A Length-Matching Routing Algorithm for High-Performance Printed Circuit Boards

Muhammet Mustafa Ozdal and Martin D. F. Wong, *Fellow, IEEE*

***Abstract*—As the clock frequencies used in industrial applications increase, the timing requirements imposed on routing problems become tighter. Therefore, it becomes important to route the nets within tight minimum and maximum length bounds. Although the problem of routing nets to satisfy maximum length constraints is a well-studied problem, there exists no sophisticated algorithm in literature that ensures that minimum length constraints are also satisfied. In this paper, the authors propose a novel algorithm that effectively incorporates the min–max length constraints into the routing problem. The approach is to use a Lagrangian-relaxation (LR) framework to allocate extra routing resources around nets simultaneously during routing them. The authors also propose a graph model that ensures that all the allocated routing resources can be used effectively for extending lengths. Their routing algorithm automatically prioritizes resource allocation for shorter nets and length minimization for longer nets so that all nets can satisfy their min–max length constraints. This paper demonstrates that this algorithm is effective even in the cases where length constraints are tight, and the spacing between adjacent nets is small.**

***Index Terms*—Algorithms, Lagrangian relaxation, printed circuit board (PCB), routing, very large scale integration (VLSI).**

I. Introduction

ROUTING NETS within minimum and maximum length bounds is an important requirement for high-speed very large scale integrated layouts. There have been several algorithms proposed for the objective of minimizing path lengths, or satisfying prespecified maximum length constraints, especially in the context of timing-driven routing [3], [5] [6], [8], [18] [19], [21]. However, the problem of routing nets with lower bound constraints has not been studied explicitly in the literature. The main reason is that these bounds were loose most of the time, and nonsophisticated strategies (such as greedy length extension in postprocessing) were sufficient for most applications. However, as circuits start to use clock frequencies in the order of gigahertz in the current technology, the timing constraints become extremely tight, and more aggressive methods for achieving length bounds are needed in the industrial applications.

Manuscript received March 24, 2006. This work was supported in part by the National Science Foundation under Grants CCR-0306244 and in part by an IBM Faculty Partnership Award. This paper was recommended by Associate Editor S. Sapatnekar.

M. M. Ozdal was with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA. He is now with the Design and Technology Solutions Department, Intel Corporation, Hillsboro, OR 97124 USA (e-mail: mustafa.ozdal@intel.com).

M. D. F. Wong is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, Urbana, IL 61801 USA (e-mail: mdfwong@uiuc.edu).

Digital Object Identifier 10.1109/TCAD.2006.882584

Timing constraints are commonly imposed on printed circuit board bus structures, where data is clocked into registers or other circuits. For example, in the case of a 64-bit data bus, each bit travels over a different wire, and all 64 bits must arrive at the destination pins approximately at the same time. To achieve this, all the wires constituting this bus need to have approximately the same lengths. The precision with which matching must be done is directly related to the clock frequency [23]. As the clock frequency increases, the skew requirements on the propagation delays become stricter and, hence, a higher degree of length matching is required.

A typical approach used for this problem is to route the nets using a conventional routing algorithm to satisfy maximum length constraints, and then perform snaking to extend the routes of the short nets during postprocessing. The main disadvantage of such an approach is that after all the nets have already been routed, the available routing space around short nets might be limited in dense designs. Therefore, it is likely that some nets cannot be extended to satisfy minimum length constraints due to lack of routing space.

In this paper, we propose a novel algorithm that incorporates the objective of satisfying min–max length constraints effectively into the original routing problem. For the ease of presentation, we will first focus on the length-matching problem, and then, we will extend our models for the general case where individual nets might have different lower and upper bound constraints. For this, we start with redefining the routing problem as follows: Find valid routes for all nets such that: 1) The length of the longest route is kept small and 2) the shorter routes have available routing space around themselves such that it is possible to match all lengths by snaking at the end. We propose effective algorithms in this paper to handle both objectives simultaneously during routing.

As a motivating example, consider the circuit given in Fig. 1(a). Here, there are three nets that need to be routed with equal lengths, and Fig. 1(a) illustrates a typical routing solution[1] given by a conventional router. Here, all nets were routed first, and then snaking was performed at the end for length matching. Observe that the top net turned out to be the longest one, with a path length[2] of 17. Therefore, the length of the bottom net was extended by six through snaking. However, snaking was not possible for the middle net, because all routing

[1]The underlying grid structure is also shown in Fig. 1(a). Throughout this paper, we assume that routing edges go center-to-center of each grid cell, as illustrated in this Fig. 1(a). Note that each grid cell is regarded as a routing resource.

[2]All the path lengths given in this paper are in terms of number of grid cells spanned.

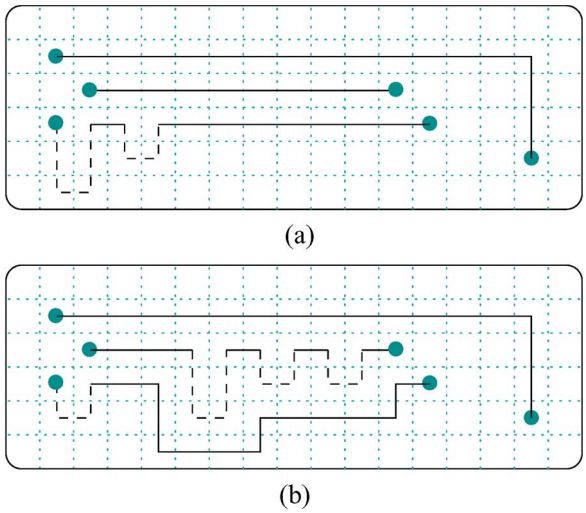


Fig. 1. Length matching based on (a) greedy snaking in postprocessing and (b) resource allocation during routing. Dashed lines indicate snaking performed. Observe that the length of the middle route could not be extended in (a).

resources around its route were used during routing. Therefore, length matching fails in this example.

Fig. 1(b) shows the solution given by the router we propose in this paper. Observe that the lengths of these three nets are matched exactly through snaking. Here, our approach is to simultaneously route each net and allocate extra routing resources (i.e., grid cells) for them. After that, these extra resources are used for snaking. There are a couple of points worth mentioning here. First of all, the number of extra grid cells allocated for a net depends on the length of its route (i.e., more grid cells are allocated for shorter nets, and vice versa). Here, it is likely that the actual routes of the nets will be affected because of this resource allocation. In this example, the bottom net is detoured so that there are enough resources allocated for the middle net. An important point here is that it is not the top net that is detoured for this purpose, because detouring the top net would increase the length of the longest route. In fact, we can say that the two objectives for length matching mentioned above are achieved simultaneously in this example.

As will be discussed in detail later, we perform multiple iterations, each of which is guided by the global objective of length matching. In one iteration, we route nets and allocate resources so as to minimize an objective Lagrangian function, which captures both minimum and maximum length constraints for all nets. Our low-level routing algorithm is based on Pathfinder negotiated congestion algorithm [1], [2], [9]. However, we propose a methodology to handle resource allocation simultaneously during path calculations, as opposed to the greedy algorithm above, which considers minimum constraints only in postprocessing. In our approach, shorter nets automatically prefer the paths where they can allocate extra resources around.

