, we denote its x-, y-, and z-coordinates by $x(v)$, $y(v)$, and $z(v)$, respectively. We denote the set of layers on a chip by Z and the set of possible directions for edges in G by $R := \{x_-, x_+, y_-, y_+, z_-, z_+\}$. For an edge $e = (v, w) \in E(G)$, we denote its direction by $r(e) \in R$ and its length by $l(e) := \|v - w\|_1$.

We call a path P in G an r-path if $r(e) = r \in R$ for all $e \in E(P)$ and we call a path a *straight path* if it is an r-path for some $r \in R$. Further, we denote the direction of a straight path P by $r(P)$. We define edge costs

$$c((v,w)) = \begin{cases} \gamma_{\{z(v),z(w)\}} & \text{if } (v,w) \text{ is a via} \\ \beta_{z(v)} \cdot l((v,w)) & \text{if } (v,w) \text{ is a jog} \\ l((v,w)) & \text{otherwise} \end{cases}$$

for two neighboring vertices $v, w \in V(G)$, where $\beta_{z(v)}, \gamma_{\{z(v),z(w)\}} \in \mathbb{Z}_{>0}$ are layer dependent parameters that encode penalty costs for wires running in nonpreferred direction and for vias, respectively. Now, the core problem to be solved for computing on-track connections can be formulated as follows.

PATH SEARCH PROBLEM
Instance: A search instance (G, c, S, T).
Task: A shortest S-T-path in G with respect to cost function c (or the information that no such path exists).

B. Multilabel Shortest Paths

We now present a generalization of the PATH SEARCH PROBLEM taking additional constraints for the resulting path into account. We discuss applications such as shortest paths with minimum segment lengths or shortest colored paths with stitching costs. In Section V-C, we explain how this approach is integrated into the interval-based on-track path search algorithm in BR.

There has been related work on finding shortest paths incorporating same-net rules such as minimum edge lengths, see [12] and [24]. However, our approach is more general and can be incorporated into any existing graph-based shortest path algorithm by only adapting the underlying graph.

The key idea is to model path properties by assigning labels to the vertices of the path and allowing only certain label changes, at some specified cost. The following definition formalizes this idea.

Definition 3: A multilabel system is a triple (L, t, d), where $L := \{l_1, l_2, \ldots, l_k\}$ is a finite set of label types, $t : L \times L \times Z \times R \to \mathbb{N} \cup \{\infty\}$ is a label transition function and $d : L \times L \times Z \times R \to \mathbb{N}_0$ is a label cost function.

A multilabel system is called well-defined if $t(l, l, p, r) \in \{1, \infty\}$, $d(l, l, p, r) = 0$ for all $l \in L, p \in Z, r \in R$ and $t(l_1, l_2, p, r) \neq \infty \Rightarrow t(l_2, l_2, p, r) = 1$ for all $l_1, l_2 \in L$, $p \in Z, r \in R$.

In the following, we only use well-defined multilabel systems, since they are general enough for our purpose and we will utilize their properties for our graph construction. If we want to specify path properties (e.g., minimum segment lengths in certain directions) by multilabel systems, we cannot just inspect the edges of a path in G, since there may be straight paths which are long (and thus legal), but contain short edges only. Because of this, we directly define multilabel sequences and paths based on straight paths instead of edges.

Definition 4: A multilabel sequence in G is a pair (P, ϕ), where P is a sequence $v_1, P_1, v_2, \ldots, v_k, P_k, v_{k+1}$ such that P_i is a straight path in G from v_i to v_{i+1} and $\phi : \{v_1, v_2, \ldots, v_{k+1}\} \to L$ for a multilabel system $\mathcal{L} := (L, t, d)$. If each vertex in G is traversed at most once (P, ϕ) is called *multilabel path*. A multilabel sequence is called *feasible* with respect to $\mathcal{L}$ if for each $i \in \{1, 2, \ldots, k\}$ we have

$$l(P_i) \geq t(\phi(v_i), \phi(v_{i+1}), z(v_i), r(P_i)).$$

The cost of a feasible multilabel sequence (P, ϕ) is given by

$$c_{\mathcal{L}}(P) := \sum_{i=1,2,\ldots,k} (c(P_i) + d(\phi(v_i), \phi(v_{i+1}), z(v_i), r(P_i))).$$

As an example, we define a multilabel system $\mathcal{L}_{\min} := (L, t, d)$ such that feasible multilabel S-T-paths in G are exactly those paths where all inner segments have length at least $l_{\min}$ (note that the end segments of a path are same-net clean by construction of G, see Section V-A). For this let $L := \{\text{pref}, \text{jog}, \text{via}\}$, $d \equiv 0$ and the label transition function t given by the following table.

Note that $\mathcal{L}_{\min}$ is a well-defined multilabel system. Here, the label types pref, jog, and via at some vertex have the meaning that the incoming path segment must be directed in pref, jog, or via direction, respectively. The label transition function ensures that at least distance $l_{\min}$ is covered between label type changes in x- or y-direction. Note that the end segments of the path may be shorter, since one can start and end

			x_-/x_+	y_-/y_+	z_-/z_+
pref	$\rightarrow$	pref	1	∞	∞
jog	$\rightarrow$	pref	$l_{\min}$	∞	∞
via	$\rightarrow$	pref	$l_{\min}$	∞	∞
pref	$\rightarrow$	jog	∞	$l_{\min}$	∞
jog	$\rightarrow$	jog	∞	1	∞
via	$\rightarrow$	jog	∞	$l_{\min}$	∞
pref	$\rightarrow$	via	∞	∞	1
jog	$\rightarrow$	via	∞	∞	1
via	$\rightarrow$	via	∞	∞	1

Fig. 8. Definition of label transition function for layers with preferred direction x. The value of $t(l_1, l_2, r, p)$ is given by the entry in row $l_1 \rightarrow l_2$ and column r. For layers with preferred direction y, the roles of jog and pref are interchanged.

the path with an arbitrary label type. We now consider the following problem.

MULTILABEL PATH SEARCH PROBLEM

Instance: A search instance (G, c, S, T) and a well-defined label system $\mathcal{L}$.

Task: A shortest feasible multilabel S-T-path in G with respect to cost function c and label system $\mathcal{L}$ (or the information that no such path exists).

We cannot hope to solve this problem in polynomial time in general since it is NP-hard.

Theorem 1: The MULTILABEL PATH SEARCH PROBLEM is NP-hard.

