Algorithms for Maze Routing With Exact Matching Constraints

Muhammet Mustafa Ozdal and Renato Fernandes Hentschke

***Abstract*—Exact route matching is an important constraint for analog and mixed signal designs with nonuniform metal stacks. In this paper, we propose a constrained-path-based maze routing algorithm that can handle exact matching constraints for multiple nets. We also propose a scalable framework that utilizes the proposed maze routing algorithm for realistic problem sizes. Compared to the pattern routing algorithms proposed recently [8], our algorithms allow a more thorough exploration of the solution space by allowing bends to be inserted to avoid congested regions. Furthermore, we propose an ILP-based model to explore the impact of the initial routing topologies on the final solutions. Our experimental study demonstrates that the proposed algorithm leads to significant reductions in congestion costs compared to the previous algorithm.**

***Index Terms*—Dynamic programming, integer linear programming, matching, routing.**

I. Introduction

As THE POPULARITY and complexity of system-on-chips (SOCs) are increasing, more analog components are integrated with digital logic on the same chip. Although robust design automation tools exist for digital parts, the analog design automation is not as mature. Traditionally, manual or semiautomatic tools have been used for analog parts, mainly because of the complexity of analog design. However, with increased time-to-market pressures for modern SOCs, analog and mixed-signal (AMS) parts can become the bottleneck. Hence, improvements in the automation of AMS physical design can significantly improve productivity by reducing the complex, time-consuming, and error-prone manual tasks.

One of the most common classes of constraints in AMS designs is the matching of specific devices and interconnects between them. The correct functionality of analog circuits may depend on exact matching of devices and interconnects, as in the case of pipelined analog/digital converters. Furthermore, common-mode rejection and supply noise rejection characteristics also depend on how well the devices and interconnects are matched. In this paper, we focus on routing of nets with exact matching constraints.

Manuscript received November 5, 2012; revised February 15, 2013 and May 24, 2013; accepted July 22, 2013. Date of current version December 16, 2013. A preliminary version of this paper was presented in IEEE/ACM ICCAD in November 2012 [9]. This paper was recommended by Associate Editor Y. W. Chang.

The authors are with the Intel Corporation, Hillsboro, OR 97124 USA (e-mail: mustafa.ozdal@intel.com; renato.f.hentschke@intel.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2013.2279516

Exactly matching the electrical properties of the interconnects becomes more difficult in the presence of routing layers with different characteristics. For example, it was reported in [1] that the layer pitches for a 65 nm Intel logic technology are 220 nm, 280 nm, 330 nm, 480 nm, 720 nm, and 1080 nm for metal three to metal eight, respectively. It is common in the industry today to have upper layers with reduced resistivities than the lower layers. It was shown that matching the lengths of the routes is not sufficient to match the electrical characteristics of the interconnects. Spice simulations in [8] have shown that even two different L-routes with identical wirelengths and layers (but different segment ordering from driver to receiver) can have significant delay mismatches due to the differences in metal layers. For example, consider a route with a horizontal segment A, and a vertical segment B, where A is assigned to a more resistive layer than B. The interconnect delay will be larger when A is connected to the driver directly, followed by B, compared to when B is connected to the driver, followed by A. Furthermore, there are several factors (such as routing topologies, order of wire segments, the locations and the number of vias) that make it hard to match the electrical characteristics of multiple routes.

For matching the interconnect properties exactly, the following route matching formulation was proposed for a given set of nets [8]: 1) The route of each net n must start from the driver and end at the receiver of n, 2) the routes of all nets must have identical number of wire segments, and 3) the ith segment of each net must have the same wirelength and the same layer assignment. Note that this is a significantly more complex problem than the previous length matching formulations for PCB routing [10]–[12], [14]. When each net has a different horizontal and vertical distance between its driver and receiver, it is especially difficult to satisfy all of these three conditions at the same time, while minimizing the total routing costs. For example, Fig. 1 illustrates two nets with exactly matching routes. Observe that, for both routes, a horizontal segment of length ten is connected to the driver, followed by a vertical segment of length eight, a horizontal segment of length six, and a vertical segment of length two. Although not shown here for clarity, the layer assignment of the segments must be identical as well. The problem gets even more complicated when more than two nets need to be matched exactly.

In a typical mixed-signal SOC, both analog and digital nets need to share the common routing resources. Even if the analog nets are prioritized over the digital nets, there are congestion hotspots and blockages that need to be avoided. In

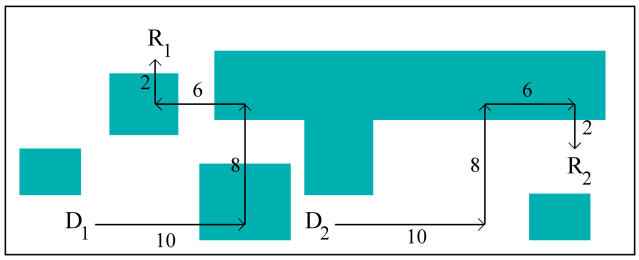

Fig. 1. Exactly matching routes for two nets with (driver, receiver) pairs (D_1, R_1) and (D_2, R_2). The congested regions are shown as shaded rectangles.

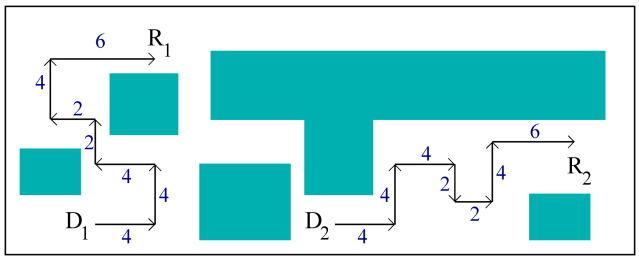

Fig. 2. Different exactly matched routing solution for the problem shown in Fig. 1. All congested regions (shaded rectangles) are avoided in this case.

[8], dynamic programming (DP) algorithms were proposed to solve the exact route matching problem through pattern routing. Although those algorithms can successfully satisfy the exact matching constraints, they have very little flexibility in terms of avoiding the congested regions or blockages, due to the nature of pattern routing. In particular, the algorithms in [8] were allowed to change mainly the ordering of the segments to avoid congestion. However, as shown in Fig. 1, changing the ordering of the segments (e.g., changing the order from $10 \rightarrow 8 \rightarrow 6 \rightarrow 2$ to $8 \rightarrow 10 \rightarrow 2 \rightarrow 6$ for both nets) will not always be sufficient to avoid all the congested regions. On the other hand, a flexible maze routing algorithm can avoid the congested regions. However, the main difficulty is to enforce the exact matching constraints for all nets during maze routing. Direct maze routing extensions [3] lead to impractically high runtime and space complexities, and they cannot handle realistic problem sizes, as will be discussed in Section III-A.

In this paper, we propose a scalable maze routing algorithm that can effectively minimize congestion while obeying the exact route matching constraints. Fig. 2 shows a sample maze routing solution for the problem of Fig. 1. Here, all congested regions are avoided, and the exact route matching constraint is satisfied (i.e., the segment ordering and the individual segment lengths and layers are identical for both nets).

The rest of the paper is organized as follows. We first review the existing route matching models in Section II. Then, in Section III, we propose a maze routing algorithm that leads to exactly matched routes for multiple nets. In particular, we first show why a straightforward maze routing extension would not be effective for this problem in Section III-A. After that, we propose a constrained-path-based exactly-matched maze routing algorithm in Section III-B, and discuss scalability considerations in Section III-C. In Section IV, we propose a graph-based high-level algorithm that utilizes the proposed maze routing algorithm on sequences of wire segments. This algorithm can be used as a scalable framework applicable to practical problems. We also propose an ILP-based method in Section V to generate multiple routing configurations to explore the impact of the initial topology selection. In our experimental study (Section VI), we show that our proposed algorithms can significantly improve congestion compared to the previous work.

II. Preliminaries and Related Work

The length matching problem has been studied extensively in the literature for board-level routing [10]–[12], [14]. However, the IC-level exact matching problem is significantly different because 1) each metal layer has a horizontal or vertical routing orientation unlike mostly-planar routing in board-level routing, 2) different metal layers can have different electrical properties, and 3) the matching requirement for analog nets is more stringent because the correct functionality of the circuit depends on matching. In a relatively recent work, Lin *et al.* [5] studied the transistor-level analog device and interconnect matching problems. However, that work does not focus on how to compute matching routes between different devices, which can be in different locations of the design. In general, the earlier analog routing works mainly focused on device level routing [2], or noise and yield effects [4], [6].

