BSG-Route: A Length-Matching Router for General Topology

Tan Yan and Martin D. F. Wong
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Email: {tanyan2, mdfwong}@illinois.edu

***Abstract*— Length-matching routing is a very important issue for PCB routing. Previous length-matching routers [1]–[3] all have assumptions on the routing topology whereas practical designs may be free of any topological constraint. In this paper, we propose a router that deals with general topology. Unlike previous routers, our router does not impose any restriction on the routing topology. Moreover, our router is gridless. Its performance does not depend on the routing grid size of the input while routers in [1]–[3] do. This is a big advantage because modern PCB routing configurations usually imply huge routing grids. The novelty of this work is that we view the length-matching routing problem as an area assignment problem and use a placement structure, Bounded-Sliceline Grid (BSG) [4], to help solving the problem. Experimental results show that our router can handle practical designs that previous routers can't handle. For designs that they could handle, our router runs much faster. For example, in one of our data, we obtain the result in 88 seconds while the router in [3] takes more than one day.**

I. Introduction

As the scale of electronic systems grows, the size of printed circuit boards (PCBs) also increases. Nowadays, a large scale board usually hosts thousands of nets [5], making manual design an extremely time consuming and error-prone task. On the other hand, the increasing clock frequency imposes various physical constraints, e.g., length-matching routing, pairwise routing, etc., on high performance PCBs [6], [7]. These constraints make traditional IC and PCB routers not applicable to modern PCB routing. To our knowledge, there is no mature commercial or academic automated router that handles these constraints well. The time consuming, error-prone manual design style is still the most popular way to design a high performance PCB. Therefore, automated PCB router that are tuned to handle such constraints becomes a necessity in modern design.

As an important issue in high performance board routing, bus routing with length-matching constraints (i.e., all the wires in a bus are expected to be routed with specified length bounds) has been researched in [1]–[3]. However, all these works are restricted by certain topological constraints: [1], [2] can only be applied when the two components are facing each other and [3] assumes that each wire must be routed monotonically in one direction, say, from left to right. These constraints limit the application of these routers. For example, none of the three routers can be applied to the bus in Fig. 1.

This work was partially supported by the National Science Foundation under grant CCF-0701821 and a grant from the Fujitsu Laboratories.

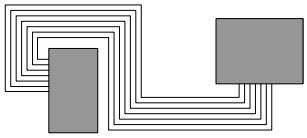


Fig. 1. A bus that cannot be routed by previous length-matching routers.

Another disadvantage of these routers is that they are all gridded. Modern PCBs usually have very fine wiring pitch and long distance between components, making the size of the routing grid very large for gridded routers even though the solution may look very simple.

In this paper, we propose a length-matching router that is capable of handling any given topology. The novelty in our approach is that we regard the length-matching routing problem as an area assignment problem and introduce a placement structure, Bounded-Sliceline Grid (BSG) [4], to help solving the problem.

Our router has the following virtues:

1) It is the first length-matching router that is free of any topological constraint. It handles general topology.

2) It produces a gridless routing. Its performance is not sensitive to the routing grid size of the input while previous routers' are. Therefore, our router performs better in practice because practical designs usually have very large routing grids.

3) It is area-focused. Experimental results show that it can efficiently utilize the routing area for wire extension to meet the length bounds.

The rest of this paper is organized as follows: Section II gives some necessary backgrounds for the understanding of our router. Section III describes our length-matching router for general topology. Experimental results are presented in Section IV and Section V concludes this paper.

II. Backgrounds

In this section, we will first introduce the length-matching routing problem and then give a brief review on the Bounded-Sliceline Grid (BSG) structure.

A. The Length-Matching Routing Problem

Bus routing for PCB is usually divided into two phases: escape routing and area routing (see Fig. 2). Escape routing is to route from the pin array inside a component to the boundary of the component. Area routing is to complete the connections between the boundaries. Escape routing and area routing have

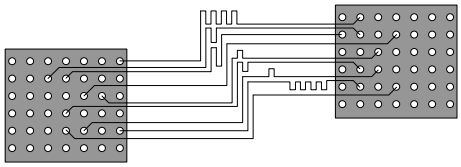


Fig. 2. Escape routing (inside the gray components) and area routing (between the two components).

different tasks. Escape routing must guarantee that the pin ordering on both boundaries are matched in order to provide a planar topology for area routing. The focus of area routing, on the other hand, is to carefully detour the wires to meet the length bounds while maintaining the planar topology inherited from escape routing. That is why we also call area routing "length-matching routing".