Proof: We reduce the HAMILTONIAN S-T-PATH PROBLEM in a 2-D grid graph H (see [25] for a proof of NP-hardness) to the MULTILABEL PATH SEARCH PROBLEM.

So let (H, s, t) be an instance of the HAMILTONIAN S-T-PATH PROBLEM, where $s := (s_x, s_y)$ and $t := (t_x, t_y)$. Let G be the 3-D grid graph containing a copy of H on layer 0, two additional vertices $s' := (s_x, s_y, 1)$ and $t' := (t_x, t_y, 1)$ and two additional edges $e_s := ((s_x, s_y, 1), (s_x, s_y, 0))$ and $e_t := ((t_x, t_y, 0), (t_x, t_y, 1))$.

We now define $n := |H|$ and $\mathcal{L} := (L, t, d)$, where $L := \{l_1, l_2, \ldots, l_n\}$,

$$t(l_i, l_j, z, r) := \begin{cases} 1, & \text{if } r \in \{x_-, x_+, y_-, y_+\} \text{ and } j \leq i+1 \\ 1, & \text{if } r = z_- \text{ and } i = j = 1 \\ 1, & \text{if } r = z_+ \text{ and } i = j = n \\ \infty, & \text{else} \end{cases}$$

and $d \equiv 0$. Note that $\mathcal{L}$ is a well-defined multilabel system. We further set $S = \{s'\}$ and $T = \{t'\}$.

We claim that H contains a Hamiltonian path if and only if there is a feasible multilabel S-T-path in G with respect to $\mathcal{L}$, proving NP-hardness of the MULTILABEL PATH SEARCH PROBLEM. To see this, first note that leaving S is only possible with label type l_1 and entering T is only possible with label type l_n. Therefore, since the label type index increases by at most one at any label transition on layer 0, a feasible multilabel S-T-path must visit each vertex on layer 0 exactly once, yielding a Hamiltonian path in H. On the other hand, a Hamiltonian path P in H can be extended to a feasible multilabel S-T-path in G by just adding the two edges e_s and e_t and increasing the index of the label type by one on each edge on layer 0. ■

Input: A search instance (G, c, S, T) and a well-defined multilabel system $\mathcal{L}$

Output: A shortest feasible multilabel sequence P in (G, c, S, T) with respect to $\mathcal{L}$

1: Compute modified search instance $(G_\mathcal{L}, d_\mathcal{L}, S_\mathcal{L}, T_\mathcal{L})$
2: Compute shortest path P' for instance $(G_\mathcal{L}, d_\mathcal{L}, S_\mathcal{L}, T_\mathcal{L})$
3: Map P' to shortest feasible multilabel sequence P
4: **return** P

Fig. 9. Multilabel sequence algorithm.

In practice, the number of label types is a small constant, but we conjecture that even for two label types the problem remains NP-hard.

The constraint making the problem hard is that a path must not visit any edge or vertex multiple times. Therefore, we relax this constraint and search for a shortest feasible multilabel sequence. This can be done in polynomial time as described in the following. We build a modified search instance $(G_\mathcal{L}, d_\mathcal{L}, S_\mathcal{L}, T_\mathcal{L})$ from (G, c, S, T) and a given well-defined multilabel system $\mathcal{L}$, search a shortest path in $(G_\mathcal{L}, d_\mathcal{L}, S_\mathcal{L}, T_\mathcal{L})$ and then translate this path back to a shortest feasible multilabel sequence P in (G, c, S, T) (see Fig. 9). If P is actually a path, then it is the desired shortest feasible multilabel path in G. We will discuss later how cycles in P are handled and how often they actually appear with usual label systems on practical instances.

We now describe how the modified search instance is build. Given a graph G and a label system $\mathcal{L} := (L, t, d)$, we define the graph $G_\mathcal{L}$ as follows. For each vertex in G, we have one copy per label type in $G_\mathcal{L}$, that means $V(G_\mathcal{L}) := V(G) \times L$. Now, let $v \in V(G_\mathcal{L})$, $l_1, l_2 \in L$ and $r \in R$ be fixed. Let w be the first vertex reachable in G by an r-path P from v whose length is at least $t(l_1, l_2, z(v), r)$. If such a w exists, then we insert an edge $e = ((v, l_1), (w, l_2))$ into $G_\mathcal{L}$. The cost of e is defined as $d_\mathcal{L}(e) := c(P) + d(l_1, l_2, z(v), r)$.

We insert edges in this way for all vertices, label types, and directions. The graph $G_\mathcal{L}$ contains exactly $|L|\,|V(G)|$ vertices and at most $|L|^2\,|V(G)|$ edges. Note that usually $|L|$ is a very small constant (≤ 5 for our applications). We denote the induced subgraph $G[V(G) \times \{l\}]$ as G_l for any label type $l \in L$. Edges between different such subgraphs represent valid label changes.

Building the graph $G_\mathcal{L}$ can be done in time $O(|L|^2 \cdot d_{\max} \cdot |V(G)|)$, where $d_{\max}$ is the maximum number of edges of a straight path corresponding to an edge in $G_\mathcal{L}$ (the path denoted as P in the definition of the edges above). We have

$$d_{\max} \leq \max_{z \in Z} \frac{\max_{l_1, l_2 \in L, r \in R} t(l_1, l_2, z, r)}{\min_{e=(v,w) \in E(G)\,:\, z(v)=z} l(e)} + 1$$

which is a small constant in practice (here, the numerator is typically a minimum segment length or distance coming from a design rule and the denominator is in the order of the wiring pitch). See Fig. 10 for a small example of the graph construction for label system $\mathcal{L}_{\min}$.

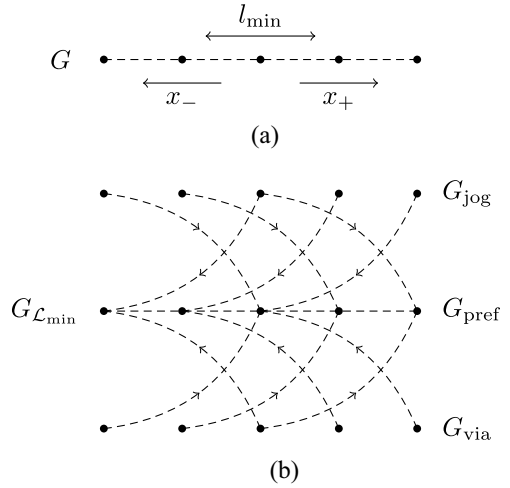


Fig. 10. (a) Simple grid graph G, dashed lines are bidirectional edges in x_- and x_+ direction, and $l_{\min}$ value needed for label system $\mathcal{L}_{\min}$. (b) Graph $G_{\mathcal{L}_{\min}}$ corresponding to graph G in (a) and label system $\mathcal{L}_{\min}$ based on $l_{\min}$ value in (a). Curved dashed lines are directed and represent label changes, straight dashed lines are bidirectional.