In a more recent work, the exact route matching problem for analog and mixed signal designs was formulated as follows [8]: Given a set of nets $\mathcal{N}$, compute a route R_n for each net $n \in \mathcal{N}$, where R_n consists of k_n segments as $R_n = [s_1^n, ..., s_{k_n}^n]$. For exact route matching constraint to hold, the following three conditions must be satisfied.

1) The segment counts are identical for all nets, i.e., $k_1 = k_2 = ... = k_N$, where N is the number of nets.
2) The lengths of the respective routing segments are equal for all nets, i.e., $s_j^1 = s_j^2 = ... = s_j^N$, for each segment j, $1 \leq j \leq k_n$.
3) The layer assignment of the respective routing segments are identical for all nets, i.e., $layer(s_j^1) = layer(s_j^2) = ... = layer(s_j^N)$, for each segment j, $1 \leq j \leq k_n$.

The exact route matching problem was formulated in [8] as a system of equations for the horizontal and vertical segments as: $C_H \times L_H = D_H$ (for horizontal segments), and $C_V \times L_V = D_V$ (for vertical segments). Here, D_H is the distance vector of size N, where $D_H[n]$ $(1 \leq n \leq N)$ is the horizontal distance between the driver and receiver terminals of net n, and N is the number of nets in $\mathcal{N}$. Then, L_H is the solution vector of size k_h, where $L_H[j]$ is the length of the jth horizontal segment, and k_h is the number of horizontal segments in the matched routing solution. While each net must have identical lengths and layers for each segment, the segment directions for different nets can be different. The matrix C_H with size $N \times k_h$ is called the configuration matrix, which simply determines the direction of each horizontal segment for each net. In particular, if the direction of the jth segment in the routing solution for net n is positive, then $C_H[n, j] = 1$; otherwise, $C_H[n, j] = -1$. Similar definitions also apply for the vertical segments. Fig. 3 shows the corresponding equations for the horizontal and vertical segments of Fig. 1.

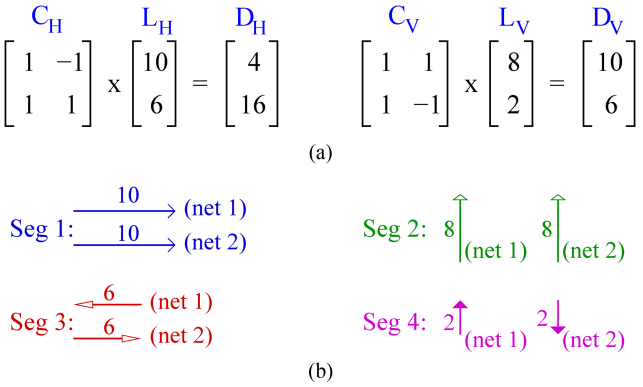

Fig. 3. (a) Equations for the example in Fig. 1. (b) Corresponding segment lengths and directions.

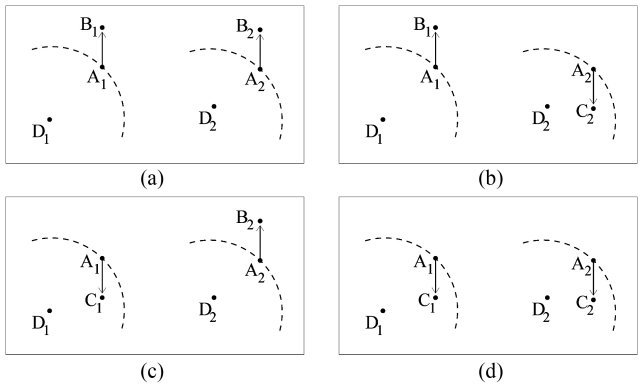

Fig. 4. Simultaneous wavefront expansion for two nets. D_1 and D_2 correspond to the drivers of nets one and two, respectively. Adding a vertical segment to the current partial solution state (A_1, A_2) leads to four new solution states. (a) State (B_1, B_2). (b) State (B_1, C_2) (c) State (C_1, B_2). (d) State (C_1, C_2).

It was also shown in [8] that there is a class of configuration matrices C_m that leads to a feasible matching solution for any set of horizontal or vertical distance vectors D_H or D_V. Authors use these configuration matrices to compute the segment lengths L_H and L_V corresponding to any given distance vectors D_H and D_V. After the directions and lengths of the routing segments were determined, dynamic programming algorithms were proposed to reorder the segments to minimize the total routing costs.

Although the previous algorithms can successfully satisfy the exact matching constraints, the solution space explored is very limited because of the nature of pattern routing. For example, if there are congested regions or blockages in the design (as is commonly the case in typical SOC designs), as shown in Fig. 1, reordering the segments may not be sufficient to avoid all the congested regions. The previous work [8] attempted to address this issue by splitting some of the segments during pattern routing. However, the runtime and space complexity of the proposed algorithm was exponential in the number of additional bends introduced. In Section VI, we show in our experimental study that it is not possible to run [8] with extra bend counts more than one within reasonable runtime limits, due to the exponential runtime.

In another work, Gao *et al.* [3] proposed a direct maze routing extension to handle matching constraints. However, we show in Section III-A that the complexity of that algorithm is $O(|\mathcal{G}|^{|\mathcal{N}|})$, where $|\mathcal{G}|$ is the routing grid size, and $|\mathcal{N}|$ is the number of nets to be matched. Obviously, this runtime and space complexity is too high to be practical for realistic problem sizes. Gao *et al.* [3] proposed to reduce the routing grid size using an idea similar to Hanan grid model. However, for an arbitrary congestion map and large number of blockages, even the reduced grid sizes are expected to be too large for such an expensive algorithm.

In this paper, we propose a maze routing framework for nets with exact matching constraints. Fig. 2 shows the maze routing solution corresponding to the problem in Fig. 1, which avoids all congested regions while still matching the routes of all nets exactly. Our basic algorithm takes a solution to the equations $C_H \times L_H = D_H$ and $C_V \times L_V = D_V$ as input, and produces maze routing solutions corresponding to the segment types defined in L_H and L_V. The main advantage of our proposed algorithm compared to the previous algorithms is that it can explore the solution space much more efficiently and extensively within reasonable runtime to find solutions with smaller congestion.[1] Specifically, while the algorithms of [8] have exponential runtime and space complexity in the number of extra bends, our proposed maze routing framework can handle arbitrary number of extra bends in the solutions.

III. Exactly Matched Maze Routing

In this section, we first discuss why a straightforward maze routing extension is not suitable for this problem (Section III-A). Then, we propose a new maze routing algorithm based on constrained-path optimization (Section III-B), and discuss its scalability implications (Section III-C).

A. Straightforward Maze Routing Extension

The traditional maze routing algorithm operates on a single net at a time. However, in our problem, multiple nets need to be routed in exactly the same way. One can extend the traditional maze routing algorithm in a straightforward way by maintaining a separate wavefront for each net, and by expanding from each wavefront simultaneously to make sure that each partial path has exactly matching routes. This was indeed the approach utilized by [3].

Fig. 4 illustrates an example for two nets. Here, consider a partial solution state (A_1, A_2) on the current wavefront of each net. Four possible ways of expanding from this partial state in the vertical orientation are illustrated in Fig. 4(a)–(d). Observe that the exact route matching constraint will be satisfied as long as the new segment added has the same layer for all nets, regardless of whether the segment is in the positive or negative direction. Such wavefront expansions can start from (D_1, D_2), and continue until the solution state (R_1, R_2) is encountered, where (D_1, D_2), (R_1, R_2) correspond to the driver and receiver terminals of nets one and two, respectively.

The main problem with this approach is the impractically high runtime and space complexity. As in the example of Fig. 4, simultaneous wavefront expansion requires keeping track of a

[1]For the purpose of generality, we use congestion minimization objective in this paper. However, hard blockages can also be handled in a straightforward way by setting their congestion costs to infinity.