Due to the effort of escape router, the input to length-matching router is usually regarded to be planar. However, previous routers also require extra assumptions: [1] and [2] assume that the two boundaries to be connected are facing each other as in Fig. 2 and [3] assumes that the wires detour in only one direction, i.e., the routing can detour either horizontally or vertically but not both. As a result, none of them can solve the routing problem in Fig. 1. One remedy to this issue is to first route from both boundaries to some virtual boundaries facing each other and then perform length-matching routing between the virtual boundaries [8]. However, this means that the area around the components cannot be fully utilized for wire extension.

Furthermore, previous routers are all gridded. The ratio of the area of the input routing domain A to the minimum wire separation ε defines the size of the routing grid. We call this grid size *input routing grid size* because both A and ε are input-dependent. The performance of a gridded router is very sensitive to the input routing grid size. For example, whether the two components in Fig. 2 have a distance of 100ε or 10000ε makes a huge difference to the router, although the two problems look quite similar in human designers' eyes. Moreover, modern PCBs usually have very small ε and the components they host might be located far apart. This means that the input routing grid size can be very large, making gridded routers unbearably slow.

Now we formulate the general topology length-matching routing problem as follows:

Input:
- The location and size of the components and the location of the pins on the components' boundaries.
- A planar topology of the nets connecting the pins.
- A rectangular domain for the routing. All the wires are bounded within this domain.
- A separation rule $\varepsilon > 0$ for the wires. Any wire segment must be separated from other wire segments or the components by a distance of ε.
- Length bounds (l_i, u_i) for each net i. l_i is the lower bound and u_i is the upper bound. $l_i \leq u_i$.

Output: A rectilinear routing following the given topology and satisfying the separation rule. The length of each net i satisfies $l_i \leq length(i) \leq u_i$.

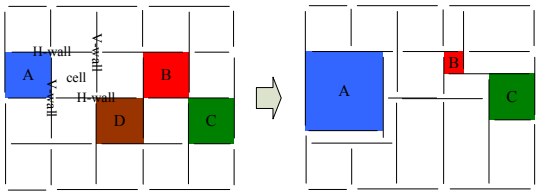


Fig. 3. The BSG structure and cell sizing.

B. Bounded-Sliceline Grid

Bounded-Sliceline Grid (BSG) [4] was invented to handle the placement of function modules in ICs. It uses a set of vertical and horizontal segments to partition the whole plane into rectangular cells. Each cell in the grid is bounded by four segments, two vertical (which we call *V-walls*) and two horizontal (which we call *H-walls*). Each wall spans over two cells. If we let all the cells be unit size (1×1), then every wall will have a length of 2 and the BSG will form a uniform grid. See the left side of Fig. 3 for an illustration.

The size and location of a cell are determined purely by the positions of the four walls surrounding it. By moving the walls, we can enlarge a cell (cell A in Fig. 3), shrink a cell (cell B), move a cell (cell C) or make a cell vanish (cell D).

III. Our BSG-Route

In this section, we will present our BSG-based length-matching router. We will first explain the general idea and then discuss the details.

A. The Idea

The key issue of length-matching routing is how to control the length of the wires. An interesting observation is that the length of a wire is the area it occupies divided by ε if we regard the wire as a fat wire with width ε (recall that ε is the separation rule, see Fig. 4). Therefore, instead of think about how to detour the wires to meet the length bounds, an alternative is to consider how to assign the area to the nets such that the assigned area of each net matches its length bounds. This idea leads to our length-matching router: BSG-route.

The flow of our router is as follows (see Fig. 5): First, we embed the given topology onto a BSG. Then, we size the cells in the BSG so that the total area of the cells occupied by a net satisfies its length bounds. Finally, we perform detail routing inside each cell to turn the assigned area into expected length. In the rest of this section, we will discuss each step in detail.

B. BSG Embedding

First, we need to map the components, pins and nets onto a BSG. This can be done either manually or by heuristic algorithms. There may exist multiple embeddings for the same

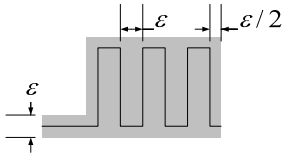
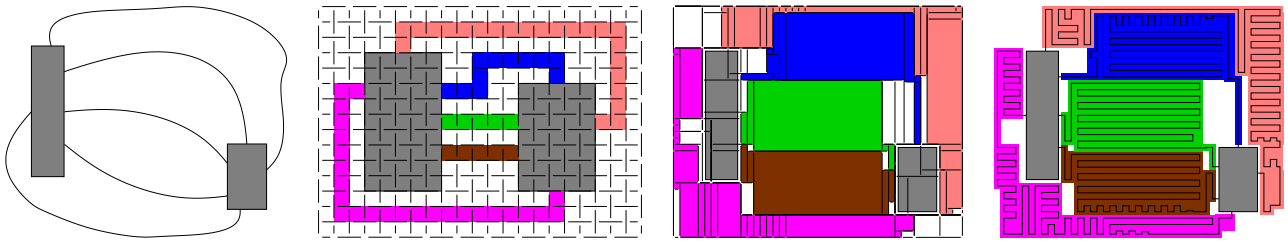


Fig. 4. The length of a wire is the area it occupies (gray area) divided by ε.