The remainder of this paper is organized as follows. In Section II, we summarize the relevant work in the literature, and discuss why they are not applicable for this problem. Then, we propose an LR-based algorithm that facilitates allocating extra resources during routing in Section III. After that, we propose a graph model in Section IV to perform resource allocation in accordance with snaking. Specifically, this model makes sure that if the number of extra grid cells allocated for net i is S_i, it is possible to extend length of net i by an amount equal to S_i through snaking. In other words, resource allocation is done in such a way that every allocated grid cell can be used for snaking later. After that, we outline the low-level routing algorithm we use in Section V. Then, in Section VI, we briefly explain how to extend this method for more general problems. Note that although we propose a systematic approach based on LR for this problem, the solution found is not guaranteed to be optimal, because of the nonconvex nature of the problem. Furthermore, it is not guaranteed that a feasible solution will be found, even if one exists. However, we demonstrate the effectiveness of our heuristics through experiments in Section VII.

II. Related Work

There have been several routing algorithms proposed in the literature for the objective of satisfying maximum length constraints [3], [5], [6], [8], [18], [19], [21]. Typically, these algorithms try to keep the lengths of critical nets shorter, but they do not consider explicit minimum length constraints.

A related problem in the literature is the zero/bounded-skew clock tree routing problem [17]. Here, the objective is to construct a clock tree such that the arrival times for all source-sink pairs are (almost) equal. However, our bus routing problem is different in the sense that each terminal pair belongs to a different net, and no overlaps are allowed between different pairs. On the other hand, in the clock tree routing problem, there is a single net (with multiple terminals), and the objective is to find a routing tree, instead of independent pairwise connections. Several algorithms have been proposed in the literature for this problem [4], [16], [17], [24]. However, they are based on tree construction methods most of the time, and they are not applicable to the case where each pairwise connection must be routed independent of each other.

If the length-matching problem consists of only two nets, it is possible to use a wave-expansion method to find a feasible solution [20]. Let us denote the source-sink pair of the two nets as (s_1, t_1) and (s_2, t_2). Here, waves are expanded originating from these four terminals, and their intersections are checked repeatedly. Namely, whenever waves from s_1 and t_1 meet, the length of the corresponding path is compared with every path between s_2 and t_2. This process continues until a match is found between the lengths of two paths. Note that this approach does not explicitly avoid short circuits between the two nets, so special care must be taken, such as restricting propagating waves to separate portions of the layout [20]. However, we cannot use this technique in our bus-routing problem, since the number of nets is typically much larger than two. Here, the main problem is that conflicts between different nets cannot be detected during simultaneous wave expansions, and it would not be practical to limit the waves of all nets to separate regions when there are multiple nets.

On the other hand, some traditional routing tools allow users to specify length-matching constraints. However, as the clock frequencies increase, the constraints for typical high-end circuits become extremely tight, and these tools fail to find a feasible routing solution for many high-end industrial designs.

A commonly used practical approach here is to route all nets first and then to perform length extension in postprocessing. The disadvantage of such an approach is that after all nets have already been routed, the available routing space around short nets might be limited in dense designs. Therefore, it is likely that some nets cannot be extended to satisfy minimum length constraints due to lack of space. In Section VII, we will present an experimental comparison of this practical approach with our framework.

III. Routing-Resource Allocation

A. Problem Formulation

The original length-matching problem can be stated as follows. Given a circuit, and a set of nets $\mathcal{N}$, find a congestion-free routing solution for each net in $\mathcal{N}$ such that the maximum path length is minimized, and the difference between the minimum and maximum path lengths does not exceed the predefined tolerance value Δ. Here, the input circuit is assumed to be modeled as a uniform $n \times m$ grid structure, where each grid cell is marked as either a routing resource, or a blockage. Each net in $\mathcal{N}$ is assumed to have two fixed terminals on the grid structure. A routing solution for a set of nets $\mathcal{S}$ is defined to be congestion free, if and only if no routing resource on the grid is used by more than one net in $\mathcal{S}$.

To solve the length-matching problem, we introduce two main objectives for the router: 1) to keep the path lengths of longer nets small and 2) to allocate extra routing resources around shorter nets such that their lengths can be extended through snaking. Intuitively, we want to minimize the expression: $\sum_{i\in\mathcal{N}}(\alpha_i L_i - \beta_i S_i)$, where $\mathcal{N}$ is the set of all nets; L_i is the length of net i's route; S_i is the total number of extra grid cells allocated for net i; and α_i and β_i are weighting terms. One can argue that α_i for long nets should be large, giving priority to path length minimization. On the other hand, β_i for short nets should be large, giving more priority to resource allocation. In this section, our focus will be on how to set and update these parameters dynamically such that the two main objectives are achieved simultaneously.

For simplicity of the presentation, we assume that routing will take place on one layer only. Furthermore, our focus will be to route only one bus, i.e., all the given nets need to be routed with the same length. However, it is straightforward to extend our models and algorithms to a multilayer multibus routing problem, or to the general problem where each net has a different length constraint, as will be discussed in Section VI. In addition, we introduce some restrictions for the resulting routing solutions. We assume that there is a preferred direction for each net, and all the snaking will be performed perpendicular to this direction. Furthermore, the resulting routes will not have any detour toward opposite of the preferred direction. For example, if the preferred direction is RIGHT, then snaking will be performed UP and DOWN (as in Fig. 1); detouring toward LEFT will not be allowed. These restrictions are necessary for the models we propose. However, we believe that they will not degrade the solution quality, because a typical routing solution given by a conventional router would also satisfy these conditions. For simplicity of presentation, we will first assume that there is a global preferred direction for all nets. However, it is possible to generalize our models to the case where each net has an individual preferred direction, as will be discussed in Section VI.

B. LR-Based Resource Allocation

LR is a general technique for solving optimization problems with difficult constraints. The main idea is to replace each complicating constraint with a penalty term in the objective function. Specifically, each penalty term is multiplied by a constant called Lagrangian multiplier (LM), and added to the objective function. The Lagrangian problem is now the optimization of the new objective function, where difficult constraints have been relaxed and incorporated into the new objective function. If the optimization is a minimization problem, then the solution of the Lagrangian problem is guaranteed to be a lower bound for the original optimization. In fact, LR is a two-level approach: In the low-level Lagrangian problem, it is solved for fixed LM values. In the high level, LM values are updated iteratively such that the optimal value obtained in the low level is as close to the real optimal value as possible. Typically, a subgradient method is used to update LM values in the high level. Intuitively, the LM values corresponding to the constraints that are not satisfied in the current iteration are increased (hence, the weights of these constraints in the low-level objective function are increased) and vice versa. The iterations continue until a convergence criterion is satisfied. Further details can be found in various survey or tutorial papers about LR [10]–[12].

The length-matching problem can be formulated as a constrained optimization problem. Assume that we somehow determine[3] a target length T and our purpose is to route each net i in set $\mathcal{N}$ with a path length in the range $T - \Delta$ and T.

Based on the resource allocation idea we have discussed before, it is possible to give the following formulation:

$$\begin{aligned} \text{minimize} \quad & \sum_{i\in\mathcal{N}} L_i \\ \text{subject to :} \quad & \forall i, \quad L_i \le T \\ & \forall i, \quad L_i + S_i \ge T - \Delta. \end{aligned} \tag{1}$$

Again, L_i denotes the length of net i's route, and S_i denotes the number of extra grid cells allocated for net i. Suppose for now that it is possible to extend the length of net i by an amount up to S_i using snaking (in Section IV, we will propose a model that will facilitate this). Observe that the first constraint above simply states that the total length should not exceed the target length. On the other hand, with the second constraint, we make sure that shorter nets allocate enough routing resources for snaking.