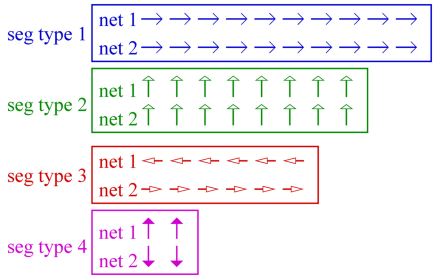

Fig. 5. Set of unit segments $\mathcal{U}_S$ corresponding to the example in Fig. 3 (b). Each unit segment has a type, which defines the direction for each net. There are ten, eight, six, and two unit segments corresponding to types one, two, three, and four, respectively.

different partial solution state corresponding to each combination of grid locations for all nets (e.g., (B_1, C_1), (B_1, C_2), (B_2, C_1), (B_2, C_2) in this example). It is possible to show that the number of partial solution states in such an algorithm will be $O(|\mathcal{G}|^{|\mathcal{N}|})$, where $|\mathcal{G}|$ is the grid size, and $|\mathcal{N}|$ is the number of nets. Note that a typical value for $|\mathcal{G}|$ can be more than a million even for a global routing grid. Obviously, such high runtime and space complexity is not practical even when the number of nets to be matched is small (e.g., between two and five). In [3], authors try to reduce the grid sizes by using a model similar to Hanan grid, and they state that their approach is not scalable for matching beyond three nets. However, in practice, reducing the grid sizes significantly is not possible for real designs because of arbitrary congestion maps and large number of blockages. Furthermore, exact route matching is not guaranteed for a nonuniform grid. In summary, a straightforward maze routing extension is impractical to be applied on realistic problems. In the next subsection, we propose a different maze routing algorithm that allows controlling the runtime and space complexity within practical limits.

B. Proposed Maze Routing Algorithm

We first define a set of segment types that can be utilized in an exactly matched routing solution as follows.

Definition 1: A segment type t for a set of nets $\mathcal{N}$ is defined as a specific routing direction for each net in $\mathcal{N}$. By definition, a segment type can be either a horizontal type (i.e., contains only horizontal directions), or a vertical type (i.e., contains only vertical types).

For example, for three nets, some of the horizontal segment types can be defined as: $(\rightarrow;\rightarrow;\rightarrow)$, $(\rightarrow;\rightarrow;\leftarrow)$, $(\rightarrow;\leftarrow;\rightarrow)$, $(\rightarrow;\leftarrow;\leftarrow)$, etc., corresponding to nets $(n1, n2, n3)$, respectively.

Definition 2: An exact route matching solution for a set of nets $\mathcal{N}$ can be defined as a sequence of segments $s_1 ... s_k$, where each s_i $(1 \leq i \leq k)$ has a specific: 1) segment type, 2) segment length, and 3) layer assignment.

Consider the example in Fig. 1. Here, the matched routing solution consists of a sequence of four segments with types $(\rightarrow;\rightarrow)$, $(\uparrow;\uparrow)$, $(\leftarrow;\rightarrow)$, $(\uparrow,\downarrow)$, and lengths ten, eight, six, and two, respectively. Similarly, the routing solution in Fig. 2 consists of a sequence of seven segments with different types and lengths. The main objective of our proposed algorithm is to compute such a sequence of segments to minimize congestion and via counts.

Definition 3: Let $\mathcal{U}_S$ denote a set of unit segments, where each unit segment in $\mathcal{U}_S$ has a specific type and length. Let u_t denote the number of unit segments in $\mathcal{U}_S$ with type t. The set $\mathcal{U}_S$ is defined to be feasible for $\mathcal{N}$ iff for each net $n \in \mathcal{N}$, a path can be constructed from driver D_n to receiver R_n by using all unit segments in $\mathcal{U}_S$ corresponding to net n.

Fig. 5 shows a set of unit segments $\mathcal{U}_S$, in which there are ten, eight, six, and two unit segments of types one, two, three, and four, respectively. Observe that each segment type has a specific direction for each of the two nets. Fig. 6 illustrates an example routing solution utilizing all the segments in $\mathcal{U}_S$. Since $\mathcal{U}_S$ leads to a complete path between the driver and receiver terminal of each net, $\mathcal{U}_S$ is defined to be feasible for these two nets.

Even for a fixed $\mathcal{U}_S$, the size of the solution space can be very large. For the example of Fig. 5, one can enumerate $\frac{26!}{10!\,8!\,6!\,2!}$ different solutions. The algorithm proposed in this paper aims to compute the best solution in this large solution space. In contrast, the basic pattern routing algorithms in [8] only explored different segment orderings where a horizontal segment follows a vertical segment and vice versa. For the same example, such pattern routing algorithms would explore only eight different solutions.[2]

Given a set of nets, the first step of our approach is to compute a feasible set of unit segments $\mathcal{U}_S$. For this purpose, we can solve the equations $C_H \times L_H = D_H$ and $C_V \times L_V = D_V$ from [8], as explained in Section II. For the example of Fig. 1, the corresponding equations for horizontal and vertical segments are shown in Fig. 3(a). The segment lengths (from L_H and L_V) and segment directions (from C_H and C_V) are illustrated in part (b) of the same figure. Observe that a segment type t is defined in $\mathcal{U}_S$ corresponding to each segment in Fig. 3(b), and u_t (the number of unit segments of type t in $\mathcal{U}_S$) is determined based on the corresponding segment length. In this section, our main focus is on how to explore the solution space defined by a given $\mathcal{U}_S$. Later, in Section V, we will propose an ILP-based approach to compute multiple feasible $\mathcal{U}_S$ sets.

Definition 4: The exactly matched maze routing problem for $\mathcal{N}$ and a feasible set of unit segments $\mathcal{U}_S$ is to compute a path P_n for each net $n \in \mathcal{N}$ such that: 1) P_n consists of exactly all the unit segments in $\mathcal{U}_S$ corresponding to n, 2) the segment type and layer of $P_n[i]$, $0 \leq i \leq |P_n|$, are identical for all nets, and 3) the total routing cost is minimized.

Here, the first condition is to make sure that the path from each driver D_n to receiver R_n is complete. The second condition is to enforce the exact route matching constraints on all nets. The third condition is the optimization objective, and can be defined in different ways. In this paper, we define the routing cost as

$$cost = \alpha.congestion + via\_count \tag{1}$$

where α is a weighting term to trade off between congestion and via counts.

[2]The previous work [8] attempted to address this issue by introducing extra bends in the routing patterns. However, we will show in our experimental study that it is impractical to have more than one bend in those algorithms due to unacceptable runtime overheads.

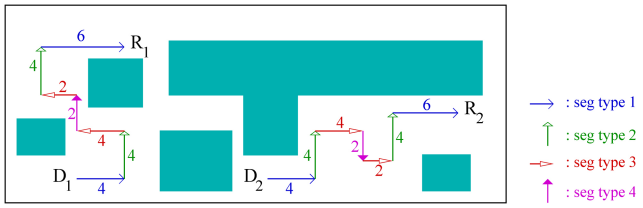

Fig. 6. Exactly matched routing solution for two nets utilizing all the unit segments in $\mathcal{U}_S$ of Fig. 5.

The exact route matching problem as formulated in Definition 4 is a constrained path optimization problem, and can be solved using dynamic programming. For this, let us define a partial solution state as follows.

Definition 5: Assume that there are m segment types in $\mathcal{U}_S$, and the number of unit segments of type t is denoted as u_t. A partial solution state is defined as $\{c_1, c_2, ..., c_m, last\_layer\}$, where each c_t, $1 \leq t \leq m$ corresponds to the number of unit segments of type t in the partial solution, and $last\_layer$ is the layer of the last segment in the partial solution.

Remark 1: A valid partial solution must have $0 \leq c_t \leq u_t$ for each t, $1 \leq t \leq m$. Obviously, the partial solution corresponding to the empty path will have $c_t = 0$ for all t. Similarly, the partial solution corresponding to the complete path will have $c_t = u_t$ for all t.

In the partial solution state definition, observe that we have not distinguished individual nets because all nets must use exactly the same segment types and layer assignment due to the exact matching constraint. The following lemma states the optimal substructure property.

Lemma 1: Consider two partial solutions A and B corresponding to the same partial solution state. If the routing cost of A is less than the routing cost of B, then it is guaranteed that B cannot exist in any optimal solution.