By defining $S_{\mathcal{L}} := \{(v, l) \in V(G_{\mathcal{L}}) \mid v \in S\}$ and $T_{\mathcal{L}} := \{(v, l) \in V(G_{\mathcal{L}}) \mid v \in T\}$, we obtain a new shortest path instance $(G_{\mathcal{L}}, d_{\mathcal{L}}, S_{\mathcal{L}}, T_{\mathcal{L}})$.

We will now show that for a well-defined multilabel system $\mathcal{L}$ every feasible multilabel path P in G corresponds to a path P' in $G_{\mathcal{L}}$ such that $c_{\mathcal{L}}(P) = d_{\mathcal{L}}(P')$. Let (P, ϕ) be a feasible multilabel path in G, where P is a sequence $v_1, P_1, v_2, \ldots, v_k, P_k, v_{k+1}$ of straight paths. Let $i \in \{1, 2, \ldots, k\}$ be fixed. Then, by definition $G_{\mathcal{L}}$ contains an edge $((v_i, \phi(v_i)), (w, \phi(v_{i+1})))$ such that w lies on the straight path P_i. We map P_i to this edge plus possible edges between $((w, \phi(v_{i+1})), (v_{i+1}, \phi(v_{i+1})))$ which are part of $G_{\mathcal{L}}$ by definition (here, we use that $\mathcal{L}$ is well-defined, otherwise these edges could be missing). Doing this mapping for each straight path of P yields a path P' in $G_{\mathcal{L}}$ with $c_{\mathcal{L}}(P) = d_{\mathcal{L}}(P')$ (note that keeping a label type does not produce label costs, since $\mathcal{L}$ is well-defined).

On the other hand, each path P' in $G_{\mathcal{L}}$ can be mapped to a feasible multilabel sequence P in G with $c_{\mathcal{L}}(P) = d_{\mathcal{L}}(P')$, just by mapping each edge of P' to a corresponding straight path in G which exists by construction of $G_{\mathcal{L}}$.

For an example of a shortest feasible multilabel path with respect to $\mathcal{L}_{\min}$ in a graph G and its corresponding shortest path in $G_{\mathcal{L}_{\min}}$, see Fig. 11.

For the shortest path search in $G_{\mathcal{L}}$ (step 2 in Fig. 9), we can now use any existing graph-based shortest path algorithm. For the runtime analysis of this algorithm, note that the number of vertices and the number of edges increase at most by the factors $|L|$ and $|L|^2$, respectively, which are small in practice.

As mentioned earlier, we do not store G (as well as $G_{\mathcal{L}}$) explicitly, but rather query the distance rule checking oracle (see Sections II and III) for usable edges as needed. Suppose, we query if edge $((v, l_1), (w, l_2))$ is usable, then it is very useful if the answer may depend on the involved label types l_1 and l_2. In this case, label types can represent different wire types or wire colors, we just have to check if an edge is usable *when used with a certain wire type or color*.

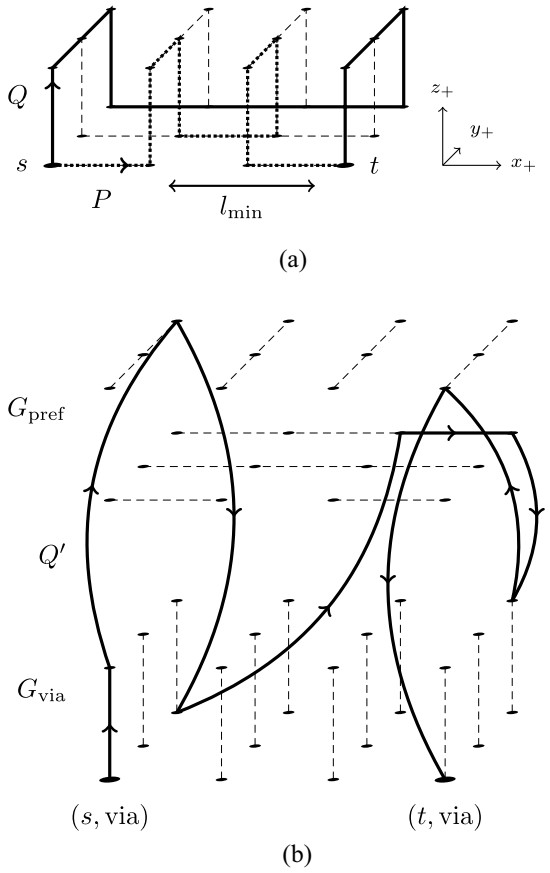


Fig. 11. (a) 3-D grid graph G (edges are dashed and bidirectional) with a shortest s-t-path P (dotted) and a shortest feasible multilabel s-t-path Q (solid) with respect to label system $\mathcal{L}_{\min}$ and with the restriction that s and t are labeled with type via, i.e., $\phi(s) = \phi(t) = \text{via}$. While P contains five segments shorter than $l_{\min}$, potentially causing various same-net errors (minimum area, minimum segment length, minimum same-net via distance), the path Q does not have any segment shorter than $l_{\min}$. (b) Graph $G_{\mathcal{L}_{\min}}$ (edges are dashed and bidirectional) and the shortest (s, via)-(t, via)-path Q' corresponding to path Q from (a) (solid lines). Each curved arc represents a label type change on path Q. Other label transition edges in $G_{\mathcal{L}_{\min}}$ which are not used by Q' are left out for the sake of clarity.

We illustrate this at the following label system. Let $L := \{\text{red}, \text{blue}\}$, $t(l_1, l_2, p, r) = \infty$ if both $l_1 \neq l_2$ and r is a preferred routing direction on layer p, and $t(l_1, l_2, p, r) = 1$ otherwise. Now, we say that edge $((v, l_1), (w, l_2))$ is usable if a wire running from v to w *and colored with* l_1 does not introduce a diff-net violation. With this label system, one can find shortest colored paths avoiding stitches in preferred direction. The label cost function can incorporate stitching costs for other directions. We use this label system in cases where we did not find a path with automatic coloring, see Section V-D.