If we apply LR on this formulation, our objective becomes minimization of

$$\sum_{i\in\mathcal{N}} L_i + \sum_{i\in\mathcal{N}} \lambda_{iL}(L_i - T) - \sum_{i\in\mathcal{N}} \lambda_{iS}(L_i + S_i - T + \Delta). \tag{2}$$

[3]Initially, T can be set based on the maximum Manhattan distance of the terminal positions of the input nets. If no routing solution is found with target length T, it can be increased gradually throughout the execution.

```
ROUTE-AND-LENGTH-MATCH (Inputs: N, T)
Initialize λ_iL, λ_iS to zero for each i ∈ N
while termination condition not occurred do
    route all nets for fixed λ_iL, λ_iS values (Section V)
    for each i ∈ N do
        check constraints for current route
        update λ_iL and λ_iS values accordingly
perform snaking using the extra grid cells allocated
```

Fig. 2. High-level algorithm description.

Here, each λ_{iL} and λ_{iS} are LM corresponding to length and resource constraints given in the original formulation in (1). Intuitively, we would want longer nets to have larger λ_{iL} values (so that length minimization is prioritized for them), and shorter nets to have larger λ_{iS} values (so that resource allocation is prioritized for them).

The high-level algorithm we propose for length matching during routing is given in Fig. 2. For the following discussions in this section, assume that we have a subroutine for finding the routing solution that minimizes objective function in (2), for fixed λ_{iL}, λ_{iS} values. Observe in Fig. 2 that we iteratively call this subroutine, and update the LM until some convergence criterion is satisfied. We use an update scheme similar to subgradient method, but we have tailored it specifically for this problem. Given a routing solution in iteration k, and the current multiplier values λ_{iL}^k and λ_{iS}^k, the multipliers for iteration $k+1$ are calculated as follows:

$$\lambda_{iL}^{k+1} = \begin{cases} \max\left(0, \lambda_{iL}^k - t_k(T - L_i)^\gamma\right), & \text{if } L_i \le T \\ \lambda_{iL}^k + t_k v_{iL}(L_i - T)^\gamma, & \text{otherwise.} \end{cases} \tag{3}$$

$$\lambda_{iS}^{k+1} = \begin{cases} \max\big(0, \lambda_{iS}^k - t_k \\ \qquad \times (L_i + S_i - T + \Delta)^\gamma\big), & \text{if } L_i + S_i \ge T - \Delta \\ \lambda_{iS}^k + t_k v_{iS}(T - \Delta - L_i - S_i)^\gamma, & \text{otherwise.} \end{cases} \tag{4}$$

Note that t_k is the step size used in subgradient method, and it is updated in each iteration such that it slowly converges to zero. Specifically, we use the convergence condition given by Held *et al.* [15], which states that as $k \to \infty$; it should be the case that $t_k \to 0$ and $\sum_{i=1}^{k} t_i \to \infty$. The terms v_{iL} and v_{iS} denote the number of iterations the length constraint $(L_i \le T)$ and the resource constraint $(L_i + S_i \ge T - \Delta)$ for net i have been violated, respectively. If a constraint is not satisfied repeatedly for several iterations, then its multiplier is increased more rapidly. Finally, $\gamma \le 1$ is a constant we have introduced for this problem, and it is used to smoothen the effect of the amount of length or resource constraint violation, which can have large values. Our experiments have shown that setting it to a value as small as 0.1 gives decent results.

C. Handling Oscillation Problems

It is known that solution oscillation is a serious and inherent problem for LR-based methods [13], [22]. Note that even if the LM converge to their optimal values in the subgradient method, the solution to the original problem might oscillate between two extremes with a slight change of the multipliers. Guan *et al.* [14] identify one cause of such a behavior as the existence of homogeneous subproblems. A similar problem also exists in the formulation we have given in the previous section.

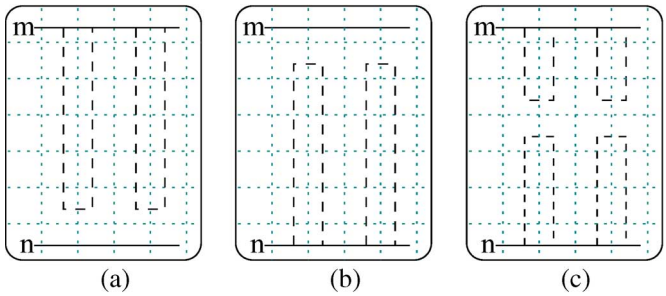


Fig. 3. Parallel routing segments of net m and net n, together with allocated routing resources (indicated by dashed lines). (a) Resource allocation if $\lambda_{mS} > \lambda_{nS}$. (b) Resource allocation if $\lambda_{nS} > \lambda_{mS}$. (c) Desirable resource allocation if λ_{nS} is only slightly larger than λ_{mS}.

Fig. 3 illustrates this problem with an example of two parallel routing segments. Assume that both net m and net n need to allocate extra routing resources (i.e., grid cells) around their routes to satisfy their resource constraints. Observe that to minimize objective function in (2), the intermediate grid cells should be allocated by net m, or net n, depending on the values of λ_{mS} and λ_{nS}. Specifically, if $\lambda_{mS} > \lambda_{nS}$, then the function in (2) will be minimized if S_m has its maximum value. Hence, all the intermediate grid cells will be allocated by net m [Fig. 3(a)]. On the other hand, if $\lambda_{nS} > \lambda_{mS}$, then S_n will be set to its maximum value as in Fig. 3(b) to minimize the objective function. Note that even if the difference between the two LM is infinitely small, the solution will be one of these extreme cases;[4] so the solution will always oscillate between these two. The desirable behavior would be as shown in Fig. 3(c) when λ_{mS} and λ_{nS} are close to each other.

A typical remedy for this kind of a problem is to use augmented LR [22], [25], where a penalty term is added to the Lagrangian function to avoid oscillations. Using a similar idea, we can modify the objective function in (2) such that our new objective becomes the minimization of

$$\sum_{i \in \mathcal{N}} (L_i + \lambda_{iL} L_i - \lambda_{iS} S_i) + \sum_{i \in \mathcal{N}} \sum_{e \in P_i} \epsilon (s^e)^2 \tag{5}$$

where P_i denotes the path of net i, e denotes a unit edge (between two neighboring grid cells) in P_i, and s^e denotes the number of extra grid cells allocated around edge e, i.e., $\sum_{e \in P_i} s^e = S_i$.

Here, we first simplified the original function in (2) by eliminating the constant terms. Then, we have added the term $\sum_{i \in \mathcal{N}} \sum_{e \in P_i} \epsilon (s^e)^2$ as a penalty term for resource allocation. Note that, ϵ is expected to be a small constant compared to the initial step size t_0 used to update LM. Intuitively, we want the penalty term to be ineffective in earlier iterations, but as the multiplier values start to converge to their optimal values,

[4]The case $\lambda_{mS} = \lambda_{nS}$ would give an arbitrary outcome, so we ignore this case in our discussions.