Proof: It is straightforward to show that the partial solution corresponding to B in any full solution can be replaced by A, and this will lead to a lower cost solution. ■

Based on the definition of partial solution state (Definition 5) and the optimal substructure property (Lemma 1), we can use standard DP techniques to solve the exactly matched maze routing problem as shown in Fig. 7. Here, we start from the initial state where $c_t = 0$ for each t ($1 \leq t \leq m$), and $last\_layer$ is set to the layer of the driver terminal. Then, we iteratively process the partial solution states in topological order. A partial solution state ps_j is defined to be topologically after another state ps_k if and only if it is possible to reach ps_k by adding unit segments from $\mathcal{U}_S$ to ps_j. When a partial state $currSt$ is processed in Fig. 7, we try to expand to new states by adding a new unit segment from $\mathcal{U}_S$. The edge cost between the current state and the new state is computed based on the congestion and via costs of adding the new unit segment to the partial routes of all nets. After all solution states are processed, we construct the routing solution by backtracing the parent pointers from the last solution state, where $c_t = u_t$ for each t ($1 \leq t \leq m$).

Theorem 1: The algorithm in Fig. 7 computes the optimal route matching solution for a given feasible set of unit segments $\mathcal{U}_S$. The space and time complexity of this algorithm is $O(m \cdot \ell^2 \cdot (u_{max})^m)$, where m is the number of segment types, ℓ is the number of layers, and u_{max} is the largest u_t value ($1 \leq t \leq m$). When m and ℓ are small constants, as is typically the case in practice, the complexity of the algorithm becomes $O((u_{max})^m)$.

```
EXACTLY-MATCHED MAZE ROUTE (𝒩, 𝒰_S)
  Initialize all state costs to infinity
  Set the cost of initial state (Remark III.1) to zero
  For each partial state currSt in topological order do
    For each newSt reachable from currSt in single step do
      If cost(newSt) > cost(currSt) + edge_cost(currSt → newSt)
        cost(newSt) := cost(currSt) + edge_cost(currSt → newSt)
        parent(newSt) := currSt
  Backtrace parent pointers from the last state (Remark III.1)
```

Fig. 7. DP-based algorithm to compute matching routes for a set of nets $\mathcal{N}$ using the unit segments in $\mathcal{U}_S$.

Proof: From Definitions 3 and 4, the routes computed must be complete (i.e., starting from the driver and ending at the receiver of each net) and exactly matching. The optimality is due to the optimal substructure property stated in Lemma 1. From Definition 5, the number of partial solution states is $O(\ell \cdot (u_{max})^m)$. In the algorithm of Fig. 7, there are at most $O(m \cdot \ell)$ expansions from each partial solution state, leading to the complexity stated in the theorem. ■

Until now, for simplicity of presentation, we have assumed that the edge costs are static, i.e., they do not dynamically change during path computations. However, it is possible that some routing segments in the same path may overlap with each other. Even in global routing, self overlaps are not desirable, because the congestion costs are computed assuming that a single segment passes through each edge. For example, consider an edge that has (capacity-usage)=1. If one of the two segments s_1 or s_2 passes through this edge, it will have zero congestion. However, if both pass at the same time, the congestion value will be one. Note that the main difficulty here is to detect this during path computations, when $s1$ and $s2$ belong to the same solution.

This is also known as the dynamic path optimization problem, where the edge costs change based on the partial paths during computation. In general, this is an NP-complete problem, and we handle it in a conservative way by avoiding self overlaps. In particular, every time we expand from a partial solution state, we check whether the new edges (for all nets) overlap with any segment in the partial solution. For efficient implementation, we keep track of longer back-pointers corresponding to the maximal horizontal and vertical segments. For example, in the solution of Fig. 6, we can backtrace from the final state (R_1, R_2) to the initial state in seven steps using the longer back-pointers, regardless of the lengths of the individual segments. In general, this check is expected to increase the runtime of the proposed algorithm by a factor of the number of bends in a partial solution.

C. Scalability Considerations

Theorem 1 states that the complexity of the proposed maze routing algorithm is $O((u_{max})^m)$, where m is the number of segment types, and u_{max} is the maximum number of unit segments of the same type in $\mathcal{U}_S$. Note that the segment types are defined based on the elements in L_H and L_V vectors, each of which is guaranteed to have size less than or equal to the number

of nets in $\mathcal{N}$ [8]. Since the number of nets to be exactly matched is typically a small constant (e.g., between two and five), m is expected to be small as well. However, if u_{max} is large, then there may still be practical problems. Fortunately, due to the way we formulated our maze routing algorithm, it is possible to keep both u_{max} and m at reasonable values.

Observe that our proposed dynamic programming algorithm in Fig. 7 does not rely on any assumptions about the lengths of the unit segments in $\mathcal{U}_S$. So, it is possible to keep u_{max} at a reasonable value by using unit segment lengths larger than one. For example, in Fig. 5, we have ten copies of unit segment type one in $\mathcal{U}_S$, because the length of the original segment is ten in Fig. 3(b). If we change the unit size to two, then we will have five copies of segment type one in $\mathcal{U}_S$ instead. Intuitively, if a segment type has many copies in $\mathcal{U}_S$, then it means that there is more flexibility in the maze router to avoid congestion. So, for such segment types, we can increase the unit segment lengths (and hence, reduce the number of copies in $\mathcal{U}_S$) without reducing the flexibility of maze routing significantly. Note that a pattern router such as [8] corresponds to the extreme case where each segment type has a single copy of the original size in $\mathcal{U}_S$. Obviously, this would restrict the solution space significantly. In contrast, our formulation allows a tradeoff between solution quality and runtime by choosing an appropriate unit segment length.

The second way to keep our algorithms scalable is to limit the number of segment types m. The basic idea is to iteratively apply maze routing on a subset of segment types at a time, and then build the full solution by merging these partial maze routing solutions. The details of this framework will be described in Section IV.

Note that the straightforward maze routing algorithm (Section III-A) can use neither of these two techniques to make it scalable. Since there are no segment types to guide the search process, its complexity is $O(|\mathcal{G}|^{|\mathcal{N}|})$, where $|\mathcal{G}|$ is the grid size, and $|\mathcal{N}|$ is the number of nets. Increasing the step size of the search process for such an algorithm can easily lead to cases where there is no valid exactly matched solution. On the other hand, our algorithm is still guaranteed to find exactly matched solutions, independent of the unit segment lengths in $\mathcal{U}_S$.

Another note is about the algorithm proposed in [8]. Although, the main idea of [8] is based on pattern routing, the authors also showed how to split long segments to increase the flexibility of pattern routing. In the complexity analysis of [8], it was shown that runtime and space complexity were both exponential in the number of additional bends. On the other hand, the algorithm we propose in this paper has polynomial time complexity as long as the number of segment types is constant, regardless of the number of bends in the solution. In our experimental study (Section VI), we will show how the runtime of [8] becomes impractically high when we increase the number of splits to more than one, while our algorithm can easily explore solutions with large numbers of bends.

IV. Scalable Route Matching Framework

In this section, we propose an exact route matching framework that allows the flexibility of maze routing, while allowing scalable execution on realistic problem sizes. For this, we start with an original topology such as the one shown in Fig. 1. This starting topology can be obtained by either the algorithms in [8], or by setting $u_t = 1$ for each segment type t in $\mathcal{U}_S$ (i.e., by having exactly one copy for each segment type as discussed in Section III-C). The basic idea of our framework is to apply the maze routing algorithm proposed in Section III-B between different corners of the original topology, and combine different maze routing results in an optimal way to minimize the total routing cost.

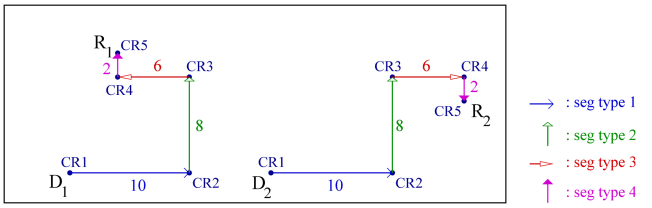

Fig. 8. Corners and the segment types illustrated for the topology shown in Fig. 1.

Fig. 8 shows the labeling of the corners in the original topology of Fig. 1. Observe that corner $CR1$ corresponds to the empty partial solution state (i.e., $\{c_1 = 0; c_2 = 0; c_3 = 0; c_4 = 0\}$ in Definition 5), whereas corner $CR2$ corresponds to the state after segment type 1 is fully utilized (i.e., $\{c_1 = 10; c_2 = 0; c_3 = 0; c_4 = 0\}$). The other corners $CR3$, $CR4$, and $CR5$ are defined similarly. Note that $CR5$ corresponds to the full solution state, where all segment types are fully utilized (i.e., $\{c_1 = 10; c_2 = 8; c_3 = 6; c_4 = 2\}$). Note that for simplicity of presentation, the basic idea is explained ignoring the existence of multiple layers. The actual implementation extends this idea to handle multiple layers.