We also have a similar label system where label types correspond to different wire types, and wire type changes are only allowed in certain directions and at some penalty cost. This is useful to avoid electrically bad configurations in wiring (for example, a thin wire between two wide wires) and minimum edge errors caused by wire type changes.

We now describe how cycles in shortest feasible multilabel sequences are handled and how they can be avoided in advance. Note that the appearance of cycles very much

depends on the involved label systems and cost functions. Cycles with length 3 or more occur very rarely in practical instances, on our testbed in less than 0.4% of all multilabel path searches used in the framework presented in Section V-D. We remove such remaining cycles, yielding a multilabel path which is feasible except for violations caused by this removal. These rare violations do not hurt too much, and many of the corresponding same-net errors can be fixed by a post-processing anyway, yielding paths which are almost clean in terms of those same-net errors respected by the label system.

Cycles of length 2 (2-cycles) appear more often, thus we treat them in a special way depending on the label system. If the label system does not model any distance rules, but for example minimum area or minimum edge rules (such as the one defined in Fig. 8), then cycles of length 2 are often desirable, since they serve as additional metal shapes ensuring that minimum edge lengths or minimum areas are satisfied. But if we have for example a label system forcing certain minimum distances between two vias on the path to compute, then 2-cycles may lead to unfixable situations.

For such label systems, we adapted our Dijkstra-based shortest path algorithm in the following way. When labeling from a node v to an adjacent node w, we only allow this labeling operation if the currently shortest path to v does not arrive at v from the same direction as w. With this, we avoid two-cycles, but we may also forbid legal paths, thus the returned path may not be shortest possible anymore (or we may not even find a path if one exists). However, in practice this approach gives a good trade-off between length and the number of DRC-errors in the computed paths.

C. Multilabel Interval-Based Path Search

In Section V-B, we described a framework to compute shortest feasible multilabel sequences with existing graph-based shortest path algorithms (see Fig. 9). We now show how to integrate this framework into the *interval-based on-track path search* (simply denoted as path search in the following), the main shortest path algorithm in BR [2] for those connected components which are not connected by the dynamic program introduced in Section IV. The path search uses a Dijkstra-based algorithm [26] with two major speed-up features: a *future cost* (similar to the A* heuristic [27]) to reduce the number of labeling steps, and merging vertices to so-called *intervals* to speed up sequences of labeling steps.

A *future cost* is a function $\pi : V(G) \to \mathbb{N}_0$ 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 (with respect to c) from v to T in G for each $v \in V(G)$, and that shortest s-T-paths with respect to c_π (which serves as a feasible potential, see e.g. [28]) are also shortest s-T-paths with respect to c, for each $s \in S$. However, computing them with respect to c_π consumes substantially less label operations (the better the lower bound π, the less label operations), since Dijkstra's algorithm operates goal-oriented in this case (similar to the A* heuristic [27]). Note that each future cost π for an instance (G, c, S, T) is also a valid (but potentially weak) future cost for $(G_\mathcal{L}, d_\mathcal{L}, S_\mathcal{L}, T_\mathcal{L})$.

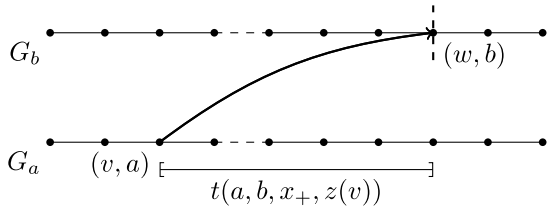


Fig. 12. Example showing that intervals (solid straight lines) must be split for path search in graph $G_\mathcal{L}$. Labeling from (v, a) to (w, b) is allowed, but it is not allowed to any vertex left from (w, b). Thus, the interval must be split at vertex (w, b) to avoid implicit labels in the left part.

Therefore, we can use this future cost to compute a shortest path in step 2 of Fig. 9.

The merging of congenerous and consecutive vertices to *intervals* was proposed in [29] for the special case of equidistant routing tracks which match in all layers with the same preferred direction (see [30], [31] for further generalizations and [2] for a short description). The key idea is that we group vertices that can be labeled at once to intervals and maintain label functions for these intervals. When the path search labels one vertex of an interval, then the label function of the whole interval is also updated at once, saving label operations on the other vertices of the interval later on.

When computing a shortest path for our modified search instance $(G_\mathcal{L}, d_\mathcal{L}, S_\mathcal{L}, T_\mathcal{L})$ as given in Fig. 9, we can use the same merging into intervals, but we may have to split intervals at label operations (such splits are not needed in the standard path search algorithm). We illustrate this in Fig. 12.

In [32], a new runtime bound for the path search applied to multilabel instances is given, including the additional split operations and the buildup of the modified search instance (step 1 in Fig. 9), see the theorem below which we note for the sake of completeness.

Theorem 2 [32]: A shortest feasible multilabel sequence P with respect to a given label system $\mathcal{L} = (L, t, d)$ for the instance (G, c, S, T) and future cost $\pi : V(G) \to \mathbb{N}_0$ can be found in time

$$O(\min\{(\Lambda+1)d_{\max} \cdot k^2 |\mathcal{I}|^2 \cdot \log(k|\mathcal{I}|),\ d_{\max} \cdot k^2 n \cdot \log(kn)\}),$$

where Λ is the cost of P with respect to c_π, $\mathcal{I}$ is the set of intervals representing the vertices of G, $d_{\max}$ is defined as in Section V-B, $k = |L|$ and $n = |V(G)|$.

Note that k and $d_{\max}$ are very small constants in practice (typically both ≤ 5). Also, note that an ordinary shortest path can be found in time

$$O(\min\{(\Lambda+1)|\mathcal{I}| \cdot \log(|\mathcal{I}|),\ n \cdot \log(n)\})$$

for the same instance and notations as in the theorem above (see [29]–[31] for details). We point out that the choice of design rules considered in the definition of the label system impacts its size k and hence offers a trade-off between accuracy and runtime.

In practical experiments, multilabel path searches are also much slower than standard path searches, motivating why one should not always use multilabel path searches, but only when needed, which we explain in more detail in Section V-D.