(a) Input Topology (b) BSG Embedding (c) Cell Sizing (d) Detail Routing

Fig. 5. An illustration of our idea.

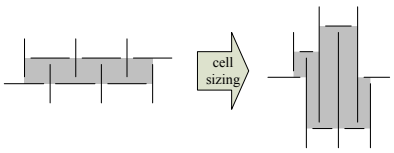
topology. Which embedding we choose has little impact on the final routing result as long as we follow the guidelines below:

1) BSG is a structure that represents *left-right*, *above-below* relations between objects. Therefore, the topological relation between objects should be kept in the embedding. If one component is located to the left of the other, then it should still be on the left in the embedding. Similarly, if a pin is above another pin, its cell should also be above the other's cell.

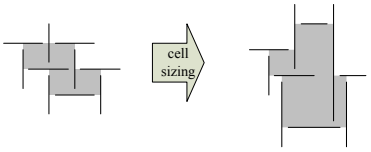
2) The number of BSG cells we use does NOT depend on the input routing grid size because we can later size the cells to match the routing domain. Instead, the BSG size depends on how flexible we want the routing to be. If we want the routing to be more flexible, we should use more cells for the embedding (that is, add more BSG cells between the two components). However, we should not use an excessively large BSG because this will enlarge the scale of later cell sizing problem. Our experience is that a 200×200 BSG would provide enough flexibility for a routing problem with around 100 nets. This guideline also explains why our router is insensitive to the input routing grid size.

3) We need to allow at least one empty cell between two nets. If two adjacent cells are occupied by two nets, moving the wall between them affects the areas of both nets. This means we lose the flexibility of controlling their areas independently.

4) A component should be mapped into multiple cells forming a rectangular area. The number of cells it occupies depends NOT on its physical size but on the number of pins around its boundary. A component with more pins requires more BSG cells because each pin takes up an individual cell on the boundary. Notice that we also need to plan at least one empty cell between adjacent pins according to guideline 3.



(a) One BSG embedding and its cell sizing result



(b) Another BSG embedding and its cell sizing result

Fig. 6. Different embeddings give the same area assignment after cell sizing.

As we claimed before, the selection of embeddings has little impact on the final result as long as we have enough cells for the flexibility. This is because even with different BSG embeddings, we are able to obtain the same area assignment through careful cell sizing. As can be seen from Fig. 6, even though the initial embeddings (left side) are different, the shape and size of the area assigned to the net (gray area on the right) are the same after cell sizing[1]. Therefore, the key step of our router is cell sizing.

C. Cell Sizing

After we embed the topology onto a BSG, we size the BSG cells for area assignment. The cell sizing problem is essentially to determine the location of the walls. We can formulate it as a mathematical programming problem. For every V-wall v, we use a variable x_v to represent its x coordinate. (Notice that the y coordinates of the two ends of this wall are not determined by the V-wall itself. They are determined by the positions of the two H-walls at its two ends.) For every H-wall h, we use a variable y_h to represent its y coordinate. For every cell i, we have four walls surrounding it. We name the variables representing its left, right, bottom and top walls as $x_{i,l}$, $x_{i,r}$, $y_{i,b}$ and $y_{i,t}$ respectively (see Fig. 7). Notice that these names are only aliases of actual variables. Different names may refer to the same variable. For example, $x_{i,r}$, $x_{j,l}$, $x_{p,r}$ and $x_{q,l}$ all refer to the same variable x_v of wall v in Fig. 7.

To guarantee that the final routing is legal and the length constraints are satisfied, we need to enforce the following constraints on the variables:

1) Basic Constraints: Every cell of the BSG must have a non-negative area. That is, the top wall of a cell cannot be placed below the bottom wall of the cell and the right wall of

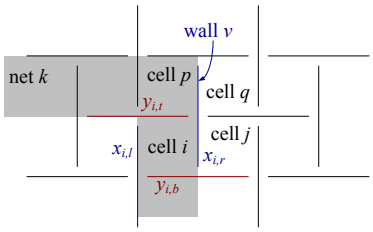


Fig. 7. Every wall in the BSG is assigned a variable to represent its position.

