Maze Routing Algorithms with Exact Matching Constraints for Analog and Mixed Signal Designs

Muhammet Mustafa Ozdal
Strategic CAD Labs
Intel Corporation
Hillsboro, OR 97124
mustafa.ozdal@intel.com

Renato Fernandes Hentschke
Core CAD Technologies
Intel Corporation
Hillsboro, OR 97124
renato.f.hentschke@intel.com

***Abstract*—Design automation for analog and mixed signal designs has become more important, as analog and digital components are integrated on the same system-on-chips (SOCs). Exact route matching is an important constraint for analog and mixed signal designs with non-uniform 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. The experimental study demonstrates that the proposed algorithm leads to significant reductions in congestion costs compared to the previous algorithm.**

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 semi-automatic 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.

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 *65nm* Intel logic technology are *220nm, 280nm, 330nm, 480nm, 720nm,* and *1080nm* for metal3 to metal8, 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. Obviously, 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, the locations and the number of vias, order of wire segments, etc.) that make it hard to match the electrical characteristics of multiple routes.

IEEE/ACM International Conference on Computer-Aided Design (ICCAD) 2012, November 5-8, 2012, San Jose, California, USA
Copyright ©2012 ACM 978-1-4503-1573-9/12/11... $15.00

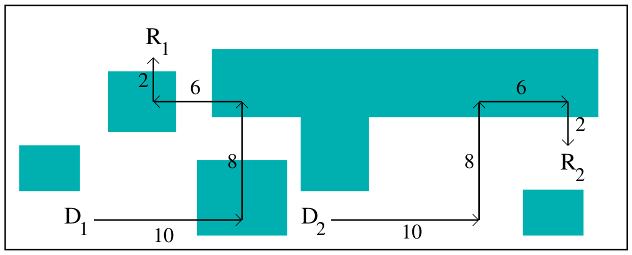


Fig. 1. Exactly matching routes for 2 nets with (driver, receiver) pairs (D_1, R_1) and (D_2, R_2). The congested regions are shown as shaded rectangles.

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 i^{th} 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 [9], [10], [11], [12]. When each net has a different horizontal and vertical distance between its driver and receiver, it is especially difficult to satisfy all of these 3 conditions at the same time, while minimizing the total routing costs. For example, Figure 1 illustrates two nets with exactly matching routes. Observe that, for both routes, a horizontal segment of length 10 is connected to the driver, followed by a vertical segment of length 8, a horizontal segment of length 6, and a vertical segment of length 2. 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 2 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 [8], dynamic programming (DP)

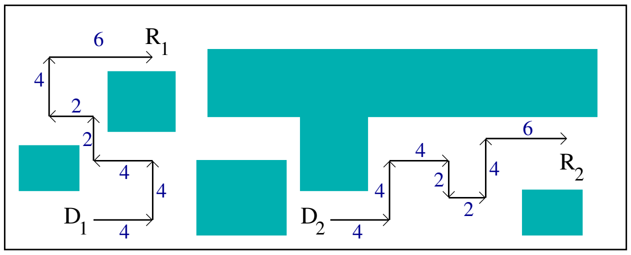


Fig. 2. A different exactly-matched routing solution for the problem shown in Figure 1. All congested regions (shaded rectangles) are avoided in this case.

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 Figure 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 (such as [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. Figure 2 shows a sample maze routing solution for the problem of Figure 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 starts with an initial topology and computes 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. In our experimental study (Section V), we show that our proposed algorithms can significantly improve congestion compared to [8].

II. Preliminaries and Related Work

The length matching problem has been studied extensively in the literature for board-level routing [9], [10], [11], [12]. 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 3 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 linear equations for the horizontal and vertical segments: $C_H \times L_H = D_H$ (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 j^{th} 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 j^{th} 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. Figure 3 shows the corresponding linear equations for the horizontal and vertical segments of Figure 1.