It is also an interesting question which speedup the merging of vertices to intervals yields in the case of multilabeling. Note that in this case the number of intervals contributes quadratically to the theoretical worst-case bound given in Theorem 2. Indeed, the speedup gained by intervals is much lower for multilabeling compared to the standard path search. For the most complex label system we use, the speedup is only 35%, while for the standard path search it is 65% (we did these experiments on the same testbed as the one described in Section VI). Here, the runs without merging vertices to intervals were also done with the interval-based path search where all intervals were split to singletons. Thus, the actual benefit of intervals may be smaller, since the path search could be implemented faster in practice without interval support.

D. DRC-Aware Path Search Framework

A drawback of many shortest path algorithms used for detailed routing is that most same-net rules cannot be easily incorporated. See Fig. 11(a) for an example where a shortest path may contain many same-net errors. In previous technology nodes (up to 22 nm with single patterning) BR routed long connections by simply calling a standard path search and then trying to fix same-net errors by a post-processing step. This approach gave excellent results in combination with an external DRC-fixing step as demonstrated in [2].

However, in advanced technology nodes, design rule dimensions do not scale well with feature miniaturization anymore (meaning that feature size decreases much more than for example minimum area values) which requires relatively more space for fixing same-net errors afterwards. Here, a post-processing step often fails to get a path DRC-clean which motivates the need for a correct-by-construction path search mode.

In multiple patterning technologies, finding a connection with automatic coloring may fail, motivating the need for a path search mode which is able to choose arbitrary colors on its own.

However, the above desirable modes of the path search (using multilabeling) are much slower than the standard path search combined with post-processing which is still sufficient in many situations. Therefore, we propose the framework sketched in Fig. 13 incorporating same-net errors as well as color choices (in contrast to automatic coloring as described in Section III), which we explain in the following.

As input, we use a set of label systems $\mathcal{M}$ which are ordered by the DRC-rule types they respect. Formally, we define a partial order $\prec$ on $\mathcal{M}$ such that for $\mathcal{L}_1, \mathcal{L}_2 \in \mathcal{M}$ we have $\mathcal{L}_1 \prec \mathcal{L}_2$ if and only if $\mathcal{L}_2$ respects all DRC-rule types that $\mathcal{L}_1$ respects, but at least one additional type. $\mathcal{M}$ always contains the *standard label system* which stands for a call of the standard path search without multilabels, this label system is smaller than any other in $\mathcal{M}$ with respect to $\prec$.

The set $\mathcal{U}$ contains at each point in time the label systems which have not yet been used. Since it does not make sense to start a path search with the same label system twice, we always choose label systems from $\mathcal{U}$.

The path search is called in a loop, starting with as little restrictions as possible (the standard path search). Later, the obtained path is post-processed, and if there are still DRC-errors left (for example due to missing space for local fixes), then we start a new path search using a label system in $\mathcal{U}$ which avoids as many remaining DRC-error types as possible and which is *least restrictive* with this property (that means smallest with respect to $\prec$). We iterate this as long as there are DRC-errors that are fixable by any label system in $\mathcal{U}$ left. The set D collects all DRC-error types which have been left in post-processed paths over all previous iterations. The decisions when to change the path (line 20 of the algorithm) and which label system to choose next (line 24) may be based on more complex criteria than just counting DRC-errors, of course.

Input: A search instance (G, c, S, T) and a partially ordered set of label systems $(\mathcal{M}, \prec)$ as described.
Output: A path P in (G, c, S, T), preferably short and DRC-clean, or $\emptyset$ if no path can be found.

```
 1: U ← M
 2: P ← ∅
 3: D ← ∅
 4: mode ← uncolored
 5: L ← standard label system (the smallest in U w.r.t. ≺)
 6: repeat
 7:     Compute shortest feasible multilabel sequence P'
        in (G, c, S, T) w.r.t. L and mode
 8:     if no such sequence found then
 9:         if mode = uncolored then
10:             mode ← colored
11:             goto line 7
12:         else
13:             return P
14:         end if
15:     end if
16:     U ← U \ {L}
17:     Remove possible cycles in P'
18:     Post-process P'
19:     if P' has fewer DRC-errors than P or P = ∅ then
20:         P ← P'
21:         D ← D ∪ {DRC-error types in P}
22:     end if
23:     if U contains label system avoiding DRC-errors left in P then
24:         L ← a label system in U avoiding most
                DRC-error types of D, smallest w.r.t. ≺
25:     else
26:         return P
27:     end if
28: until
```

Fig. 13. DRC-aware path search framework.

If we do not find any path at some point, we start a new search with mode = colored which means that we now compute a shortest feasible multilabel sequence where we allow color deviations from the preferred track colors. This can be done by a modified label system which has for each original label type of $\mathcal{L}$ one label type per color, similar to the colored label system described in Section V-B. To formalize this label system, we use the following definition.

Definition 5: The cross product of two label systems $\mathcal{L}_1 := (L_1, t_1, d_1)$ and $\mathcal{L}_2 := (L_2, t_2, d_2)$ is defined as $\mathcal{L}_1 \times \mathcal{L}_2 := (L, t, d)$, where $L := L_1 \times L_2$, $t((l_1, l_2), (l_1', l_2'), z, r) := \max(t_1(l_1, l_1', z, r), t_2(l_2, l_2', z, r))$ and $d((l_1, l_2), (l_1', l_2'), z, r) := d_1(l_1, l_1', z, r) + d_2(l_2, l_2', z, r)$ for all $(l_1, l_2), (l_1', l_2') \in L, z \in Z, r \in R$.

Now, if we have mode = colored in line 7, then we compute a shortest feasible multilabel sequence with respect to $\mathcal{L} \times \mathcal{L}_{\text{color}}$, where $\mathcal{L}_{\text{color}}$ is a label system encoding allowed color changes and stitching costs (similar to the one described

TABLE I
CALLS AND RUNTIMES OF STANDARD AND MULTILABEL PATH SEARCH IN DRC-AWARE PATH SEARCH FRAMEWORK ON TESTBED DESCRIBED IN SECTION VI

	Calls		Runtime in (hh:mm)	
Standard path searches	1881452	(95.5%)	12:01	(67.2%)
Multi label path searches	89280	(4.5%)	5:52	(32.8%)
Total	1970732	(100%)	17:53	(100%)

in Section V-B). Using label system $\mathcal{L} \times \mathcal{L}_{\text{color}}$ means that we respect all DRC-rules encoded in $\mathcal{L}$ as well as all constraints and costs for possible color changes (stitches) encoded in $\mathcal{L}_{\text{color}}$. To keep packing density for later connections as high as possible, we include penalty costs for deviations from the preferred colors.