We can define a graph structure, where each node corresponds to a corner of the original topology, and each edge corresponds to the best routing solution between the corresponding corners. Fig. 9 shows this graph model corresponding to the example of Fig. 8. Here, the edges between adjacent corners (e.g., $CR1 \rightarrow CR2$) correspond to routing between the respective corners with a single segment type. In other words, they correspond to the routing segments in the original topology without any maze routing. The edges between corners with distance two (e.g., $C1 \rightarrow C3$) correspond to the best exactly matched maze routing result between the respective corners using two segment types (e.g., segment types one and two for the edge $C1 \rightarrow C3$). In general, the edges between corners with distance d correspond to maze routing with d segment types.

As explained in Section III-C, the complexity of our exactly matched maze routing algorithm depends on both the number of unit segments (u_{max}) in $\mathcal{U}_S$ and the number of segment types (m). The runtime of maze routing can be kept within practical limits by controlling the step size in maze routing (i.e., the length of the unit segments in $\mathcal{U}_S$), and the number of segment types processed. The graph model we propose in this section allows controlling both parameters to obtain a good tradeoff between routing quality and runtime. In particular, we can run fine-grain maze routing (i.e., with small step sizes) for the graph edges corresponding to two or three segment types. For these cases, u_{max} can be large because of small m values. On the other hand,

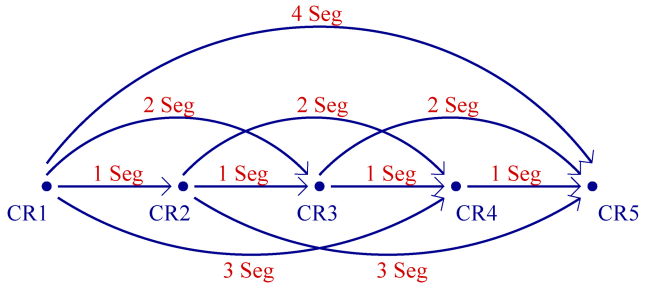

Fig. 9. Graph model corresponding to the corners and segment types shown in Fig. 8.

for the edges corresponding to the segment types with larger than three segments, we can run coarse-grain maze routing (with large step sizes). In other words, u_{max} can be kept small to allow larger m values.

As an example, consider the direct edge from $CR1$ to $CR5$ in Fig. 9. This edge allows interleaving of all four segment types shown in Fig. 8 in the same maze routing run. However, since $m = 4$, we may need to keep u_{max} small by increasing some of the unit segment lengths in $\mathcal{U}_S$, leading to coarse-grain maze routing with large step sizes. In contrast, consider the two edges $CR1 \rightarrow CR3$ and $CR3 \rightarrow CR5$. Each of these edges corresponds to a maze routing run with $m = 2$. Since m is small, u_{max} value can be kept larger, allowing fine-grain maze routing with small step sizes. Note that it is not known which of the two paths ($CR1 \rightarrow CR5$ or $CR1 \rightarrow CR3 \rightarrow CR5$) will have smaller cost, because the heuristics utilized are different. However, the objective here is to generate a variety of sub-solutions so that we can choose the best combination to construct the final path.

After computing the maze routing solution corresponding to each graph edge, we can set the edge costs to be equal to the routing cost of the corresponding maze route. Computing the shortest path on this graph will give us the best combination of different maze routing solutions. Observe that the proposed model is a directed acyclic graph, and the shortest path can be computed by processing each corner node in topological order, i.e., in increasing order of corner indices. Before processing node i, one can update the congestion costs of the routing grid based on the grid edges utilized in the partial solution up to node i. This way overlaps between subsolutions corresponding to different edges of this graph can be taken into account.

Once the shortest path is computed, we can substitute the maze routing solutions corresponding to the edges on the shortest path to obtain the final result.

Theorem 2: The routing solution obtained after merging the maze routing results of the individual edges on the shortest path is guaranteed to be exactly matching for all nets.

Proof: The maze routing solution corresponding to any edge is guaranteed to be exactly matching for all nets. Stitching these routes together will lead to exactly-matched routes for all nets. ■

Our overall framework can be summarized as follows. We first choose an initial routing topology for the given set of nets by either 1) solving the system of equations described in Section II, or 2) using the ILP-based methodology proposed in Section V. Then, we create the graph model $\mathcal{G}$ described in this section. We compute the cost of each edge in $\mathcal{G}$ by using our maze routing algorithms proposed in Section III-B. After that, the shortest path for $\mathcal{G}$ is computed, and the solution is translated to the individual matching routes.

V. Exploring Multiple Initial Routing Topologies

In Definition 4, we have defined the exactly matched maze routing problem based on a set of unit segments $\mathcal{U}_S$, which is obtained by solving the equations $C_H \times L_H = D_H$ and $C_V \times L_V = D_V$. As explained before, a single solution to each of these equations is proposed in [8], and the other potential solutions are ignored because a large solution space can be explored starting from the single solution. In this section, we propose a method to generate multiple solutions to these equations to understand whether further congestion minimization is possible by exploring multiple initial topologies in our maze routing framework.

A. Basic ILP Formulation

We first propose an alternative method to solve the equation $C \times L = D$ using ILP. Note that this method can be applied to solve both equations $C_H \times L_H = D_H$ and $C_V \times L_V = D_V$, corresponding to the horizontal and vertical segments, independently.

Given a distance vector D, our objective is to compute a configuration matrix C and a segment length vector L such that $C \times L = D$. Observe that both C and L are unknown, and only D is known. So, the formulation is not linear in its original form. We need to convert this to a linear formulation to be able to use ILP.

Let the number of elements in D be denoted as n, and let the desired number of segments in L be m. For equation $C \times L = D$ to hold, C must have n rows and m columns. Let c_{ij} ($1 \leq i \leq n$ and $1 \leq j \leq m$) be the element of C at the ith row and jth column. Let l_j be the jth element of L, and d_i be the ith element of D. The constraints corresponding to $C \times L = D$ can be written as

$$\sum_{j=1}^{m} c_{ij} \times l_j = d_i \qquad \text{for } 1 \leq i \leq n. \tag{2}$$

Observe that the constraints defined in 2 are not linear and cannot be modeled as ILP constraints as is. However, we can make use of the fact that each c_{ij} value must be either -1 or 1 (Section II). Let us define a new set of variables k_{ij} such that $k_{ij} = c_{ij} \times l_j$, for $1 \leq i \leq n$ and $1 \leq j \leq m$. Clearly, k_{ij} is equal to either l_{ij} or $-l_{ij}$. We can also define a binary variable b_{ij} corresponding to each k_{ij} such that

$$k_{ij} = \begin{cases} l_j & \text{if } b_{ij} = 1 \\ -l_j & \text{if } b_{ij} = 0. \end{cases} \tag{3}$$

For the purpose of enforcing this semantic, the following linear constraints can be defined in our ILP model:

$$\begin{aligned} b_{ij} &\in \{0, 1\} \\ -l_j &\leq k_{ij} \leq l_j \\ k_{ij} &\leq -l_j + M \times b_{ij} \\ k_{ij} &\geq l_j + M \times (b_{ij} - 1) \\ &\text{for } 1 \leq i \leq n \text{ and } 1 \leq j \leq m \end{aligned} \tag{4}$$

where M is chosen to be a very large constant (i.e., $M \gg l_j$ for any j).

Lemma 2: The semantic defined in (3) is guaranteed to hold if the set of linear constraints in (4) is satisfied.

Proof: If $b_{ij} = 1$, the third constraint in (4) will always be satisfied because $M \gg l_j$, and the fourth constraint will reduce to $k_{ij} \geq l_j$. Due to the second constraint in (4), we must have $k_{ij} = l_j$. Similarly, if $b_{ij} = 0$, then the fourth constraint will always hold, and the third constraint will reduce to $k_{ij} \leq -l_j$. Due to the second constraint, this implies that $k_{ij} = -l_j$. ■

After that, we can add the original constraints of $C \times L = D$ in linear form

$$\sum_{j=1}^{m} k_{ij} = d_i \qquad \text{for } 1 \leq i \leq n. \tag{5}$$

Finally, the ILP objective function can be defined as wirelength minimization by adding the following:

$$\text{minimize} \quad \sum_{j=1}^{m} l_j. \tag{6}$$

In summary, the ILP model can be set up using the set of constraints defined in (4) and (5), and the objective function in (6) for horizontal and vertical segments independently. The ILP constraints guarantee that the starting topologies satisfy the exact matching constraints, while the objective function makes sure that no unnecessary detour is introduced in the topologies. Once a solution is computed by the ILP solver, the segment length vector L can be constructed using the returned l_j values. Similarly, the configuration matrix C can be constructed using the b_{ij} values and (3).