[1]Notice that the embedding in Fig. 6 (a) requires one more cell than Fig. 6 (b) to achieve the same area assignment. That is why we need to use enough cells for the flexibility.

a cell cannot lie to the left of the left wall of the same cell. Therefore, for each cell i we have the following constraints (assuming a coordinate system with x-axis pointing right and y-axis pointing up):

$$x_{i,r} - x_{i,l} \geq 0 \quad (1)$$

$$y_{i,t} - y_{i,b} \geq 0 \quad (2)$$

Furthermore, if a cell is occupied by a net, we must make sure that the size of the cell allows one wire to pass. Since we regard the wire as a fat wire with width ε, this means that both the width and the height of the cell should be at least ε:

$$x_{i,r} - x_{i,l} \geq \varepsilon \quad (3)$$

$$y_{i,t} - y_{i,b} \geq \varepsilon \quad (4)$$

So the basic constraints for a cell are either (1), (2) or (3), (4) depending on whether the cell is empty or occupied.

2) Location constraints: The locations of the components and the pins are given in the input and the cell sizing should conform to them. Therefore, we have to fix the walls on the four boundaries of a component. For a component m with its left-bottom corner located at (L_m, B_m) and right-top corner located at (R_m, T_m), we have the following constraints:

$$x_{i,l} = L_m \quad \text{for cells on the left boundary of } m \quad (5)$$

$$x_{i,r} = R_m \quad \text{for cells on the right boundary of } m \quad (6)$$

$$y_{i,b} = B_m \quad \text{for cells on the bottom boundary of } m \quad (7)$$

$$y_{i,t} = T_m \quad \text{for cells on the top boundary of } m \quad (8)$$

If a pin p is located on the top boundary of a component (see Fig. 8), then the cell it occupies is also on the top boundary of the component in the BSG embedding. Therefore, the y-coordinate of this pin is already fixed by constraint (8). We only need to introduce the following constraints to fix it at its x-coordinate X_p (suppose its BSG cell is i):

$$x_{i,l} = X_p - \varepsilon/2 \quad (9)$$

$$x_{i,r} = X_p + \varepsilon/2 \quad (10)$$

Again, since we view wires as fat wires, we leave a $\varepsilon/2$ margin on both sides[2]. The constraints for pins located on the other three boundaries can be derived in the same way.

At last, we need to fix the walls on the boundaries of the entire BSG at the boundaries of the input routing domain. This can be done by using constraints (5), (6), (7) and (8), imaging that the entire routing domain is a big component.

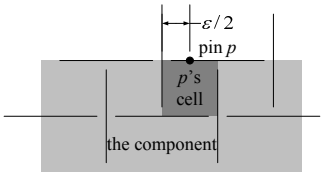


Fig. 8. Constraints on the pin location.

[2]Notice that according to guideline 4, we should plan at least one empty cell between two adjacent pins. So if two adjacent pins have a distance larger than ε, the emtpy cell will provide the extra space and (9) and (10) will be satisfied for both pins.

3) Length Constraints: We also need to put constraints on the sizes of the cells to satisfy the length bounds. For a net k, we have constraint (11) for its upper length bound u_k and constraint (12) for its lower length bound l_k (in both constraints, i denotes all the BSG cells occupied by net k):

$$\sum_i ((x_{i,r} - x_{i,l}) + (y_{i,t} - y_{i,b}) - \varepsilon) \leq u_k \quad (11)$$

$$\frac{\sum_i ((x_{i,r} - x_{i,l})(y_{i,t} - y_{i,b}))}{\varepsilon} \geq l_k(1+\delta) \quad (12)$$

The discussion on how we obtain these constraints and what δ stands for involves the explanation of how we route inside each cell so we postpone it until section III-E where we explain how we perform detail routing.

D. Solve The Problem

By putting all these constraints together, we have a feasibility problem: find a solution satisfying all the constraints from (1) to (12). Although the problem contains only linear and quadratic constraints, it is not a convex programming problem because constraint (12) is not convex. Discussions with experts in mathematical optimization reveal that it very difficult to turn this problem into a convex programming problem[3]. Therefore, we use another approach to tackle this problem.