If in this mode still no path can be found, then we take the path from the last round, if there is any. Otherwise, we know that there exists no path, even when using the least restrictive standard path search and allowing color deviations. In such a case, we start the whole framework again, this time allowing rip-up of other nets as described in [2]. We may also start a rip-up loop if we found a path without rip-up which is bad with respect to DRC-errors, netlength or other metrics, depending on the *criticality* of the path. We do not go into more details here.

How to choose the label systems in $\mathcal{M}$ very much depends on the design rules and the post-processing routines used in the framework. Defining all label systems used in BR in detail is out of the scope of this paper. The most important DRC-error types avoided by label systems in BR are minimum area, minimum segment length, and minimum same-net via distance errors. We also incorporate threshold values for *line-end* depending diff-net spacing rules, that means we avoid short edges forcing bigger spacings to other nets. Moreover, we use a label system managing wire type changes within a path, avoiding electrically bad configurations and minimum edge errors caused by a wire type change. This label system is similar to $\mathcal{L}_{\text{color}}$ and can also be used in combination with other DRC-aware label systems by using the cross product.

It is often not the best choice to let $\mathcal{M}$ only contain the standard label system and one label system avoiding *all* above error types, since in many cases only few error types are left in the post-processed path. In such cases, we first try special label systems for the remaining error types instead of the complex label system avoiding all error types, speeding up the multilabel path search call substantially.

In Section VI, we present results (Table III) confirming that the DRC-aware path search framework described in this section, integrated into our combined flow, yields improvements in all measured routing metrics, while even decreasing the total runtime.

We finally demonstrate that the portion of multilabel path searches within the framework is reasonably low. For this, we summed up all calls and runtimes of standard path searches and multilabel path searches for the run containing the DRC-aware path search framework in Table III (see Table I). We see that only 4.5% of all path searches are multilabel path searches, consuming only 32.8% of the total path search runtime.

TABLE II
COMPARISON OF THE PREVIOUS PIN ACCESS APPROACH [11] (LIGHT GRAY ROWS) AND THE GPA APPROACH DESCRIBED IN SECTION IV (DARK GRAY ROWS)

Design	Time(hh:mm)		Errors		Netl.	Vias	Scenics
Nets	BR	Total	BR	Final	(m)	(K)	≥ 1.5
A	0:10	0:58	915	39	0.53	259	33
37360	0:14	1:07	946	23	0.53	252	33
B	0:10	0:31	2255	0	0.62	344	168
42542	0:15	0:49	2111	0	0.61	336	164
C	0:12	0:50	2858	20	0.56	344	90
42637	0:14	0:51	2987	21	0.55	335	80
D	0:19	0:50	2517	6	0.62	400	68
50133	0:17	0:49	2612	10	0.62	387	62
E	0:10	0:40	955	2	0.48	339	74
50793	0:10	0:40	980	0	0.47	326	75
F	0:20	1:26	2514	2	1.05	621	1
82748	0:26	1:24	2311	0	1.05	591	2
G	0:26	1:23	3462	27	1.29	789	68
102995	0:34	1:49	3481	23	1.29	759	72
H	0:21	1:19	2659	2	1.21	750	247
107475	0:25	1:38	2756	2	1.21	724	276
I	0:49	2:51	10070	26	3.30	1555	133
190550	0:51	3:03	10209	19	3.29	1502	126
J	2:11	11:27	45148	38	6.12	4088	333
516197	2:20	12:03	43032	33	6.08	3889	314
Sum	5:08	22:15	73353	162	15.78	9494	1215
1223430	5:46	24:13	71425	131	15.70	9108	1204
Change	+12.3%	+8.8%	-2.6%	-19.1%	-0.5%	-4.1%	-0.9%

VI. EXPERIMENTAL RESULTS

The focus of BR is near optimum packing of wires with respect to wiring length, detours, power, timing, and manufacturing yield. Since the design rule complexity has continuously increased with each new technology, we aim to avoid only the most important classes of DRC-errors in BR. Therefore, in practice an industry standard router (*ISR*) is used as a post-processing step after BR to resolve remaining DRC-errors locally. We denote this combined flow by *BR + ISR*. Both BR and ISR do not use stitching.

The following test runs were done on Intel Xeon E7 8837 2.67 GHz machines using eight threads. The ten instances used are industrial designs from a real-world multiple patterning technology node. The nets of these test instances have to be routed with individual, prescribed wire widths, via sizes, and routing layers based on timing requirements.

All tables in this section compare the runtime, number of remaining errors (DRC-errors and opens), netlength, number of vias, and number of scenics for different runs. We call a net scenic if it has a total wiring length of at least 100 µm and a detour of at least 50% over the length of a minimum Steiner tree (for nets with up to nine terminals) or approximate minimum Steiner tree (for nets with ten or more terminals).

The runtime is given for the BR part of BR + ISR (in the "BR" column) and for the total flow (in the "Total" column). The errors are measured after BR (in the "BR" column) by the internal DRC checker of ISR, and after completion of the total flow (in the "Final" column) by an industry standard sign-off checking tool.

Table II shows a comparison between BR + ISR using the previous pin access approach described in [11] and using our new algorithm for the Generalized Pin Access Problem, denoted GPA. We achieve significant reductions in total

TABLE III
COMPARISON OF BR + ISR WITHOUT (LIGHT GRAY ROWS) AND WITH (DARK GRAY ROWS) DRC-AWARE PATH SEARCH FRAMEWORK, AS DESCRIBED IN SECTION V-D

Design	Time (hh:mm)		Errors		Netl.	Vias	Scenics
Nets	BR	Total	BR	Final	(m)	(K)	≥ 1.5
A	0:10	1:19	4405	13	0.53	256	40
37360	0:14	1:07	946	23	0.53	252	33
B	0:12	0:45	6842	1	0.62	341	256
42542	0:15	0:49	2111	0	0.61	336	164
C	0:20	1:11	8445	24	0.56	341	129
42637	0:14	0:51	2987	21	0.55	335	80
D	0:14	0:48	7938	2	0.62	391	74
50133	0:17	0:49	2612	10	0.62	387	62
E	0:09	0:48	4171	0	0.47	329	81
50793	0:10	0:40	980	0	0.47	326	75
F	0:20	1:23	10728	3	1.05	598	2
82748	0:26	1:24	2311	0	1.05	591	2
G	0:24	1:43	16083	39	1.29	769	81
102995	0:34	1:49	3481	23	1.29	759	72
H	0:20	1:22	10834	4	1.21	733	371
107475	0:25	1:38	2756	2	1.21	724	276
I	0:39	3:00	31299	37	3.30	1519	169
190550	0:51	3:03	10209	19	3.29	1502	126
J	1:53	12:33	105107	43	6.08	3935	313
516197	2:20	12:03	43032	33	6.08	3889	314
Sum	4:41	24:52	205852	166	15.72	9217	1516
1223430	5:46	24:13	71425	131	15.70	9108	1204
Change	+23.1%	-2.6%	-65.3%	-21.1%	-0.1%	-1.2%	-20.6%