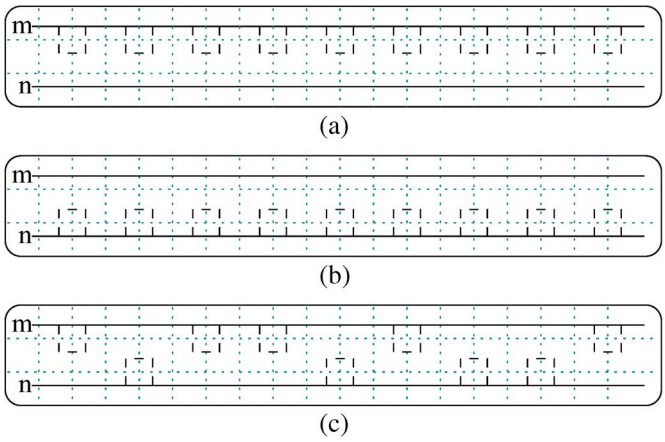


Fig. 4. Parallel routing segments of net m and net n, together with allocated resources (indicated by dashed lines). (a) Resource allocation if $\lambda_{mS} > \lambda_{nS}$. (b) Resource allocation if $\lambda_{nS} > \lambda_{mS}$. (c) Desirable resource allocation if λ_{mS} is slightly larger than λ_{nS}.

we want it to effectively dampen the oscillations. Note that the resulting behavior will be similar to the one illustrated in Fig. 3(c). Also, as a side effect, we had to eliminate the term $-\sum_{i\in\mathcal{N}} \lambda_{iS} L_i$ from the function in (2). The reason can be explained by using the example given in Fig. 3. Assume that both net m and n have small λ_L but large λ_S values, and assume that λ_{nS} is slightly larger than λ_{mS}. Due to the penalty term added, it is possible that the term $-\lambda_{nS} L_n$ dominates instead of $-\lambda_{nS} S_n$, so L_n will be maximized instead of S_n. The result would be similar to the case shown in Fig. 3(b), but this time with a snaking-like behavior[5] instead of resource allocation. Therefore, we also need to remove the term $-\sum_{i\in\mathcal{N}} \lambda_{iS} L_i$. It is interesting to note here the similarity between the new objective function in (5) and the intuitive formula $\sum_{i\in\mathcal{N}} (\alpha_i L_i - \beta_i S_i)$ given in Section III-A.

Another source of possible oscillations is due to the fact that we route all nets using fixed LM values. As shown in Fig. 4, if λ_{mS} is even slightly larger than λ_{nS}, all the intermediate grid cells would be allocated for net m and vice versa to minimize the objective function in (5). The reason for such a behavior is that the LM is updated only after the complete routing solution is found using the fixed multiplier values. For instance, assume that it is required to allocate extra grid cells for both net m and n to satisfy their resource constraints [i.e., as in Fig. 4(c)]. If the solution in iteration k is as in Fig. 4(a), λ_{mS} would be decreased and λ_{nS} would be increased for the next iteration. Therefore, the solution in iteration $k+1$ would be as in Fig. 4(b). Similar arguments suggest that the solution will always oscillate between these two extreme cases.

We propose a simple yet effective heuristic for this problem. First, we rewrite the objective function in (5) without any modifications as follows:

$$\sum_{i\in\mathcal{N}} \sum_{e\in P_i} \left(1 + \lambda_{iL} - \lambda_{iS} s^e + \epsilon (s^e)^2\right). \tag{6}$$

[5]The routing algorithm we use (Section V) maximizes length if all the edge weights are negative.

Again, $e \in P_i$ is a unit edge in the path of net i. This formulation suggests that we need to access the variables λ_{iL} and λ_{iS} for each edge $e \in P_i$. To avoid the oscillation problem described above, we will apply random smoothing each time such an access occurs. Specifically, instead of using λ_{iL}^k and λ_{iS}^k in iteration k, we will use

$$\lambda_{iL} = \alpha \lambda_{iL}^k + (1-\alpha)\lambda_{iL}^{k-1} \tag{7}$$

$$\lambda_{iS} = \alpha \lambda_{iS}^k + (1-\alpha)\lambda_{iS}^{k-1} \tag{8}$$

where α is a random number in the range of [0,1], and it is regenerated for each access to λ_{iL}, λ_{iS} values. Observe that such a smoothing is not expected to effect the results if there are no oscillations (since the multiplier values in iterations $k-1$ and k would be consistent with each other). However, in case of oscillations as in Fig. 4(a) and (b), the result is expected to turn out eventually as in Fig. 4(c).

IV. Graph Model

In this section, we propose a graph model that facilitates resource allocation during shortest path calculations. The significance of this model is that all the extra grid cells allocated for net i can be used for extending the length of net i through snaking. In other words, our low-level routing algorithm will operate on this graph, so that there will be a one-to-one correspondence between resource allocation (during routing) and snaking (in postprocessing). For simplicity of the presentation, we will give the graph model in case the preferred direction (see Section III-A) is RIGHT. It is straightforward to extend this model for the other directions.

As a first step, we define a supernode corresponding to each routing grid cell. A supernode N is defined to contain three subnodes: u_N, d_N, and s_N. Each subnode corresponds to a different state of N in terms of the direction of the incoming edge. Namely, u_N, d_N, and s_N define the cases where the incoming edge to N is upward, downward, and straight, respectively. Fig. 5 illustrates this graph model with an example. Here, supernodes A, B, and C correspond to three neighboring routing grid cells, where B and C are right and down neighbors of A, respectively. All eleven edges are illustrated separately with the corresponding physical explanation. For instance, the edge $s_A \rightarrow s_B$ corresponds to the case where the incoming edge to A is straight, and the connection from A to B is also straight. As another example, the edge $u_A \rightarrow d_B$ corresponds to the case where the incoming edge to A is upwards, and the connection from A to B is through allocating some of the top grid cells. Note that in this case, the direction of the incoming edge to B (from A) is regarded as downwards, assuming that the allocated grid cells will be used for snaking later.

One point to observe in Fig. 5 is that resource allocation is possible only through the edges $u_A \rightarrow d_B$, $d_A \rightarrow u_B$, $s_A \rightarrow d_B$, and $s_A \rightarrow u_B$. This guarantees that all the allocated grid cells during minimum-cost path calculations can be used for snaking later. The issues such as assigning weights to these edges, determining the amount of resource allocation, etc., will

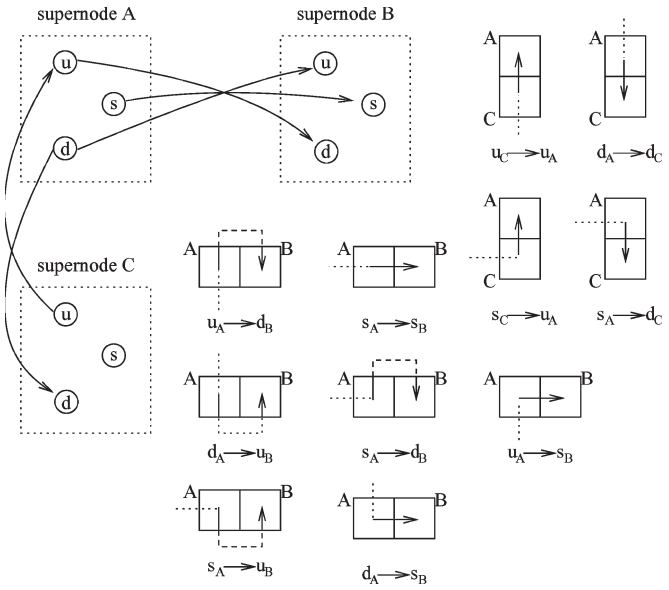


Fig. 5. Three supernodes together with their subnodes are displayed on the upper left corner. Only 5 of the 11 edges are drawn in the big picture for clarity. All 11 edges between supernodes A, B, and C are illustrated separately on the right.

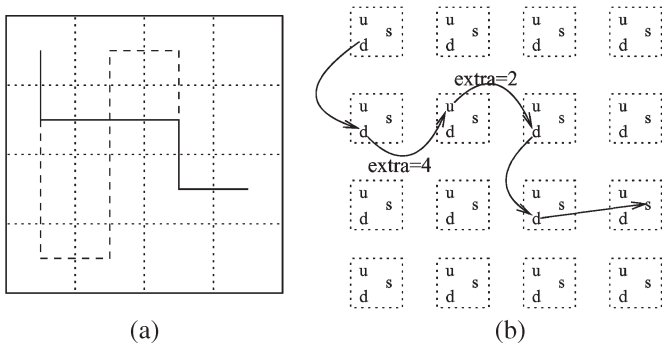


Fig. 6. (a) Example of a routing segment, where allocated grid cells are shown with dashed lines. (b) Corresponding path in our graph model.

be discussed in Section V. However, we can state the following lemma based on the discussions above.