An observation is that if we fix all the V-walls, i.e., fix all x as constants, then (12) becomes a linear constraint for y. Therefore, we can first fix the V-walls (x as constants) and solve for the H-walls (y as variables) under all the constraints involving y (such as (2), (4), (7), (8), (11) and (12)). This gives us a linear programming (LP) problem which we call *H-problem* because it determines the location of H-walls. Then we do the opposite, we use the solution of the H-problem we just solved as constants and solve for the location of V-walls under the constraints involving x (such as (1), (3), (5), (6), (9), (10), (11) and (12)). This gives us another LP problem which we call *V-problem*. We alternately solve the H-problems and V-problems and finally we will reach a feasible solution.

However, two questions must be answered before we can make this method work:

1) How do we determine the initial constants in (11) and (12)?: For the first problem in the sequence, we don't have any previous problem, which means we don't actually have the x or y constants in (11) and (12). In this case, we assume that all the walls are evenly distributed within the rectangular domain. This assignment may not conform to the component boundaries or pin locations. This means the location constraints may not be satisfied. However, it does not matter. An iteration of H-problem and an iteration of V-problem will force the solution to satisfy the location constraints.

[3]One idea is to let $w_i = x_{i,r} - x_{i,l}$ and $h_i = y_{i,t} - y_{i,b}$ and then (12) becomes $\sum_i w_i h_i \geq l_k(1+\delta)\varepsilon$ and all the other constraints remain linear. Even though non-convex itself, this new constraint is a posynomial so we can use the trick for geometric programming to turn it into a convex constraint. Unfortunately, this trick makes constraints (5) to (10) non-convex.

2) What if we get an infeasible LP problem during the iterations?: If the input is solvable and the BSG embedding follows our guidelines, then there should always be a feasible solution that satisfies the basic constraints and the location constraints ((1) to (10)). If an LP problem is not feasible, the reason must be that the initial constants or constants obtained from previous iteration is not proper for length constraints (11) and (12). We avoid this infeasibility by introducing a slack variable $s \geq 0$ into the length constraints:

$$\sum_i ((x_{i,r} - x_{i,l}) + (y_{i,t} - y_{i,b}) - \varepsilon) - s \leq u_k \tag{13}$$

$$\frac{\sum_i ((x_{i,r} - x_{i,l})(y_{i,t} - y_{i,b}))}{\varepsilon} + s \geq l_k(1 + \delta) \tag{14}$$

When $s \to +\infty$, both constraints are satisfied no matter what x and y are (notice that x and y are all bounded by the routing domain). When $s = 0$, the two constraints are the same as (11) and (12). Therefore, instead of find a solution to the original feasibility problem, we try to minimize s under the new constraints (13) and (14). If we can minimize s to 0, then we have a feasible solution to our original problem.

Now we reformulate our H-problem and V-problem:

H-problem (y as variables)
minimize: s
satisfying: basic constraints involving y
location constraints involving y
length constraints (13) and (14)

V-problem (x as variables)
minimize: s
satisfying: basic constraints involving x
location constraints involving x
length constraints (13) and (14)

We iteratively solve the H-problem and the V-problem until $s = 0$ or s cannot be further reduced (but still > 0). If the later happens, it means our cell sizing problem does not have a feasible solution. We should relax our length bounds.

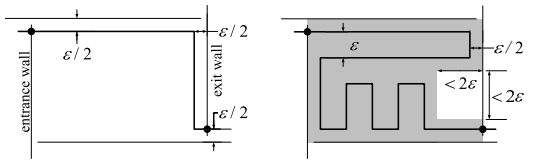
Theorem 1: Our iterative optimization method will terminate in finite iterations.

Proof: Everytime we solve an H-problem or a V-problem, s can only decrease or remain the same. Since s has a lower bound 0, the optimization must end in finite iterations. ■

Experiments show that we usually reach $s = 0$ by solving only $2 \sim 3$ LP problems.

E. Detail Routing

Before we explain how to route inside a cell, we first introduce some definitions. In the BSG embedding, a net must enter a cell by going through a wall and exit the cell by going through another wall. We call the first wall *entrance wall* of the cell and the second wall *exit wall* of the cell. For example, in Fig. 7, the entrance wall of cell i is its top wall and the exit wall is its bottom wall assuming net k comes from the left and turns to the bottom. If the entrance wall and exit wall of a cell are of the same type (both H-walls or both V-walls), we call the cell *straight cell*. Otherwise, we call it *corner cell*. In Fig. 7, cell i is a straight cell and cell p is a corner cell.



(a) Minimum routing length (b) Maximum routing length

Fig. 9. The minimum and maximum routing length inside a BSG cell.