TABLE IV
COMPARISON OF ISR (LIGHT GRAY ROWS) AND BR + ISR (DARK GRAY ROWS)

Design	Time (hh:mm)		Errors		Netl.	Vias	Scenics
Nets	BR	Total	BR	Final	(m)	(K)	≥ 1.5
A		2:45		32	0.59	346	809
37360	0:14	1:07	946	23	0.53	252	33
B		2:37		10	0.78	494	3021
42542	0:15	0:49	2111	0	0.61	336	164
C		1:46		27	0.68	479	2364
42637	0:14	0:51	2987	21	0.55	335	80
D		1:55		10	0.69	531	830
50133	0:17	0:49	2612	10	0.62	387	62
E		1:26		4	0.51	466	377
50793	0:10	0:40	980	0	0.47	326	75
F		3:59		26	1.15	821	1175
82748	0:26	1:24	2311	0	1.05	591	2
G		3:05		25	1.42	944	1758
102995	0:34	1:49	3481	23	1.29	759	72
H		4:33		27	1.46	919	3941
107475	0:25	1:38	2756	2	1.21	724	276
I		7:22		20	3.90	1837	5893
190550	0:51	3:03	10209	19	3.29	1502	126
J		19:49		59	6.31	5230	2136
516197	2:20	12:03	43032	33	6.08	3889	314
Sum		49:17		240	17.49	12073	22304
1223430	5:46	24:13	71425	131	15.70	9108	1204
Change		-50.9%		-45.4%	-10.2%	-24.6%	-94.6%

number of final errors, vias, and netlength, at a moderate runtime penalty. Although the number of errors after BR reduces only slightly, minimum area violations are cut down by 62% on average. This is the most important class of same-net design rule violations to avoid, as fixing them locally requires additional space which is scarce on pin access layers. Both approaches were able to connect all except two pins on the entire testbed.

In Table III, we compare BR + ISR without and with the DRC-aware path search framework (see Section V-D). The results confirm that using the framework gives improvements in all measured routing metrics, most significantly the final errors and scenics decrease by more than 20%. Although the overhead for running multilabel path searches leads to a higher BR runtime, the total runtime of BR + ISR decreases.

Table IV compares ISR alone with our proposed BR + ISR flow, including the DRC-aware path search framework and the circuit row-based pin access. The results demonstrate that our combined BR + ISR flow is far superior in every aspect. It runs almost twice as fast while considerably improving on wiring length, routing detours and vias, positively affecting timing and manufacturing yield. Across all chips of our testbed, BR leaves only 26 opens, which confirms that we can route real-world multiple patterning designs without stitching, with an even lower DRC error rate than [2] achieved on single patterning technologies.

ACKNOWLEDGMENT

The authors would like to thank Prof. J. Vygen for a lot of valuable input and discussions. They also would like to thank the entire routing team, both at IBM and the Research Institute for Discrete Mathematics at the University of Bonn, for the contributions, especially the integration of multilabel support into the interval-based on-track path search of BR which was done by F. Nohn. They also thank the anonymous reviewers for many helpful comments.

REFERENCES

[1] B. Korte, D. Rautenbach, and J. Vygen, "BonnTools: Mathematical innovation for layout and timing closure of systems on a chip," *Proc. IEEE*, vol. 95, no. 3, pp. 555–572, Mar. 2007.

[2] M. Gester *et al.*, "BonnRoute: Algorithms and data structures for fast and good VLSI routing," *ACM Trans. Design Autom. Electron. Syst. (TODAES)*, vol. 18, no. 2, p. 32, 2013.

[3] M. Cho, Y. Ban, and D. Z. Pan, "Double patterning technology friendly detailed routing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, 2008, pp. 506–511.

[4] K. Yuan, K. Lu, and D. Z. Pan, "Double patterning lithography friendly detailed routing with redundant via consideration," in *Proc. 46th ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, 2009, pp. 63–66.

[5] Y.-H. Lin and Y.-L. Li, "Double patterning lithography aware gridless detailed routing with innovative conflict graph," in *Proc. 47th ACM Design Autom. Conf.*, Anaheim, CA, USA, 2010, pp. 398–403.

[6] X. Gao and L. Macchiarulo, "Enhancing double-patterning detailed routing with lazy coloring and within-path conflict avoidance," in *Proc. Conf. Design Autom. Test Europe (DATE)*, Dresden, Germany, 2010, pp. 1279–1284. [Online]. Available: http://dl.acm.org/citation.cfm?id=1870926.1871233

[7] Q. Ma, H. Zhang, and M. D. Wong, "Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology," in *Proc. 49th ACM Annu. Design Autom. Conf.*, San Francisco, CA, USA, 2012, pp. 591–596.

[8] Y.-H. Lin, B. Yu, D. Pan, and Y.-L. Li, "TRIAD: A triple patterning lithography aware detailed router," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, San Jose, CA, USA, Nov. 2012, pp. 123–129.

[9] J.-R. Gao and D. Z. Pan, "Flexible self-aligned double patterning aware detailed routing with prescribed layout planning," in *Proc. 2012 ACM Int. Symp. Phys. Design*, Napa, CA, USA, pp. 25–32.

[10] X. Xu, B. Cline, G. Yeric, B. Yu, and D. Z. Pan, "Self-aligned double patterning aware pin access and standard cell layout co-optimization," in *Proc. 2014 ACM Int. Symp. Phys. Design*, Petaluma, CA, USA, pp. 101–108.