B. Generating Multiple ILP Solutions

In this section, we extend the ILP formulation proposed in Section V-A to generate multiple routing topologies. The basic idea is to incrementally add extra ILP constraints such that the next solution returned is not identical to the previous solutions. However, there are various issues to consider as detailed in the next subsections.

1) *Canonical Representation:* Before describing how to generate multiple solutions, we need to ensure that the different solutions returned by ILP are not trivially similar to each other. For example, we have seen that it is possible to permute the columns of the C matrix to obtain solutions with different segment orders. Since our algorithms can already explore different segment orders efficiently for a given C matrix, there is no need to generate multiple solutions that differ only in the order of the wire segments. The reason is that solving ILP takes much longer than our proposed algorithms.

Fig. 10(a) shows an example solution for the equation $C \times L = D$, for a given distance vector D for three nets. According to Equation 3, there is a one-to-one correspondence between the entries in the C matrix and the b_{ij} values. The b_{ij} values corresponding to the C matrix in Fig. 10(a) are shown in matrix B. One can generate a different solution to the equation $C \times L = D$ by simply changing the order of the columns in C and the corresponding rows of L, as shown in Fig. 10(b). Since such segment reorderings are already explored in our algorithms

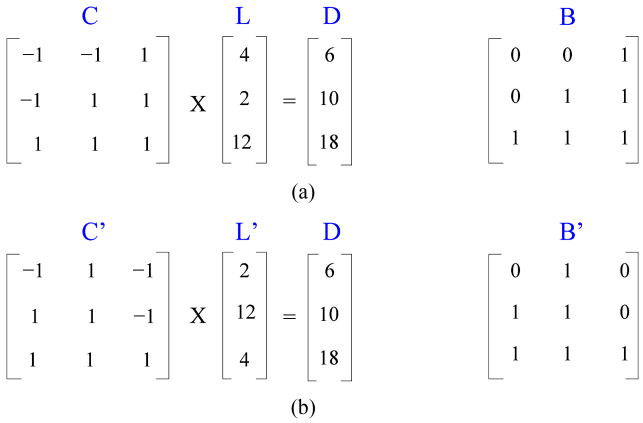

Fig. 10. (a) Example solution for equation $C \times L = D$, and the corresponding b_{ij} values (shown as matrix B). (b) Reordering of the columns of the C matrix and the corresponding rows of the L vector leads to a different solution. The B matrix in (a) is defined to be in canonical form, while the matrix B' in (b) is not.

efficiently, we do not want to generate different ILP solutions for different column permutations of the same C matrix. For this reason, we propose a canonical representation based on the b_{ij} values as follows.

Let $columnWt$ corresponding to the jth column of matrix C be defined as

$$columnWt(j) = \sum_{i=0}^{n} b_{ij} \cdot 2^{i-1}. \tag{7}$$

Intuitively, $columnWt(j)$ can be thought as the binary number corresponding to column j of the B matrix. In the example of Fig. 10(a), the $columnWt$ values for the three columns of the B matrix are four, six, and seven, respectively. Based on this definition, the following linear constraints can be added to our ILP formulation to define and enforce a canonical representation

$$columnWt(j) \leq columnWt(j+1) \qquad \text{for } 1 \leq j < m. \tag{8}$$

These constraints ensure that different column permutations of the same C matrix are not returned as different ILP solutions. For example, in Fig. 10, only the solution in part (a) satisfies these canonical constraints, and the solution in part (b) will not be returned as an ILP solution.

2) *Quality Control:* While generating multiple routing topologies, a minimum level of quality can be enforced for the returned solutions. Since the main quality metric in this stage is wirelength, we can add a constraint to limit the total length of the wire segments. It was shown in [8] that it is always possible to find an exactly matching routing topology where the total horizontal (vertical) wirelength is equal to the maximum horizontal (vertical) distance between the net pins, i.e., the maximum value in the D_H (D_V) vector. For example, in Fig. 10(a), the maximum element of the D vector is 18, and the sum of the segment length values in L is also 18.

```
GENERATE_MULTIPLE_ILP_SOLUTIONS (D, m)
  // D: the distance matrix, m: # of wire segments
  topologySet ← ∅
  for j ← 1 to m do
    formulate ILP using constraints (4), (5), (7), (8), (9),
                        and the objective function (10)
    ILP_soln ← solve ILP
    while ILP_soln is feasible
      topologySet ← {matrix C, vector L} from ILP_soln
      solnWt(j) ← columnWt(j) of ILP_soln
      add constraint (11) to the last ILP formulation
      ILP_soln ← solve ILP
  return uniquified topologySet
```

Fig. 11. Method to generate multiple initial routing topologies for maze routing.

In accordance with this, we add the following constraint to our ILP formulation to ensure that none of the generated solutions have suboptimal wirelength:

$$\sum_{j=1}^{m} l_j \leq d_{max} \tag{9}$$

where d_{max} is the maximum element of the D vector. Once this constraint is added, there is no need for the ILP objective function (6), which was defined in the basic formulation. The reason is that the sum of the horizontal (vertical) segment lengths in a matched routing solution must be at least equal to the maximum horizontal (vertical) distance between a pair of terminals.

3) *Multiple ILP Solutions:* Based on the canonical representation explained above, we propose the method in Fig. 11 to generate multiple solutions for our ILP formulation. Here, the basic idea is to enumerate different solutions with different *columnWt*(j) values, as defined in (7). Note that if the *columnWt*(j) values are different for two ILP solutions, it is guaranteed that the corresponding routing topologies will be different due to the canonical constraints defined in (8). Based on this idea, we process each column j ($1 \leq j \leq m$) independently, and define the following ILP objective function:

$$\text{minimize} \quad \text{columnWt}(j). \tag{10}$$

After solving ILP the first time in iteration j, we define *solnWt*(j) to be the *columnWt*(j) of the returned solution. To ensure that the same routing topology is not returned in the next call to the ILP solver, we add the following constraint:

$$\text{columnWt}(j) > \text{solnWt}(j). \tag{11}$$

Then, we make another call to the ILP solver, and obtain a new solution. Due to the constraint of (11), the new solution should be different than the previous one. We repeat these steps until a feasible ILP solution cannot be found. At that point, we increment j, and repeat the same set of steps for column $j+1$. Note that although the solutions generated in the same iteration j are guaranteed to be different, it is possible that some solutions generated for different j values are identical. Before returning the set of routing topologies computed, we uniquify the set of solutions to filter out such duplicates.

The method described in Fig. 11 can be used to generate solutions for equations corresponding to the horizontal and vertical segments: $C_H \times L_H = D_H$ and $C_V \times L_V = D_V$. After finding the solution set for each of the horizontal and vertical directions, it is possible to generate different routing topologies by matching every solution in the horizontal solution set with every solution in the vertical solution set. In other words, the final routing topologies are obtained by computing the cross product of the horizontal and vertical solution sets.

C. Choosing Best Initial Topology

Each of the different routing topologies generated in the previous section can be used to construct the set of unit segments $\mathcal{U}_S$, as defined in Definition 3. The maze routing algorithms proposed in this paper optimize the congestion and via costs by computing the proper ordering of the unit segments in the given set $\mathcal{U}_S$. Potentially, the maze routing results can be improved by applying our proposed algorithms on different unit segment sets (each of which is obtained using the method outlined in Fig. 11) and choosing the one that leads to the best result.

To evaluate a particular unit segment set $\mathcal{U}_S$, it is possible to use the routing framework proposed in Section IV. As discussed before, this framework can be tuned to obtain a tradeoff between solution quality and runtime. For the purpose of choosing the best initial topology, we can tune this framework such that a quality estimate can be obtained in a relatively short time. One example is by limiting the edges in Fig. 9 to two or three segments and using larger step sizes during evaluations. After evaluating each initial topology, we can choose the one that minimizes the congestion and via costs. Afterwards, we can rerun the routing framework in Section IV with more emphasis on quality for the selected initial topology.