For any cell occupied by a net, we always route from a point on its entrance wall, which we call *entrance point*, to a point on its exit wall, which we call *exit point*. The entrance point is located $\varepsilon/2$ away from the corner where the entrance wall meets another wall (see the dots in Fig. 9). Similarly, the exit point is located $\varepsilon/2$ away from the corner where the exit wall meets another wall. Therefore, the exit point of a cell always coincides with the entrance point of the succeeding cell in a net. This guarantees that the detail routing of a net is continuous between cells. The $\varepsilon/2$ spacing guarantees that wires in different cells do not violate the separation rule.

For a straight cell i, the minimum routing length from the entrance point to the exit point is their Manhattan distance $w_i + h_i - \varepsilon$ (w_i and h_i denote the width and height of the cell). Now it becomes clear why we have constraint (11). The shortest length we can route for a net is the sum of $w_i + h_i - \varepsilon$ over all the cells occupied by the net. This length must be shorter than the upper length bound of the net.

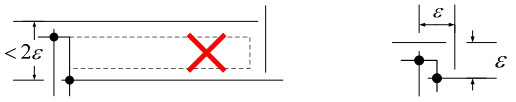
To obtain longer length, we need to detour inside the cell. The following theorem shows that we can efficiently use the cell area for length extension (see Fig. 9 (b)):

Theorem 2: For a straight cell with width $w \geq \varepsilon$ and height $h \geq \varepsilon$, there always exists a valid route ("valid" means the separation rule is satisfied[4]) from the entrance point to the exit point with its length $\geq \frac{wh}{\varepsilon} - 4\varepsilon$.

Proof: The proof is by construction. We first detour horizontally as much as possible and then detour vertically as much as possible. This leaves us with a white space of at most $2\varepsilon \times 2\varepsilon$. So the routing length is at least $\frac{wh}{\varepsilon} - 4\varepsilon$. ■

Even though very small compared to the length bounds, the $4\varepsilon^2$ white space still means that we cannot always fully turn the area into routing length. Therefore, we need to assign a little bit more area to a net than its lower length bound requires to compensate this loss. However, since $4\varepsilon^2$ is an upper bound of the white space, using it to estimate the area compensation for every cell is too pessimistic. In many cells we are not so unlucky to hit this upper bound. Therefore, we introduce a parameter δ in (12). It enlarges the area requirement in the cell sizing problem so that we are able to achieve the required lower length bound even with some waste of the area. Our experience is that $\delta = 0.05$ would be enough in practice.

[4]Some PCB manufacturers also assume a minimum feature length λ, i.e., every wire segment must be longer than λ. In this case, the maximum routing length in a cell is at least $\frac{wh}{\varepsilon} - (6\varepsilon + 2\lambda)$. Detailed proof for this lower bound is omitted. The $6\varepsilon + 2\lambda$ loss of routing length is still very small considering that both ε and λ are very small in practice.



(a) Skinny corner cell (b) Corner cell with $\varepsilon \times \varepsilon$ size

Fig. 10. A skinny corner cell does not allow wire extension.

Unfortunately, Theorem 2 is not necessarily true for corner cells because we may have a very skinny cell with its height less than 2ε but its width very large (see Fig. 10 (a)). In this case, even though the area of the cell is large, we cannot utilize it for length extension. To avoid such skinny corner cells, we force the cell to be $\varepsilon \times \varepsilon$ large. That is, if a cell i is a corner cell, we use the following constraints instead of (3) and (4):

$$x_{i,r} - x_{i,l} = \varepsilon \quad (15)$$

$$y_{i,t} - y_{i,b} = \varepsilon \quad (16)$$

By doing this, we are able to guarantee that all the extra area assigned to a net is assigned to straight cells and can be effectively used for wire extension. Of course, we will lose some flexibility because corner cells can no longer be used for wire extension. However, since corner cells take up only a very small portion of the occupied cells, this brings us negligible influence on the capability of our router.

Notice that the sizes and locations of the cells are all real numbers. So the completed detail routing does not conform to any routing grid. That is why we call our router gridless. Nevertheless, the separation rule ε is always satisfied.

IV. Experimental Results

In this section, we compare the performance of our router with that of [3]'s router. We choose the router in [3] instead of those in [1] and [2] because it imposes less restrictions on the routing topology. The other two routers can be applied to very few industrial data. Our router is implemented in C++ and the linear programming (LP) problems are solved by the open source linear solver *lp_solve* [9]. As for embedding the topology onto the BSG, we use a simple heuristic since our data have only two components. We first route the pins of both components to a channel between them and then use river routing to route inside the channel. Other heuristics such as maze routing can be employed for more complicated cases, e.g., when more than two components are involved. As mentioned before, using different heuristics to generate the embedding has insignificant influence on the final routing quality as long as our guidelines are followed. Experiments are performed on a computer with two 2.8GHz Intel Xeon processors and 4GB memory. The platform is Redhat Enterprise Linux 4.