Lemma 4.1: Let $\mathcal{R}$ be the original routing grid, and let $\mathcal{G}$ be the corresponding graph model. For any valid route (with snaking) in $\mathcal{R}$, there exists a corresponding path (with extra resource allocation) in $\mathcal{G}$. Furthermore, for any path P in $\mathcal{G}$, a route can be constructed in $\mathcal{R}$ such that all extra allocated resources are used for snaking.

Fig. 6 shows an example path on the routing grid, and its graph representation. Here, resource allocation is performed for two edges, and the notation extra $= 4$ in Fig. 6(b) means that four extra grid cells are allocated around this edge. Observe that a total of six grid cells is allocated in Fig. 6(a), and it is possible to extend the length of this path from 5 to 11 if all these grid cells are used for snaking.

V. Routing Nets

In this section, we describe the methodology we use to route all nets $i \in \mathcal{N}$, given fixed λ_{iL}, λ_{iS} values. We will first give a brief overview of the Pathfinder negotiated-congestion algorithm in Section V-A. Then, we will discuss how to incorporate our Lagrangian cost functions into this methodology in Section V-B.

A. Negotiated Congestion Algorithm

Our low-level routing algorithm is based on Pathfinder negotiated congestion algorithm, which was originally proposed for field-programmable gate-array-routing problem [1], [2], [9]. The main idea here can be summarized as follows. First, every net is routed individually, regardless of any overuse (i.e., congestion) of routing grid cells. Then, the nets are ripped-up and rerouted one by one iteratively. In each iteration, the congestion cost of each grid cell is updated based on the current and past overuse of it. By increasing the congestion cost of an overused grid cell gradually, the nets with alternative routes are forced not to use this grid cell. Eventually, only the net that needs to use this grid cell most ends up using it. More details about this heuristic-based routing algorithm can be found in [9].

In our implementation, we have used the following congestion cost function for grid cell g in iteration k:

$$\text{congestion cost}(g,k) = k^{\varphi} \cdot t_g \cdot (1 + \text{history}(g,k)) \quad (9)$$

where φ is a constant parameter, t_g is the number of nets that are passing through grid cell g in the current iteration, and $\text{history}(g,k)$ is the congestion history of grid cell g. In the beginning of the algorithm, the congestion history of each grid cell is initialized to zero. Then, after each iteration k, the congestion history of grid cell g is updated as follows:

$$\text{history}(g,k+1) = \text{history}(g,k) + v_{gc} \cdot \max(0, t_g - 1) \quad (10)$$

where v_{gc} denotes the number of consecutive turns in which grid cell g has been congested. Observe here that when a grid cell is congested for multiple iterations consecutively, its history is incremented by a larger value; hence, its congestion cost increases more rapidly. On the other hand, when a grid cell is not congested in the current iteration (i.e., when t_g is 0 or 1), its congestion history remains unchanged.

In the cost function in (9), φ is a user-defined parameter, and it is used to control how fast the congestion costs are increased in the later iterations. In practice, this parameter is set empirically, based on a tradeoff between solution qualities and execution times.

B. Incorporating Length-Matching Objectives

In one iteration of the negotiated-congestion routing algorithm, each routing grid cell has a fixed congestion-cost value, as defined in (9). The problem now is to find the best route and resource allocation for each net i, based on fixed congestion costs and fixed λ_{iL}, λ_{iS} values. Specifically, we want to find path P_i for each net i that minimizes the following expression:

$$\sum_{e \in P_i} \left(1 + \lambda_{iL} - \lambda_{iS} s^e + \epsilon (s^e)^2 + c^e\right) \quad (11)$$

where s^e is the number of extra grid cells allocated around edge e, and c^e is the total congestion cost of the grid cells occupied by edge e. Observe that this expression is obtained by incorporating congestion costs into the Lagrangian objective function in (6), which is defined in Section III-C.

To find the best path for net i, we model the routing grid as a graph using the model described in Section IV. Based on the objective function in (11), the weight of edge e is defined as follows:

$$\text{weight}(e) = \min_{s^e} \left\{ 1 + \lambda_{iL} - \lambda_{iS} s^e + \epsilon (s^e)^2 + c^e \right\}. \quad (12)$$

As described in Section IV, some types of edges are not suitable for resource allocation. If e is such an edge, then s^e is set to zero, and c^e is set to the sum of the congestion costs of the two grid cells connected by this edge. Otherwise, s^e is selected[6] so as to minimize weight(e) in (12). Note that increasing s^e means allocating more grid cells, hence possibly increasing c^e. Here, the optimal value of s^e depends on the value of λ_{iS} (i.e., the importance of resource allocation constraint), and congestion costs of the grid cells around this edge. For this graph model and the weight function, we can state the following theorem.

Theorem 5.1: Let $\mathcal{R}$ be the original routing grid and let $\mathcal{G}$ be the corresponding graph model as defined in Section IV and edge weights set based on (12). The shortest path P in $\mathcal{G}$ corresponds to the best route (with resource allocation) in $\mathcal{R}$ that minimizes the objective function in (11).

Proof: From Lemma 4.1, we know that there is a one-to-one correspondence between any valid route in $\mathcal{R}$ and any path in $\mathcal{G}$. Furthermore, consider an arbitrary path in the form: $v_0 \rightsquigarrow v_i \rightarrow v_j \rightsquigarrow v_n$, where v_0 and v_n are the terminal nodes, and v_i and v_j are any intermediate neighboring nodes. According to the graph model given in Section IV, the types of vertices v_i and v_j completely determine whether space allocation is possible around the edge $(v_i \rightarrow v_j)$. Furthermore, changing the amount of space allocation around this edge does not affect the paths $v_0 \rightsquigarrow v_i$ and $v_j \rightsquigarrow v_n$, since all paths are defined to be monotonic in the horizontal direction (see Section III-A). In other words, the amount of space allocation on a particular edge does not affect the solution for the rest of the path. Therefore, the value s^e around any edge e must be selected as defined in (12) to minimize the objective function in (11).

After setting the edge weights, the next step is to find the minimum-cost path for net i. Intuitively, defining edge weights as in (12) has two important consequences: Shorter nets (with large λ_{iS} values) will automatically prefer the routes where they can allocate enough resources around. On the other hand, longer nets will probably not be detoured despite congestion costs, because λ_{iL} will dominate the weight function in (12) for small or moderate congestion levels. Closer examination of the edges illustrated in Fig. 5 will reveal that our graph is in fact a directed acyclic graph (dag). It is known that the shortest path problem can be solved for a weighted dag in linear time [7].

[6]In our implementation, we have defined a small preset upper bound value (e.g., 10) for s^e, and we tried each even number between zero and this upper bound to find the optimal s^e value that minimizes the weight function defined.

ROUTE-ALL-NETS (Inputs: λ_{iL}, λ_{iS} for each net i)
Initialize congestion cost of each grid cell to zero
while a congestion-free routing solution not found do
  for each net $i \in \mathcal{N}$ do
    calculate edge weights
    find min cost path for net i
  increase congestion costs of overused grid cells

Fig. 7. Low-level algorithm description to route nets based on fixed LM values.

The overall method described in this section is summarized in Fig. 7.