[11] T. Nieberg, "Gridless pin access in detailed routing," in *Proc. 48th ACM Design Autom. Conf.*, New York, NY, USA, 2011, pp. 170–175.
[12] J. Maßberg and T. Nieberg, "Rectilinear paths with minimum segment lengths," *Discrete Appl. Math.*, vol. 161, no. 12, pp. 1769–1775, 2013.
[13] T. Petig, "Rectilinear Steiner trees with minimum segment lengths in VLSI routing," Diploma thesis, Res. Inst. Discrete Math., Univ. Bonn, Bonn, Germany, 2012.
[14] M. M. Ozdal, "Detailed-routing algorithms for dense pin clusters in integrated circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 3, pp. 340–349, Mar. 2009.
[15] L. Liebmann *et al.*, "Simplify to survive: Prescriptive layouts ensure profitable scaling to 32nm and beyond," *Proc. Int. Soc. Optics Photonics (SPIE) Adv. Lithography*, vol. 7275, Mar. 2009, Art. ID 72750A.
[16] Y. Zhang and C. Chu, "RegularRoute: An efficient detailed router applying regular routing patterns," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 9, pp. 1655–1668, Sep. 2013.
[17] C. Schulte, "Design rules in VLSI routing," Ph.D. dissertation, Res. Inst. Discrete Math., Univ. Bonn, Bonn, Germany, 2012.
[18] S. Batterywala, N. Shenoy, W. Nicholls, and H. Zhou, "Track assignment: A desirable intermediate step between global routing and detailed routing," in *Proc. 2002 IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, pp. 59–66.
[19] H. Tian, H. Zhang, Q. Ma, Z. Xiao, and M. D. Wong, "A polynomial time triple patterning algorithm for cell based row-structure layout," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2012, pp. 57–64.
[20] M. Ahrens, "Ein Färbungsalgorithmus für Chipverdrahtung," B.Sc. thesis, Res. Inst. Discrete Math., Univ. Bonn, Bonn, Germany, 2012.
[21] M. Ahrens, "Pin access in VLSI-routing," M.Sc. thesis, Res. Inst. Discrete Math., Univ. Bonn, Bonn, Germany, 2014.
[22] H. L. Bodlaender, "A tourist guide through treewidth," *Acta cybern.*, vol. 11, nos. 1–2, p. 1, 1994.
[23] D. Müller, K. Radke, and J. Vygen, "Faster min–max resource sharing in theory and practice," *Math. Program. Comput.*, vol. 3, no. 1, pp. 1–35, 2011.
[24] F.-Y. Chang, R.-S. Tsay, W.-K. Mak, and S.-H. Chen, "MANA: A shortest path maze algorithm under separation and minimum length nanometer rules," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 10, pp. 1557–1568, Oct. 2013.
[25] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter, "Hamilton paths in grid graphs," *SIAM J. Comput.*, vol. 11, no. 4, pp. 676–686, 1982.
[26] E. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, pp. 269–271, 1959.
[27] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.
[28] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms.* Berlin, Germany: Springer, 2012.
[29] A. Hetzel, "A sequential detailed router for huge grid graphs," in *Proc. Design Autom. Test Europe DATE*, Paris, France, 1998, pp. 332–339.
[30] S. Peyer, D. Rautenbach, and J. Vygen, "A generalization of Dijkstra's shortest path algorithm with applications to VLSI routing," *J. Discrete Algorithms*, vol. 7, no. 4, pp. 377–390, 2009.
[31] J. Humpola, "Schneller Algorithmus für kürzeste Wege in irregulären Gittergraphen," Diploma thesis, Res. Inst. Discrete Math., Univ. Bonn, Bonn, Germany, 2009.
[32] F. Nohn, "Detailed Routing im VLSI-Design unter Berücksichtigung von Multiple-Patterning," Diploma thesis, Res. Inst. Discrete Math., Univ. Bonn, Bonn, Germany, 2012.

Markus Ahrens received the B.S. and M.S. degrees in mathematics from the University of Bonn, Bonn, Germany, in 2012 and 2014, respectively, where he is currently pursuing the Ph.D. degree in mathematics from the Research Institute for Discrete Mathematics.

His current research interests include combinatorial optimization and very large scale integration design with a focus on detailed routing.

Michael Gester received the Diploma degree in mathematics from the University of Bonn, Bonn, Germany, in 2009, where he is currently pursuing the Ph.D. degree in mathematics from the Research Institute for Discrete Mathematics.

His current research interests include combinatorial optimization, computational geometry, and very large scale integration design with a focus on detailed routing.

Niko Klewinghaus received the Diploma degree in computer science as well as in mathematics from the University of Bonn, Bonn, Germany, in 2013, where he is currently pursuing the Ph.D. degree in mathematics from the Research Institute for Discrete Mathematics.

His current research interests include combinatorial optimization and very large scale integration design with a focus on detailed routing.

Dirk Müller received the Diploma and Ph.D. degrees in computer science from the University of Bonn, Bonn, Germany, in 2003 and 2009, respectively.

He is currently a Post-Doctoral Researcher in electronic design automation, with a focus on routing with the Research Institute for Discrete Mathematics, University of Bonn.

Mr. Müller has served on the Technical Program Committee of the Design Automation Conference.

Sven Peyer received the Diploma and Ph.D. degrees in mathematics from the University of Bonn, Bonn, Germany, in 2000 and 2007, respectively.

He joined IBM Germany Research and Development, Böblingen, Germany, as an Advisory Development Engineer on algorithmic solutions and methodologies in chip design in 2007. His current research interests include very large scale integration routing.

Mr. Peyer is a Technical Program Committee Member of the Conference for Design, Automation, and Test in Europe.

Christian Schulte received the Diploma and the Ph.D. degrees in computer science from the University of Bonn, Bonn, Germany, in 2006 and 2012, respectively.

From 2006 to 2012, he was a Research Assistant at the Research Institute for Discrete Mathematics, University of Bonn. Since 2013, he has been with IBM Germany Research and Development, Böblingen, Germany, in the field of electronic design automation with focus on routing.

Gustavo Téllez received the B.S. and M.S. degrees in Electrical Engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1986, and the Ph.D. degree in computer science from Northwestern University, Evanston, IL, USA, in 1996.

From 1986 to 2013, he was at IBM Electronic Design Automation, Yorktown, NY, USA, performing pioneering work on the subjects of design for manufacturability, technology migration, routing, and wire synthesis, where he is currently a Senior Technichal Staff Member. He has been the Senior Architect for routing solutions at IBM since 2005, and held the position as the Chief Co-Architect of the Open Access Change Team since 2009.

Mr. Téllez was the recipient of the Hispanic Engineer National Achievement Awards Corporation Information Technology Distinction Award in 2010 and in 2011 was named John von Neumann Visiting Professor at Bonn University, Bonn, Germany.