We use 7 data to test our router (see Table I). The *monotonic* data set allows monotonic (left to right) routing topology so the router in [3] can be applied. The topologies in the *general* data set are general and no previous routers can be applied. *monotonic_1*, *monotonic_2* and *general_1* are original industrial data and *monotonic_3*, *monotonic_4*, *general_2* and *general_3* are derived from industrial data.

The experimental results of both routers are reported in Table I. The "BSG size" column gives the size of the embedded BSG. The two columns following it give the size (the number of variables and the number of constraints) of the LP problems we formulate for cell sizing. "#.it" gives the number of LP problems we solve before s converges to 0. The runtime of our router includes the runtime of the LP solver. Actually, the majority of the runtime is spent on solving the LP problems.

Several observations can be made from the experiments:

1) The BSG size we use is about 10 times smaller than the routing grid size of [3] in both width and height. Moreover, our BSG size is not sensitive to the routing grid size of the problem. For example, the routing grid size of *monotonic_2* is much larger than that of *monotonic_3*. However, our BSG size remains similar.

2) Our router can handle industrial designs that cannot be handled by previous routers.

3) For the data that can be handled by [3], our router runs much faster. The runtime difference can be as huge as 1000x (88 seconds vs. 99491 seconds for *monotonic_4*).

4) Only 2∼3 LP problems need to be solved before it converges to a feasible cell sizing solution. This means our approach to solve the cell sizing problem is efficient.

5) The routing result of *general_3* is shown in Fig. 11. It can be seen that the routing is very dense, indicating that our router can effectively use the routing area for length-matching.

V. Conclusion Remarks

In this paper, we introduced a length-matching router that handles general topology. It is the first time that the length-matching routing problem is solved without any restriction on the routing topology. With the help of the BSG structure, we are able to convert the length-matching problem into a cell sizing problem and solve the problem by solving a sequence of linear programming problems. Due to its gridless feature, our router is insensitive to the routing grid size of the input, making it very fast for large PCB designs.

Here we list several possible extensions of our router:

1) Although we assume the components and the input routing domain to be rectangular, it is easy to tweak the router to handle rectilinear components and routing domain.

2) We only consider the separation rule in this paper. However, it is very easy to take wire width into consideration. It is also possible to consider nets with different wire widths and wire separations.

3) Although our experiments involve only two components, the router itself is capable of handling multiple components. The only difference is that we need to use a more general router, e.g., maze router, to embed the topology onto the BSG.

4) In our problem setting, we assume that the length bounds are given. Sometimes designers want to route all the nets with the same length and minimize this length. By minor modifications, our router can also handle this problem.

Acknowledgment

The authors would like to thank Dr. Hiroshi Murata of Gem Design Technology Inc. and Prof. Angelia Nedich of University of Illinois at Urbana-Champaign for the helpful discussion on the convex programming formulation of our problem.

TABLE I

EXPERIMENTAL RESULTS.

		our BSG-route					router in [3]	
		BSG size	H-problem	V-problem	#it.	runtime	routing grid size	runtime
data	#nets	$w \times h$	#var. /#constr.	#var. /#constr.		(sec)	$w \times h$	(sec)
monotonic_1	84	87×175	7829 / 14543	7829 / 14629	2	86	1181×1237	137
monotonic_2	44	125×95	6093 / 10769	6093 / 10738	2	73	2252×2383	NEM[a]
monotonic_3	83	67×173	6000 / 10898	6000 / 11002	2	56	1012×899	13859
monotonic_4	45	119×112	6826 / 12057	6886 / 12050	3	88	2252×680	99491
general_1	36	105×86	4648 / 8730	4701 / 8664	3	64	N/A	
general_2	28	62×91	2973 / 5464	2927 / 5456	3	21	N/A	
general_3	36	109×86	4822 / 9078	4877 / 9008	3	260	N/A	

[a]NEM: Not Enough Memory. The required memory to run this data exceeds the memeory of the computer (4GB).