VI. Generalizing the Models

The models and algorithms in the previous sections mainly focus on routing a single bus on a single layer, and it is assumed that all routes are monotonic in one direction. However, it is straightforward to extend these ideas to more general cases.

For a multilayer layout, we can use a three-dimensional (3-D)-grid model, where the third dimension corresponds to interlayer connections. Here, the graph model proposed in Section IV and the weight calculation scheme given by (12) can be applied to each layer independently. However, the main difference here is in modeling interlayer connections. Assume that grid cells A and B are in different layers, and a via connection is possible between them. To model such a connection, we need to create edges between all subnodes of A and B. Since resource allocation is not applicable here, the weight of these edges would only reflect the length requirement and congestion. For example, we can modify (12) for this purpose as follows:

$$\text{weight}(e_{\text{via}}) = (\lambda_{iL} d_{\text{via}} + c^e) \times \text{via\_penalty}. \quad (13)$$

Note that an interlayer connection is likely to have different delay characteristics than a regular intralayer connection, so it might be necessary to use a circuit-dependent factor d_{via} to model such difference. Furthermore, since via connections are typically undesired, the constant via_penalty is used to avoid using these edges unless they are really necessary. After that, we can use our low-level routing algorithm on this 3-D-grid structure. However, note that since this multilayer graph structure is not acyclic, we need to use a shortest path algorithm that can handle edges with negative weights,[7] such as Bellman–Ford algorithm [7]. Fig. 8 shows a sample routing solution on two layers. Here, each of nets 2 and 3 is routed on a single layer, while net 1 uses a via to switch layers. Note that the only difference in the multilayer routing model is that a net can use a via

[7]Edges with negative weights are possible due to the weight function given in (12). However, it is guaranteed that there is no negative-weight cycle, since we assume that each net has the same preferred direction in all layers. For example, if the preferred direction is RIGHT, then there will be no detour toward LEFT. Hence, a cycle cannot contain a horizontal edge. Since resource allocation is not possible around vertical edges, it is guaranteed that all edges in a cycle have positive weights.

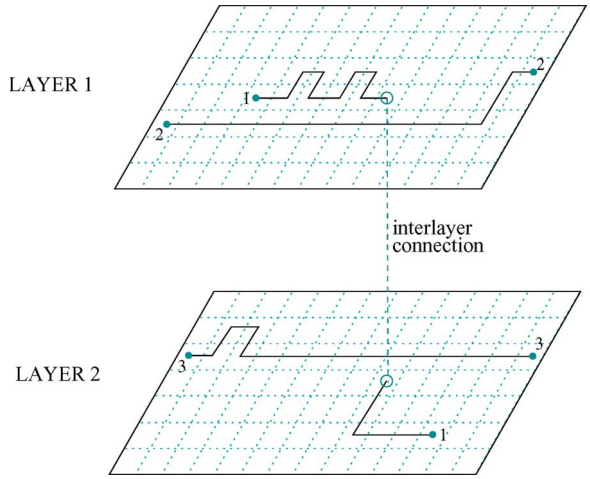


Fig. 8. Sample solution with three nets routed on two layers. The interlayer connection for net 1 is illustrated with a dashed line. Note that snaking is performed on each layer the same way as in a single-layer model.

to go in the third dimension during path calculations. By assigning a high cost to interlayer connections, we can avoid using vias if they are not really needed.

Also, we can extend these models for routing multiple buses together. For this, we need to modify the original formulation in (1) such that each bus uses a different target length T. We can also extend this formulation to the most general case, where each net has different upper and lower bound constraints

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in \mathcal{N}} L_i \\ \text{subject to :} \quad & \forall i, \quad L_i \leq T_i^{\text{ub}} \\ & \forall i, \quad L_i + S_i \geq T_i^{\text{lb}} \end{aligned} \tag{14}$$

where T_i^{ub} and T_i^{lb} are the upper and lower bounds for net i. Note that such a modification in constraints would only effect the update schedule of LM. Namely, the multipliers for iteration $k+1$ would be calculated as follows:

$$\lambda_{iL}^{k+1} = \begin{cases} \max\left(0, \lambda_{iL}^k - t_k \left(T_i^{\text{ub}} - L_i\right)^{\gamma}\right), & \text{if } L_i \leq T_i^{\text{ub}} \\ \lambda_{iL}^k + t_k v_{iL} \left(L_i - T_i^{\text{ub}}\right)^{\gamma}, & \text{otherwise.} \end{cases} \tag{15}$$

$$\lambda_{iS}^{k+1} = \begin{cases} \max\left(0, \lambda_{iS}^k - t_k \left(L_i + S_i - T_i^{\text{lb}}\right)^{\gamma}\right), & \text{if } L_i + S_i \geq T_i^{\text{lb}} \\ \lambda_{iS}^k + t_k v_{iS} \left(T_i^{\text{lb}} - L_i - S_i\right)^{\gamma}), & \text{otherwise.} \end{cases} \tag{16}$$

It is also possible to generalize our algorithms to the case where each net has an individual preferred direction, instead of a single-global preferred direction for all nets. In other words, some nets can be specified as monotonic in the horizontal direction, while some others are monotonic in the vertical direction. Note that monotonicity of routes in one direction is required for the graph model we propose in Section IV, which ensures that all the extra routing resources allocated by our low-level routing algorithm can be used for length extension. In Section IV, we have given a graph model for routes that are monotonic in the horizontal direction, and it is straightforward to generalize it for monotonicity in the vertical direction. Since our low-level routing algorithm (given in Fig. 7) routes nets one-by-one, different graph models can be used for different nets, based on the prespecified preferred directions.[8] As an example, consider Fig. 9, where two nets are monotonic in the vertical direction, and one net is monotonic in the horizontal direction. Our algorithm has successfully found the routing solution where all three nets have exactly the same length.

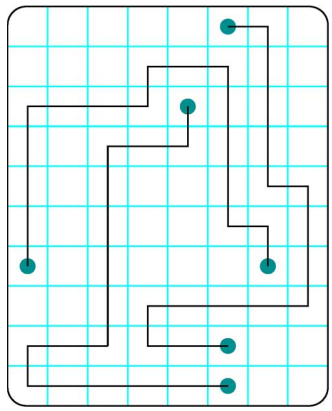


Fig. 9. Sample routing solution where two nets are monotonic in the vertical direction, and one net is monotonic in the horizontal direction. The lengths of all three nets have been matched by our algorithm.

VII. Experimental Results

In this section, we compare our framework with a commonly used greedy approach. In this approach, a negotiated-congestion routing algorithm (similar to Pathfinder [1], [2], [9]) is used to find a congestion-free routing solution for all nets. As described before, the main idea here is to route each net regardless of any congestion in the beginning; then the costs for congested routing resources are increased gradually, forcing the nets to use alternative routes. Note that this algorithm does not explicitly consider the objective of length matching during path calculations. Instead, after a conflict-free routing solution is found for all nets, a greedy postprocessing method is used for the purpose of length matching. Here, each net is processed (from shortest to longest), and jogs are inserted (as in Fig. 1) until it satisfies the minimum length constraint.

During implementation of our algorithm, we have used a heuristic to expedite the convergence of the solution. Here, after all the nets are routed for fixed λ_{iL}, λ_{iS} values in one iteration, it might be the case that some nets have more than necessary allocated resources. Our heuristic is to deallocate the extra resources from all these nets, and greedily allocate the available routing grids for shorter ones. We have observed that this heuristic decreases the running time of our algorithm.

We have implemented all these algorithms in C++, and we have performed our experiments on an Intel Xeon 2.4-Ghz system with 512-MB memory and a Linux operating system.