VI. Experimental Results

For comparison purposes, we have utilized the benchmarks used in the experimental study of [8], which were extracted from the ISPD'07 Global Routing (GR) Contest [7]. For this, GR solutions of Archer [13] were obtained to generate a congestion map for each design. Then, the granularity of the GR grid was increased by two, because it was proven in [8] that if all horizontal and vertical distances between pins are even values, then an exact matching solution is guaranteed to exist. After that, a set of two-terminal nets was chosen randomly in each benchmark to impose matching constraints on. The random selection was done only for the nets with driver-to-receiver distances between 100 and 200 GR grid cells to make the problem realistic. For all benchmarks except *newblue3*, the original routes of the chosen nets were congestion free, but many of the routing resources were fully utilized. The congestion metric of a route matching solution is defined to be the delta congestion due to the new routes. In other words, the congestion values are reported by subtracting the original congestion value (before routing the net group) from the new congestion value (after routing the net group).

In these benchmarks, the routing grid varies between 648 × 648 (*adaptec1*) and 1946 × 2512 (*newblue3*). The average Manhattan distance of the chosen 2-terminal nets is 270, and the standard deviation is 53. The exact matching constraints are enforced on groups of nets, where 40% have three nets, 34%

TABLE I
EXPERIMENTAL COMPARISON OF OUR ALGORITHM WITH EXISTING ALGORITHMS

	PATTERN-BASED [8] DEFAULT			PATTERN-BASED [8] 1-SPLIT			OUR ALGORITHM $\alpha = 0.2$			OUR ALGORITHM $\alpha = 1.0$		
Benchmark	cong cost	via cnt	runtime (sec)	cong cost	via cnt	runtime (sec)	cong cost	via cnt	runtime (sec)	cong cost	via cnt	runtime (sec)
adaptec1	410	73	0.01	339	81	3.76	283	75	0.41	237	156	0.47
adaptec2	35	72	0.01	16	76	0.24	21	40	0.40	8	60	0.41
adaptec3	95	72	0.01	60	77	1.91	47	53	0.51	27	86	0.52
adaptec4	15	67	0.01	6	71	0.11	11	41	0.51	2	52	0.51
adaptec5	175	75	0.01	137	79	2.61	120	58	0.45	87	113	0.49
newblue1	37	69	0.01	15	78	0.15	27	40	0.42	15	55	0.43
newblue2	74	62	0.02	51	70	0.98	43	50	0.44	25	85	0.48
newblue3	23	66	0.01	16	72	0.18	19	36	0.34	7	51	0.34
Avg	108	70	0.01	80	76	1.24	71	49	0.43	51	82	0.46

have four nets, and 26% have five nets, which were chosen randomly in [8]. For each benchmark circuit, there are about 50 such groups with matching constraints. The results of each benchmark are reported as average over these ~ 50 net groups.

Note that we use exactly the same benchmarks and reporting mechanisms used in [8] without any modifications, and further details about these benchmarks can be found in [8].

We have implemented our algorithms in C++, and used the GNU Linear Programming Kit (GLPK) version 4.34 to solve the ILP problems. We ran all the experiments on a 3GHz Xeon system with 75GB RAM. For each maze routing subproblem in our route matching framework, we have allowed four different segment types,[3] i.e., the maximum edge length in the graph defined in Section IV is four. We have also empirically set the unit segment lengths in $\mathcal{U}_S$ such that u_{max} <= 50 for m = 2, u_{max} <= 20 for m = 3, and u_{max} <= 5 for m = 4, based on the discussions in Section IV. Note that we have used exactly the same set of parameters for all benchmark circuits. For comparison, we used the binaries of the algorithms proposed in [8]. Note that comparison with [3] is not possible because of high runtime and space complexity of that algorithm. As explained in Section III-A, the complexity of [3] is $|\mathcal{G}|^{|\mathcal{N}|}$, where $|\mathcal{G}|$ is the grid size, and $|\mathcal{N}|$ is the number of nets to be matched in each group. In the benchmarks we use, the maximum grid size is 1946 × 2512, and the number of nets to be matched is up to five; hence, [3] is not scalable enough to handle these problems.

Table I shows our experimental results corresponding to a single initial topology as explained in Section III-B. The matching constraints were followed exactly by all algorithms, and the route mismatch is zero in all results. This indicates that the actual delay mismatch will be very small for these net groups even in the presence of nonuniform layer characteristics.[4]

In Table I, the first set of results (pattern-based default) are from the previous pattern-based algorithm in the default mode, as presented in [8]. The second set of results (pattern-based one-split) are from the same algorithm, but allowing one extra bend for each net group. In other words, splitting at most one segment is allowed during pattern routing to increase the solution space [8]. As can be seen here, the congestion costs improve due to the extended solution space explored, but the runtimes also increase. It was stated in [8] that those algorithms have exponential runtime/space complexity in the number of extra bends introduced. We have also tried to run the pattern based algorithm [8] with two extra bends, but those runs did not complete in more than one week due to the exponential increase in runtime and space complexity.

The results of our proposed algorithms are listed in Table I for α = 0.2 and α = 1.0, where α controls the tradeoff between congestion cost and via count minimization objectives in (1). Compared to the default mode of [8], our algorithm reduces congestion costs by 34% and via counts by 30% on average for α = 0.2. When α is increased to 1.0 (i.e., when the importance of congestion minimization is increased), the congestion costs are reduced by 53% in the expense of 18% increase in total via counts on average. Note that this shows the flexibility of our maze routing algorithms to reduce congestion costs by introducing additional bends and vias. As mentioned above, this flexibility is limited to only one extra bend in the algorithms of [8]. Although one-split mode of [8] reduces the congestion costs, the results of our proposed algorithm are still superior. Specifically, compared to the one-split mode pattern routing, our algorithm reduces the congestion costs by 11% and via counts by 35% on average for α = 0.2. When α is increased to 1.0, the congestion costs are reduced by 36% in exchange for 9% increase in via counts. Observe that the default pattern routing algorithm is much faster than ours, mainly because of the very restricted solution space explored. On the other hand, when one extra bend is allowed for [8], it becomes ~ 3x slower than our algorithm. This is despite the fact that our algorithm explores a solution space with arbitrary number of bends.

For the purpose of understanding the effectiveness of the methodology proposed in Section IV, we have collected further statistics as follows. For the benchmarks in Table I, the average number of corners in the initial topologies used by our algorithms is 8.45. In particular, 25% of the net groups had initial topologies with ten or 11 corners. As explained in Section IV, the final routes are chosen based on the shortest path on the graph model. We have observed that 24% of the edges on the chosen shortest paths have edge length of one, 54% have edge length of two, 15% have edge length of three, and 7% have edge length of four. The readers can refer to Fig. 9 for an example showing edges of different lengths.

[3]The problems with more than four segment types are still handled by stitching the results of multiple subproblems as described in Section IV

[4]Crosstalk issues are ignored in this discussion. In reality, the susceptible nets can be shielded to avoid mismatches due to different coupling capacitances.

TABLE II
EXPERIMENTS DEMONSTRATING THE IMPACT OF THE UNIT SEGMENT SELECTION

	OUR ALGORITHM SINGLE SET OF UNIT SEGMENTS								OUR ALGORITHM MULTIPLE SETS OF UNIT SEGMENTS							
	$\alpha = 0.2$				$\alpha = 1.0$				$\alpha = 0.2$				$\alpha = 1.0$			
Benchmark	cong cost	via cnt	weighted cost	runtime (sec)	cong cost	via cnt	weighted cost	runtime (sec)	cong cost	via cnt	weighted cost	runtime (sec)	cong cost	via cnt	weighted cost	runtime (sec)
adaptec1	283	75	132	0.41	237	156	393	0.47	292	63	121	14.94	205	167	372	15.33
adaptec2	21	40	44	0.40	8	60	68	0.41	23	34	38	64.27	7	50	57	58.61
adaptec3	47	53	62	0.51	27	86	113	0.52	56	46	57	36.72	25	74	99	37.86
adaptec4	11	41	43	0.51	2	52	54	0.51	16	36	39	16.01	1	45	46	16.30
adaptec5	120	58	82	0.45	87	113	200	0.49	135	51	78	44.95	90	109	199	45.88
newblue1	27	40	45	0.42	15	55	70	0.43	28	34	39	26.26	10	50	60	26.49
newblue2	43	50	59	0.44	25	85	110	0.48	52	44	54	12.17	22	74	96	12.38
newblue3	19	36	40	0.34	7	51	58	0.34	25	31	36	8.74	10	47	56	8.89
Avg	71	49	63	0.43	51	82	133	0.46	78	42	58	28.01	46	77	123	27.72

We have also performed experiments to explore how the solution quality depends on the initial set of unit segments $\mathcal{U}_S$. For this, we have compared two versions of our framework: 1) using a single set of unit segments $\mathcal{U}_S$ computed directly as explained in Section III-B, and 2) choosing the best set of unit segments $\mathcal{U}_S$ through the method explained in Section V. Table II compares the results of these two versions. In this table, congestion costs, via counts and runtimes are reported as before. In addition, the weighted cost values [as defined in (1)] are reported for a direct comparison of how well the two versions optimize the cost metric. For version two, on average 82 initial sets of unit segments have been computed using the ILP based method proposed in Section V-B, and the results reported in Table II under Multiple Sets of Unit Segments are the best results obtained after trying out all of them. Observe that even after trying out this many different starting configurations, the best results obtained for version two on average are only about 9% and 7% better than version one for α = 0.2 and α = 1.0, respectively.