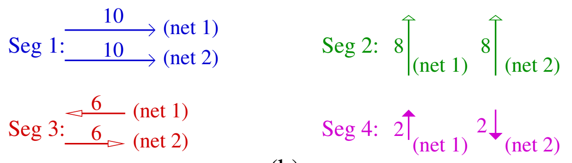
It was also shown in [8] that there is a class of configuration matrices C_m that leads to a feasible matching solution for any set of horizontal or vertical distance vectors D_H or D_V. Authors use these configuration matrices to compute the segment lengths L_H and L_V corresponding to any given distance 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 Figure 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 V, we show in our experimental study that it is not possible to run [8] with extra bend counts more than 1 within reasonable runtime limits, due to the exponential runtime.

In another work, [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. The authors of [3] proposed

$$\overset{C_H}{\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}} \times \overset{L_H}{\begin{bmatrix}10\\ 6\end{bmatrix}} = \overset{D_H}{\begin{bmatrix}4\\ 16\end{bmatrix}} \qquad \overset{C_V}{\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}} \times \overset{L_V}{\begin{bmatrix}8\\ 2\end{bmatrix}} = \overset{D_V}{\begin{bmatrix}10\\ 6\end{bmatrix}}$$

(a)



(b)

Fig. 3. (a) The equations for the example in Figure 1. (b) The corresponding segment lengths and directions.

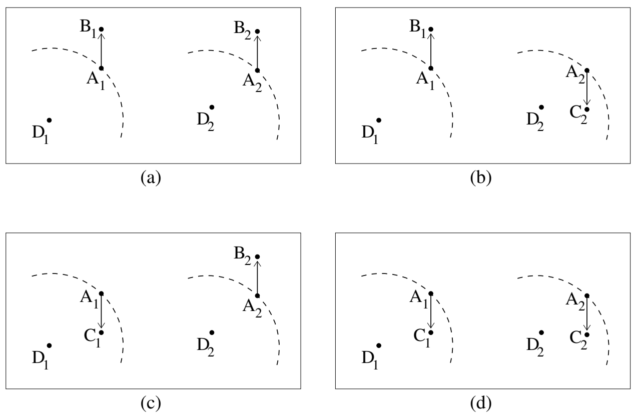


Fig. 4. Simultaneous wavefront expansion for two nets. D_1 and D_2 correspond to the drivers of nets 1 and 2, respectively. Adding a vertical segment to the current partial solution state (A_1, A_2) leads to four new solution states: (a) state (B_1, B_2), (b) state (B_1, C_2), (c) state (C_1, B_2), (d) state (C_1, C_2).

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. Furthermore, exact matching is not necessarily guaranteed for a non-uniform grid structure.

In this paper, we propose a maze routing algorithm for nets with exact matching constraints. Figure 2 shows the maze routing solution corresponding to the problem in Figure 1, which avoids all congested regions while still matching the routes of all nets exactly. Our 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 algorithms 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 the problem we focus on, 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].

Figure 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 Figure 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 1 and 2, respectively.

The main problem with this approach is the impractically high runtime and space complexity. As in the example of Figure 4, simultaneous wavefront expansion requires keeping track of a 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 $10K$ 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 2 and 5). 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 3 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 non-uniform 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 III.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 3 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 III.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]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.

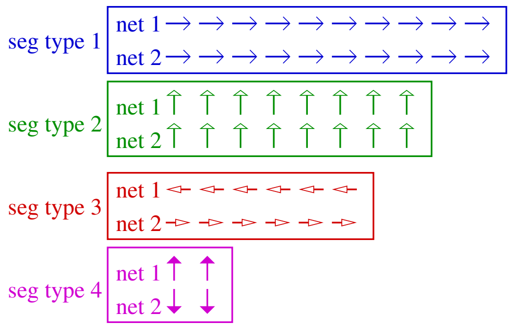


Fig. 5. The set of unit segments $\mathcal{U}_S$ corresponding to the example in Figure 3 (b).

($1 \leq i \leq k$) has a specific: 1) segment type, 2) segment length, and 3) layer assignment.

Consider the example in Figure 1. Here, the matched routing solution consists of a sequence of 4 segments with types $(\rightarrow;\rightarrow)$, $(\uparrow;\uparrow)$, $(\leftarrow;\rightarrow)$, $(\uparrow,\downarrow)$, and lengths 10, 8, 6, and 2, respectively. Similarly, the routing solution in Figure 2 consists of a sequence of 7 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 III.3. *Let $\mathcal{U}_S$ denote a set of unit segments, where each segment in $\mathcal{U}_S$ has a specific segment 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.*

Figure 5 shows a set of unit segments $\mathcal{U}_S$, in which there are 10, 8, 6, and 2 unit segments of types 1, 2, 3, and 4, respectively. Observe that each segment type has a specific direction for each of the two nets. Figure 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.