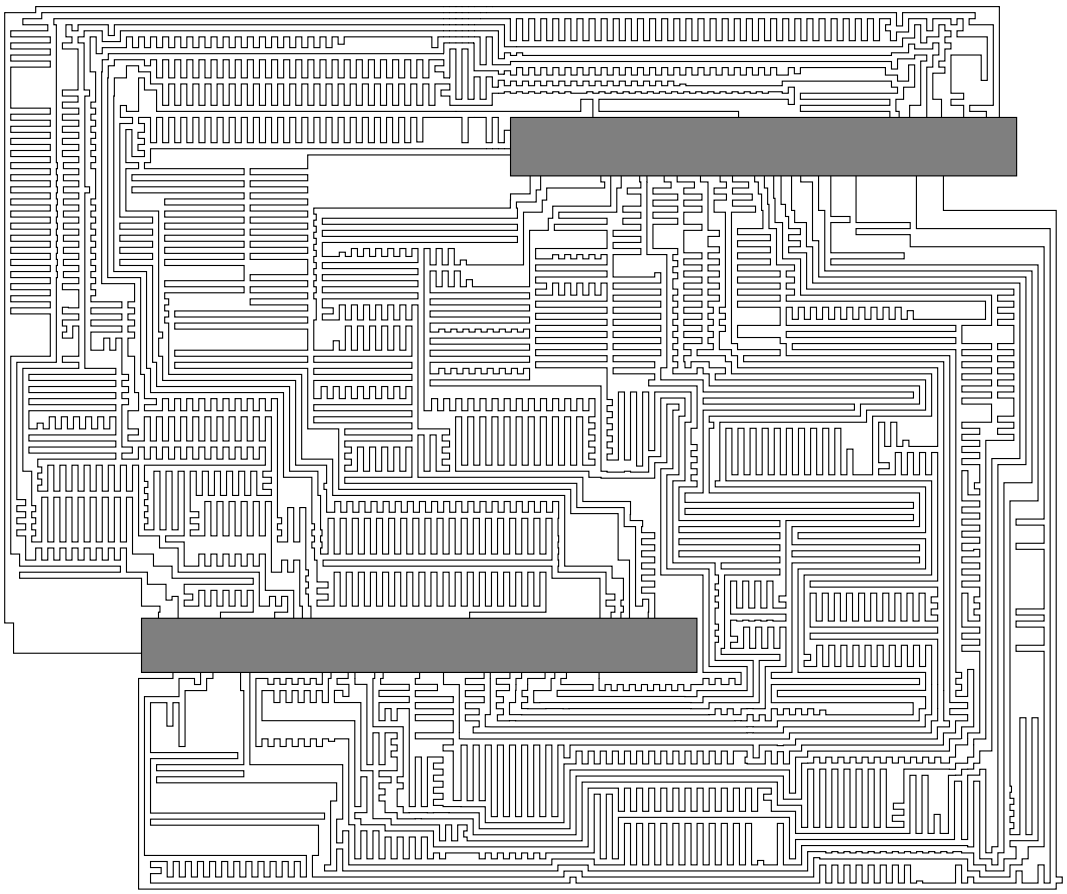


Fig. 11. Our routing result of *general_3*.

REFERENCES

[1] M. M. Ozdal and M. D. F. Wong, "Algorithmic study of single-layer bus routing for high-speed boards," *IEEE Trans. Computer-Aided Design*, vol. 25, no. 3, pp. 490–503, Mar. 2006.

[2] Y. Kubo, H. Miyashita, Y. Kajitani, and K. Takeishi, "Equidistance routing in high-speed VLSI layout design," *Integration, the VLSI Journal*, vol. 38, no. 3, pp. 439–449, Jan. 2005.

[3] M. M. Ozdal and M. D. F. Wong, "A length-matching routing algorithm for high-performance printed circuit boards," *IEEE Trans. Computer-Aided Design*, vol. 25, no. 12, Dec. 2006.

[4] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module packing based on the BSG-structure and IC layout applications," *IEEE Trans. Computer-Aided Design*, vol. 17, no. 6, June 1998.

[5] D. Wiens, "Printed circuit board routing at the threshold," in *White Paper*. Mentor Graphics, 2000. [Online]. Available: http://www.mentor.com/products/pcb/expedition/techpubs/mentorpaper_626

[6] L. W. Ritchey and J. Zasio, *Right the First Time, A Practical Handbook on High Speed PCB and System Design*, K. J. Knack, Ed. Speeding Edge, 2003.

[7] L. W. Ritchey, "Busses: What are they and how do they work?" in *Printed Circuit Design Magazine*, Dec. 2000. [Online]. Available: http://www.speedingedge.com/PDF-Files/busses.pdf

[8] T. Yan and M. D. F. Wong, "Untangling twisted nets for bus routing," in *Proc. IEEE/ACM Intl. Conf. on Computer-Aided Design*, 2007, pp. 396–400.

[9] lp_solve: an open source linear programming solver. [Online]. Available: http://sourceforge.net/projects/lpsolve