[8]The preferred direction for a net can be determined heuristically based on the relative positions of its terminals. It is also possible to try both directions one by one, and then choose the one that gives the better route.

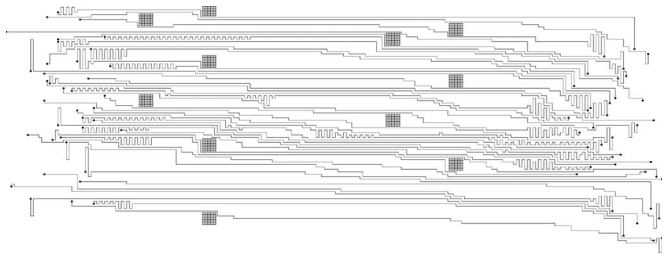


Fig. 10. Sample routing solution using LR-based resource allocation.

For illustration purposes, we have applied our algorithm on a relatively small-sized single-layer bus-routing problem,[9] the outcome of which is displayed in Fig. 10. Here, there are various nets that are routed[10] with almost the same path lengths. Specifically, we have set the constant Δ in the objective function in (1) to one in this paper.[11] In accordance with this constraint, the difference between the minimum and maximum path lengths is only one grid cell in the solution of Fig. 10. Observe that snaking could be performed even in the dense areas of the layout. Furthermore, the heights of these jogs are usually small (i.e., one or two grid cells most of the time), mainly due to the methods proposed in Section III-C to avoid solution oscillations.[12] As a result, multiple nets effectively share the available routing resources such that all satisfy their minimum length constraints.

We have also performed experiments on test-problems properties of which are summarized in Table I. Here, vertical spacing is measured in terms of the number of grid cells between the terminal points of adjacent nets, and it indicates how dense the problem is. On the other hand, Manhattan distance is given in terms of number of grid cells between two terminals of the same net. The deviation in this value is a good indicator for the amount of snaking needed to be performed. Each bus given in this table has around a hundred nets, and the objective is to route them and match their lengths. Note also that the underlying grid sizes are between 150×222 and 150×459, depending on the problem size. Similar to Fig. 10, the nets in these problems are monotonic in the same direction. Furthermore, the net terminals in the single-layer problems are ordered, as in Fig. 10, to ensure that a planar routing solution exists. On the other hand, the terminals in the two-layer problems are not ordered, since via usage is permitted for these problems.

As described before, our formulations involve some parameters due to the high-level-LR framework (Fig. 2) and the low-level negotiated-congestion algorithm (Fig. 7). In this paper, we have set these parameters as follows. In the update schedule for LM given in (3) and (4), we have set the step size t_k such that the convergence criterion given by Held *et al.* [15] is satisfied, i.e., as $k \rightarrow \infty$, it should be the case that $t_k \rightarrow 0$ and $\sum_{i=1}^{k} t_i \rightarrow \infty$, where k is the current iteration number. Specifically, we have used the function $t_k = 1/\sqrt{k}$ for this purpose. As mentioned before, exponent γ in these equations is expected to have a small value to smoothen the effect of the amount of length or resource constraint violations. Therefore, we have set $\gamma = 0.1$ in this paper. Similarly, we have set ϵ in the Lagrangian cost function [given in (5)] to 0.1. Remember that the penalty term $\sum_{i \in \mathcal{N}} \sum_{e \in P_i} \epsilon (s^e)^2$ in this function has been introduced to avoid potential solution oscillations. Setting ϵ to such a small value makes the penalty term ineffective in earlier iterations and dampens the oscillations as multiplier values start to converge to their optimal values, as described in Section III-C. For the congestion-cost function given in (9), we have empirically set φ to 0.4. As discussed earlier, the main idea of the Pathfinder algorithm is to gradually increase the congestion costs of the overused grid cells. Here, parameter φ determines how fast the congestion costs are incremented. Finally, for the two-layer problems in Table I, we have used (13) to set the weights of via edges, as described in Section VI. In this paper, we have empirically set the $\mathrm{via\ penalty}$ multiplier in this equation to four to discourage via usage. Note that our algorithms inherit the heuristic nature of Pathfinder [1], [2], [9], and successful routing of all nets is not guaranteed.

We have executed both the greedy algorithm mentioned before and the LR-based routing algorithm on these test problems. The comparison of the results are given Table II. Here, minL, maxL, and stdev denote the minimum path length, maximum path length, and standard deviation in path lengths, respectively, and all results are given in terms of the number of grid cells spanned. Moreover, the execution times of these algorithms are given under the column time, and they are reported with min:sec units. Observe that the greedy method fails to match lengths especially when the problem is dense, or the variation in net lengths is large. However, our method performs multiple iterations in such cases to effectively find the solution that satisfies length constraints. Due to these multiple iterations, the execution time increases; nevertheless, the feasible solution is obtained eventually. In this paper, we have observed that the high-level Lagrangian framework (as given in Fig. 2) took less than ten iterations most of the time. On the other hand, the low-level negotiated-congestion algorithm (as given in Fig. 7) took around 50–100 iterations. We have also observed that most of the nets are routed without congestion in the first few iterations; then the remaining few nets negotiate congested resources in the later iterations.

VIII. Conclusion

We have proposed an algorithm for routing nets within minimum and maximum length bounds. We can summarize our contributions in this paper as follows: 1) a high-level-LR framework that guides the routing iterations such that length-matching objectives are eventually satisfied; 2) incorporating the resource allocation objectives (which are guided by a

[9] Although a single-layer routing solution is illustrated here, our algorithm applies equally well to multilayer problems.

[10] We have not fine tuned our program to reduce the number of bend points. However, if these are undesirable, it is possible to eliminate them in the postprocessing.

[11] The length of a route can be extended only by an even number of grid cells. Therefore, if there are two different nets (one with an even path length, one with an odd path length), their lengths can be matched only up to one grid cell difference.

[12] As mentioned before, we use a preset upper bound value for s^e in function (12), effectively limiting the maximum height of a jog. However, in Fig. 10, the jogs have heights even smaller than this upper bound most of the time.

TABLE I
PROPERTIES OF TEST PROBLEMS

Test Problem	Vertical Spacing avg	Vertical Spacing stdev	Manhattan Dist. avg	Manhattan Dist. stdev	Grid size	Net count	Layer count
B1	3.54	2.25	106	14.43	150×356	99	one
B2	2.74	2.24	106	7.19	150×280	100	one
B3	2.66	1.72	107	17.31	150×261	96	one
B4	2.23	1.38	107	17.07	150×222	97	one
B5	3.29	2.64	117	15.05	150×459	133	two
B6	2.50	1.68	116	10.37	150×357	135	two
B7	2.30	1.43	117	18.47	150×325	133	two
B8	1.93	1.37	116	11.19	150×277	134	two
B9	1.81	1.18	117	15.74	150×231	118	two
B10	1.73	0.97	117	14.88	150×250	134	two

TABLE II
ROUTING RESULTS ON TEST PROBLEMS

Test Problem	GREEDY SNAKING minL	maxL	stdev	time	LR BASED ROUTING minL	maxL	stdev	time
B1	140	141	0.50	0:08	140	141	0.50	0:08
B2	99	127	2.78	0:11	121	122	0.50	0:19
B3	91	142	7.12	0:14	145	146	0.50	0:41
B4	66	150	10.71	0:12	145	146	0.50	3:27
B5	150	151	0.50	0:22	150	151	0.50	0:23
B6	109	140	3.04	0:25	139	140	0.50	0:47
B7	132	161	3.44	0:24	160	161	0.50	0:49
B8	103	147	6.78	0:24	144	145	0.50	4:25
B9	100	152	4.91	0:20	151	152	0.50	0:44
B10	96	152	7.75	0:25	151	152	0.50	9:46

Lagrangian function) into a state-of-the-art routing algorithm; and 3) a special graph model G such that the shortest path in G corresponds to the optimal resource allocation for the current LM of a net.