In this paper, our main focus is on how to explore the solution space defined by a given $\mathcal{U}_S$, rather than how to compute a feasible $\mathcal{U}_S$. Even for a fixed $\mathcal{U}_S$, the size of the solution space can be very large. For the example of Figure 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. For the same example, such pattern routing algorithms would explore only 8 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 linear 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 Figure 1, the corresponding equations for horizontal and vertical segments are shown in Figure 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 Figure 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.

[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 1 bend in those algorithms due to unacceptable runtime overheads.

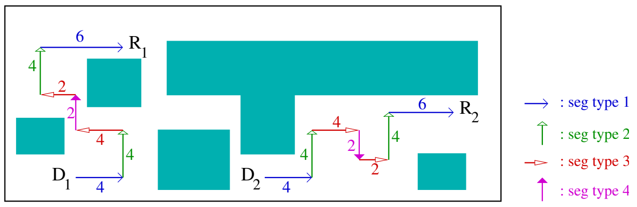


Fig. 6. An exactly matched routing solution for two nets utilizing all the unit segments in $\mathcal{U}_S$ of Figure 5.

Definition III.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.

The exact route matching problem as formulated in Definition III.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 III.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 III.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 III.2. *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 III.5) and the optimal substructure property (Lemma III.2), we can use standard dynamic programming (DP) techniques to solve the exactly

```
EXACTLY-MATCHED MAZE ROUTE ($\mathcal{N}$, $\mathcal{U}_S$)
 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 $\rightarrow$ newSt)
          cost(newSt) := cost(currSt) + edge_cost(currSt $\rightarrow$ 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 $\mathcal{U}_S$

matched maze routing problem as shown in Figure 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 Figure 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 III.3. *The algorithm in Figure 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 $O((u_{max})^m)$, where m is the number of segment types, and u_{max} is the largest u_t value ($1 \leq t \leq m$). Here, both m and the number of layers are assumed to be small constants.*

Proof: From Definitions III.3 and III.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 III.2. From Definition III.5, the number of partial solution states is $O((u_{max})^m)$. In the algorithm of Figure 7, there are at most $O(m)$ expansions from each partial solution state, which is also constant. Hence the theorem follows. ■

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 1. 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 the maximal segments in the partial solutions. In this way, we avoid extra overheads due to backtracing of the individual parent pointers.

C. Scalability Considerations

Theorem III.3 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 2 and 5), 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 Figure 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 1. For example, in Figure 5, we have 10 copies of unit segment type 1 in $\mathcal{U}_S$, because the length of the original segment is 10 in Figure 3 (b). If we change the unit size to 2, then we will have 5 copies of segment type 1 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 V), we will show how the runtime of [8] becomes impractically high when we increase the number of "splits" to more than 1, 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 Figure 1. This starting topology

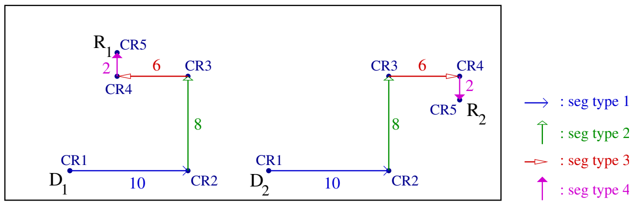


Fig. 8. The corners and the segment types illustrated for the topology shown in Figure 1.

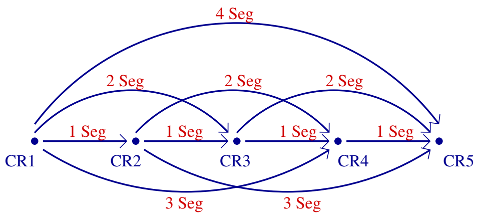


Fig. 9. The graph model corresponding to the corners and segment types shown in Figure 8.

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.

Figure 8 shows the labeling of the corners in the original topology of Figure 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 III.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. Figure 9 shows this graph model corresponding to the example of Figure 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 2 (e.g. $C1 \rightarrow C3$) correspond to the best exactly matched maze routing result between the respective corners using 2 segment types (e.g. segment types 1 and 2 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 2 or 3 segment types. For these cases, u_{max} can be large because of small m values. On the other hand, for the edges corresponding to the segment types with larger than 3 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 Figure 9. This edge allows interleaving of all 4 segment types shown in Figure 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. 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 IV.1. *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. ■