In the experiments reported in Table II, only the initial topologies with the minimum number of segments were generated. In particular, it was shown in [8] that n horizontal segments and n vertical segments are sufficient to match n nets. For the purpose of expanding the solution space of the ILP problem, we have performed additional experiments by considering more segments than the minimum required. In particular, we executed our ILP-based algorithm for each m value up to and including five. This increased the number of initial topologies from 83 to 139 on average. This change improved the quality of the final results by 5% in exchange for about 60% increase in total runtime on average. When we tried to increase m beyond five, the runtime of the ILP solver increased significantly.

These results show that most of the improvement is achieved by using a single set of unit segments. In other words, it is more important to explore the solution space of a single set of unit segments effectively rather than trying out different sets and choosing the best one. On the other hand, if congestion and via minimization is very critical for certain groups of nets, one can use the ILP-based flow described in Section V to improve the results further in exchange for increased runtime.

VII. CONCLUSION

In this paper, we have proposed scalable maze routing algorithms that satisfy exact matching constraints for analog nets. Given an initial topology that satisfies the matching constraints, our algorithms can explore a large solution space to minimize the total routing cost while obeying the exact matching constraints of the initial topology. Compared to the previous works, our algorithms explore a much larger solution space efficiently by allowing extra bends in the matched routing results. We have also proposed a scalable routing framework to handle problems with realistic sizes. Our experiments show significant congestion reductions compared to the previous work. A potential future research problem can be how to generate better initial topologies or how to improve the maze search such that it is not restricted to the unit segments of the initial topology.

The algorithms proposed in this paper are for GR with layer assignment. If the detailed router (DR) used afterward maintains fidelity to the GR solution, the final detailed routes are expected to be well-matched too. For this, the analog nets with matching constraints can be prioritized over others during DR. Alternatively, the ideas proposed in this paper can be extended for DR applications, which can be another future research topic.

ACKNOWLEDGEMENTS

The authors would like to thank G. R. Wilke (Intel Corp.) for the insightful discussions on the ILP model and the routing algorithms.

REFERENCES

[1] M. Bai, P. Bai, C. Auth, S. Balakrishnan, M. Bost, R. Brain, V. Chikarmane, R. Heussner, M. Hussein, J. Hwang, D. Ingerly, R. James, I. Jeong, C. Kenyan, E. Lee, S-H. Lee, N. Lindert, M. Liu, Z. Ma, T. Marieb, A. Murthy, R. Nagisetty, S. Natarajan, J. Neirynck, A. Ott, C. Parker, J. Sebastian, R. Shaheed, S. Sivakumar, J. Steigenvald, S. Tyagi, C. Weber, B. Woolely, A. Yeoh, K. Zhang, and M. Bohr, "A 65nm logic technology featuring 35 nm gate length, enhanced channel strain, 8 Cu interconnect layers, low-k ILD and $0.57um^2$ SRAM cell," in *Proc. IEDM*, pp. 657–660, Dec. 2004.

[2] J. Cohn, D. Garrod, R. A. Rutenbar, and L. R. Carley, "KOAN / ANAGRAM II: New techniques for device-level analog placement and routing," *IEEE J. Solid State Circuits*, vol. 26, no. 3, pp. 330–342, Mar. 1991.

[3] Q. Gao, H. Yao, Q. Zhou, and Y. Cai, "A novel detailed routing algorithm with exact matching constraint for analog and mixed signal circuits," in *Proc. ISQED*, pp. 1–6, Mar. 2011.

[4] K. Lampaert, G. Gielen, and W. Sansen, "Analog routing for manufacturability," in *Proc. CICC*, 1996, pp. 175–178.

[5] P.-H. Lin, H.-C. Yu, T.-H. Tsai, and S.-C. Lin, "A matching-based placement and routing system for analog design," in *Proc. VLSI-DAT*, 2007, pp. 1–4.

[6] S. Mitra, S. K. Nag, R. A. Rutenbar, and L. R. Carley, "System-level routing of mixed-signal ASICs in WREN," in *Proc. ICCAD*, Nov. 1992, pp. 394–399.
[7] G.-J. Nam, "ISPD placement contest updates and ISPD 2007 global routing contest," in *Proc. ISPD*, Mar. 2007, p. 167.
[8] M. M. Ozdal and R. F. Hentschke, "Exact route matching algorithms for analog and mixed signal integrated circuits," in *Proc. ICCAD*, Nov. 2009, pp. 231–238.
[9] M. M. Ozdal and R. F. Hentschke, "Maze routing algorithms with exact matching constraints for analog and mixed signal designs," in *Proc. ICCAD*, Nov. 2012, pp. 130–136.
[10] M. M. Ozdal and M. D. F. Wong, "Algorithmic study of single-layer bus routing for high-speed boards," *IEEE Trans. Comput. Aided Design*, vol. 25, no. 3, pp. 490–503, Mar. 2006.
[11] M. M. Ozdal and M. D. F. Wong, "A length-matching routing algorithm for high-performance printed circuit boards," *IEEE Trans. Comput. Aided Design*, vol. 25, no. 12, pp. 2784–2794, Dec. 2006.
[12] M. M. Ozdal and M. D. F. Wong, "Two-layer bus routing for high-speed printed circuit boards," *ACM Trans. Design Autom. Electr. Syst.*, vol. 11, no. 1, pp. 213–227, Jan. 2006.
[13] M. M. Ozdal and M. D. F. Wong, "Archer: A history-based global routing algorithm," *IEEE Trans. Comput. Aided Design Integr. Circuits Syst.*, vol. 28, no. 4, Apr. 2009, pp. 528–540.
[14] T. Yan and M. D. F. Wong, "BSG-Route: A length-matching router for general topology," in *Proc. ICCAD*, Nov. 2008, pp. 499–505.

Muhammet Mustafa Ozdal received the B.S. degree in electrical engineering, and the M.S. degree in computer engineering from Bilkent University, Ankara, Turkey, in 1999 and 2001, respectively, and the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana, IL, USA, in 2005.

He is currently a Research Scientist with the Strategic CAD Laboratories of Intel Corporation, Hillsboro, OR, USA. He has served in the technical program committees of ICCAD, DAC, DATE, ISPD and ISLPED. He was the Program/General Chair of SLIP 2012/2013, and the Contest Chair of ISPD 2012/2013. His research interests include heterogeneous computing, hardware/software co-design, and algorithms for VLSI CAD.

Dr. Ozdal was a recipient of the IEEE William J. McCalla ICCAD Best Paper Award in 2011, and the ACM SIGDA Technical Leadership Award in 2012.

Renato Fernandes Hentschke received the bachelor's, master's, and Ph.D. degrees in computer science from the Federal University of Rio Grande do Sul (UFRGS), Porto Alegre, Brazil.

Since 2007, he has been with Intel Corporation, Hillsboro, OR, USA, where he is engaged in router engines. Prior to Intel, he was a Temporary Professor with the State University of Rio Grande do Sul (Guaiba, RS, Brazil) for six months and interned with IBM, Yorktown Heights, NY, USA, as an intern student for nine months. He holds a U.S. patent and has co-authored 20 papers in the field of computer aided design of ICs. He is a reviewer of prestigious conferences, such as DAC and ICCAD. His current research interests include algorithms for physical design of VLSI circuits, such as placement, routing, and layout verification.