This paper indicates that our algorithm can be effectively used for routing nets with min–max length constraints, even in the situations where the greedy strategy fails to satisfy these constraints.

REFERENCES

[1] V. Betz and J. Rose, "Directional bias and non-uniformity in FPGA global routing architectures," in *Proc. ICCAD*, 1996, pp. 652–659.

[2] ——, "VPR: A new packing, placement and routing tool for FPGA research," in *Proc. 7th Int. Workshop Field-Programmable Logic*, 1997, pp. 213–222.

[3] K. D. Boese, J. Cong, A. B. Kahng, K. S. Leung, and D. Zhou, "On high-speed VLSI interconnects: Analysis and design," in *Proc. Asia-Pacific Conf. Circuits Syst.*, 1992, pp. 35–40.

[4] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, "Bounded-skew clock and steiner routing," *ACM Trans. Design Automat. Electron. Syst.*, vol. 3, no. 3, pp. 341–388, Jul. 1998.

[5] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Performance-driven global routing for cell based IC's," in *Proc. IEEE Int. Conf. Comput. Des.*, 1991, pp. 170–173.

[6] ——, "Probably good performance-driven global routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 11, no. 6, pp. 739–752, Jun. 1992.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press, 1992.

[8] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, "Chip layout optimization using critical path weighting," in *Proc. Des. Autom. Conf.*, 1984, pp. 133–136.

[9] C. Ebeling, L. McMurchie, and S. A. Hauck, "Placement and routing tools for the triptych FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 3, no. 4, pp. 473–482, Dec. 1995.

[10] M. L. Fisher, "The Lagrangian relaxation method for solving integer programming problems," *Manage. Sci.*, vol. 27, no. 1, pp. 1–18, 1981.

[11] ——, "An applications oriented guide to Lagrangian relaxation," *Interfaces*, vol. 15, no. 2, pp. 10–21, 1985.

[12] A. M. Geoffrion, "Lagrangian relaxation and its uses in integer programming," *Math. Program.*, vol. 2, pp. 82–114, 1974.

[13] X. Guan, P. B. Luh, and L. Zhang, "Nonlinear approximation method in Lagrangian relaxation-based algorithms for hydrothermal scheduling," *IEEE Trans. Power Syst.*, vol. 10, no. 2, pp. 772–778, May 1995.

[14] X. H. Guan, Q. Z. Zhai, and F. Lai, "New Lagrangian relaxation based algorithm for resource scheduling with homogeneous subproblems," *J. Optim. Theory Appl.*, vol. 113, no. 1, pp. 65–82, 2002.

[15] M. H. Held, P. Wolfe, and H. D. Crowder, "Validation of subgradient optimization," *Math. Program.*, vol. 6, no. 1, pp. 62–88, 1974.

[16] D. J.-H. Huang, A. B. Kahng, and C.-W. A. Tsao, "On the bounded-skew clock and steiner routing problems," in *Proc. 32nd ACM/IEEE Des. Autom. Conf.*, 1995, pp. 508–513.

[17] A. B. Kahng and G. Robins, *On Optimal Interconnections in VLSI*. Norwell, MA: Kluwer, 1995.

[18] E. Kuh, M. A. B. Jackson, and M. Marek-Sadowska, "Timing-driven routing for building block layout," in *Proc. IEEE Int. Symp. Circuits Syst.*, 1987, pp. 518–519.

[19] S. Lee and M. D. F. Wong, "Timing-driven routing for FPGAs based on Lagrangian relaxation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 22, no. 4, pp. 506–510, Apr. 2003.

[20] M. Pecht and Y. T. Wong, *Advanced Routing of Electrical Modules*, Boca Raton, FL: CRC Press, 1996.

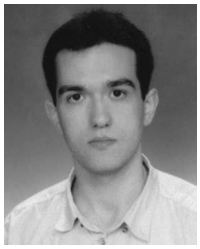
[21] S. Prastjutrakul and W. J. Kubitz, "A timing-driven global router for custom chip design," in *Proc. IEEE Int. Conf. Comput.-Aided Des.*, 1990, pp. 48–51.

[22] A. Renaud, "Daily generation management at Electricite de France: Form planning toward real time," *IEEE Trans. Autom. Control*, vol. 38, no. 7, pp. 1080–1093, Jul. 1993.

[23] L. W. Ritchey. (2000, Dec.). "Busses: What are they and how do they work?" *Print. Circuit Des. Mag.* [Online]. Available: http://www.speedingedge.com/PDF-Files/busses.pdf

[24] C.-W. A. Tsao and C.-K. Koh, "UST/DME: A clock tree router for general skew constraints," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 7, no. 3, pp. 359–379, Jul. 2002.

[25] S. J. Wang, S. M. Shahidehpour, D. S. Kirschen, S. Mokhtari, and G. D. Irisari, "Short-term generation scheduling with transmission constraints using augmented Lagrangian relaxation," *IEEE Trans. Power Syst.*, vol. 10, no. 3, pp. 1294–1301, Aug. 1995.



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 in 2005.

He is currently with the Design and Technology Solutions Department of Intel Corporation, Hillsboro, OR. His current research interests include algorithms for computer-aided design of very large scale integrated circuits, with primary focus on physical-design algorithms.

Martin D. F. Wong (M'88–SM'04–F'06) received the B.Sc. degree in mathematics from the University of Toronto, Toronto, ON, Canada, and the M.S. degree in mathematics and the Ph.D. degree in computer science from University of Illinois at Urbana–Champaign (UIUC), in 1987.

He is currently a Professor of electrical and computer engineering at UIUC. Before he joined UIUC, he was a Bruton Centennial Professor of computer sciences at the University of Texas at Austin. His research interests are computer-aided design (CAD) of very large scale integrated (VLSI) circuits, design and analysis of algorithms, and combinatorial optimization. He has published 300 technical papers and has graduated 32 Ph.D. students. He is a coauthor of *Simulated Annealing for VLSI Design* (Kluwer, 1988) and two invited articles in the *Wiley Encyclopedia of Electrical and Electronics Engineering* in 1999.

Dr. Wong is the recipient of the 2000 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS Best Paper Award for his work on interconnect optimization. He is also the recipient of Best Paper Awards at DAC-86 and ICCD-95 for his work on floorplan design and routing, respectively. His ICCAD-94 paper on circuit partitioning has been included in the book *The Best of ICCAD—20 Years of Excellence in Computer-Aided Design*, published in 2002. He has served as Technical Program Chair (1998), General Chair (1999), and a Steering Committee member (2001, 2002, 2005) for the annual ACM International Symposium on Physical Design. He also regularly serves on the technical program committees of all major VLSI/CAD conferences (e.g., Design Automation Conference, International Conference on Computer-Aided Design, International Symposium on Physical Design, Design Automation and Test on Europe, Asia and South Pacific Design Automation Conference, International Symposium on Circuits and Systems, Field Programmable Gate Array, Synthesis and System Integration of Mixed Information, Great Lakes Symposium on Very Large Scale Integration, and Southwest Symposium on Mixed Signal Design). He has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS from 1995 to 2000 and IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS from 2002 to 2005. He has also served as a Guest Editor of four special issues for TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He is currently on the Editorial Board of the *Association for Computing Machinery Transactions on Design Automation of Electronic Systems*. He is an IEEE Distinguished Lecturer.