V. 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]. In these benchmarks, exact route matching constraints are enforced on groups of nets, where each group contains between 2 and 5 nets, which is the most typical size in practice. 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. These benchmarks also include a congestion map, which was obtained by running a global router on these benchmarks. In a typical mixed-signal design, the analog nets need to be routed in the same design with digital nets. It is assumed that the congestion map provided gives the resource usage information due to the (existing or predicted) routes of the digital nets. The congestion 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). Note that we use exactly the same benchmarks and reporting mechanisms used in [8] without any modifications.

We have implemented our algorithms in C++, and 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 4 different segment types[3], i.e. the maximum edge length in

[3]The problems with more than 4 segment types are still handled by stitching the results of multiple subproblems as described in Section IV.

TABLE I
EXPERIMENTAL RESULTS

	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

the graph defined in Section IV is 4. 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 also 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, $|\mathcal{G}|$ is $40,000$, and $|\mathcal{N}|$ is up to 5; hence, [3] is not scalable enough to handle these problems.

Table I shows our experimental results. In all cases, the matching constraints were followed exactly; hence the route mismatch is zero in all results. In this table, 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 1-split") are from the same algorithm, but allowing 1 extra bend for each net group. In other words, splitting at most 1 segment is allowed during pattern routing to increase the solution space (see [8] for details). 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 with 2 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 also listed in Table I for $\alpha = 0.2$ and $\alpha = 1.0$, where α controls the tradeoff between congestion cost and via count minimization objectives in Equation 1. Compared to the default mode of [8], our algorithm reduces congestion costs by 34% and via counts by 30% on average for $\alpha = 0.2$. 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 1 extra bend in the algorithms of [8]. Although 1-split mode of [8] reduces the congestion costs, the results of our proposed algorithm are still superior. Specifically, compared to the 1-split mode pattern routing, our algorithm reduces the congestion costs by 11% and via counts by 35% on average for $\alpha = 0.2$. 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 1 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.

VI. CONCLUSIONS

In this paper, we have proposed scalable maze routing algorithms that satisfy exact matching constraints for analog nets. 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.

REFERENCES

[1] M. Bai and et al., "A 65nm logic technology featuring 35nm gate length, enhanced channel strain, 8 cu interconnect layers, low-k ILD and $0.57um^2$ SRAM cell," in *Proc. of IEDM*, 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. of Solid State Circuits*, vol. 26, no. 3, 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. of ISQED*, 2011.

[4] K. Lampaert, G. Gielen, and W. Sansen, "Analog routing for manufacturability," in *Proc of 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. of 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. of ICCAD*, 1992.

[7] G.-J. Nam, "ISPD 2007 Global Routing Contest," 2007.

[8] M. M. Ozdal and R. F. Hentschke, "Exact route matching algorithms for analog and mixed signal integrated circuits," in *Proc. of ICCAD*, 2009.

[9] M. M. Ozdal and M. D. F. Wong, "Algorithmic study of single-layer bus routing for high-speed boards," *IEEE Trans. on CAD (TCAD)*, vol. 25, pp. 490–503, 2006.

[10] M. M. Ozdal and M. D. F. Wong, "A length-matching routing algorithm for high-performance printed circuit boards," *IEEE Trans. on CAD (TCAD)*, vol. 25, pp. 2784–2794, 2006.

[11] M. M. Ozdal and M. D. F. Wong, "Two-layer bus routing for high-speed printed circuit boards," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 11, pp. 213–227, 2006.

[12] T. Yan and M. D. F. Wong, "BSG-Route: A length-matching router for general topology," in *Proc. of ICCAD*, 2008.