LENGTH-MATCHING ROUTING FOR HIGH-SPEED PRINTED CIRCUIT BOARDS

Muhammet Mustafa Ozdal

Dept. of Computer Science
Univ. of Illinois at Urbana-Champaign
Urbana, IL 61801
ozdal@uiuc.edu

Martin D. F. Wong

Dept. of Electrical and Computer Engineering
Univ. of Illinois at Urbana-Champaign
Urbana, IL 61801
mdfwong@uiuc.edu

ABSTRACT

As the clock frequencies used in industrial applications increase, the timing requirements imposed on routing problems become tighter. So, 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 the literature that ensures that minimum length constraints are also satisfied. In this paper, we propose a novel algorithm that effectively incorporates the min-max length constraints into the routing problem. Our approach is to use a Lagrangian relaxation framework to allocate extra routing resources around nets simultaneously during routing them. We also propose a graph model that ensures that all the allocated routing resources can be used effectively for extending lengths. Our 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. Our experiments demonstrate that this algorithm is effective even in the cases where length constraints are tight, and the layout is dense.

1. INTRODUCTION

Routing nets within minimum and maximum length bounds is an important requirement for high-speed VLSI layouts. There has 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 [14, 6, 3, 15, 4]. 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 non-sophisticated strategies (such as greedy length extension in post-processing) 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.

Timing constraints are commonly imposed on PCB 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 destination pins *approximately* at the same time. To achieve this, all the wires constituting this bus need to have *approximately* same lengths. The precision with which matching must be done is directly related to the clock frequency [17]. As the clock frequency increases, the skew requirements on the propagation delays become more strict, 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 max 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. So, it is likely that some nets can not 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, (2) the shorter routes have available routing space around themselves such that it is possible to match all lengths by snaking in 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 Figure 1. Here, there are three nets that need to be routed with equal lengths, and the figure illustrates a typical routing solution[1] given by a conventional router . Here, all nets were routed first, and then snaking was performed in the end for length matching. Observe that the top net turned out to be the longest one, with a path length[2] of 17. So, the length of the bottom net was extended by 6 through snaking. However, snaking was not possible for the middle net, because all routing resources around its route were used during routing. So, length matching fails in this example.

Figure 2 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

This work was partially supported by the National Science Foundation under grant CCR-0244236.

ICCAD'03, November 11-13, 2003, San Jose, California, USA.

[1] The underlying grid structure is also shown in this figure. Throughout the paper, we assume that routing edges go center-to-center of each grid cell, as illustrated in this figure. 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.

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

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 3, 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 Lagrangian relaxation on this formulation, our objective becomes minimization of:

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

Here, each λ_{iL} and λ_{iS} are Lagrangian multipliers corresponding to length and resource constraints given in the original formulation (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 Figure 3. For the following discussions in this section, assume that we have a subroutine for finding the routing solution that minimizes objective function (2), for fixed λ_{iL}, λ_{iS} values. Observe in Figure 3 that we iteratively call this subroutine, and update the Lagrangian multipliers 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:

IF $L_i \leq T$ THEN
  $\lambda_{iL}^{k+1} = max(0, \lambda_{iL}^k - t_k(T - L_i)^\gamma)$
ELSE
  $\lambda_{iL}^{k+1} = \lambda_{iL}^k + t_k v_{iL}(L_i - T)^\gamma$;

IF $L_i + S_i \geq T - \Delta$ THEN
  $\lambda_{iS}^{k+1} = max(0, \lambda_{iS}^k - t_k(L_i + S_i - T + \Delta)^\gamma)$
ELSE
  $\lambda_{iS}^{k+1} = \lambda_{iS}^k + t_k v_{iS}(T - \Delta - L_i - S_i)^\gamma)$

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 0. Specifically, we use the convergence condition given by Held, et al[13], 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 ($L_i \leq T$) and the resource constraint ($L_i + S_i \geq T - \Delta$) 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, $\gamma \leq 1$ 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.

ROUTE-AND-LENGTH-MATCH(Inputs: $\mathcal{N}$, T)
Initialize λ_{iL}, λ_{iS} to zero for each $i \in \mathcal{N}$
while termination condition not occurred do
  simultaneously route all nets for fixed λ_{iL}, λ_{iS} values
  for each $i \in \mathcal{N}$ do
    check constraints for current route
    update λ_{iL} and λ_{iS} values accordingly
perform snaking using the extra grid cells allocated

Figure 3: High level algorithm description

2.3. Handling Oscillation Problems

It is known that solution oscillation is a serious and inherent problem for Lagrangian relaxation based methods [11, 16]. Note that even if the Lagrangian multipliers 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. [12] 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 subsection.

Figure 4 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 (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 function (2) will be minimized if S_m has its maximum value. Hence, all the intermediate grid cells will be allocated by net m (Figure 4(a)). On the other hand, if $\lambda_{nS} > \lambda_{mS}$, then S_n will be set to its maximum value as in Figure 4(b) to minimize the objective function. Note that even if the difference between two Lagrangian multipliers 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 like as shown in Figure 4 (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 Lagrangian relaxation* [16, 18], where a penalty term is added to the Lagrangian function to avoid oscillations. Using a similar idea, we can modify objective function (2) such that our new objective becomes the minimization of:

$$\sum_{i \in \mathcal{N}} (L_i + \lambda_{iL} L_i - \lambda_{iS} S_i) + \sum_{i \in \mathcal{N}} \sum_{e \in P_i} \epsilon (s^e)^2 \tag{3}$$

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 (2) by eliminating the constant terms. Then, we have added the term

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

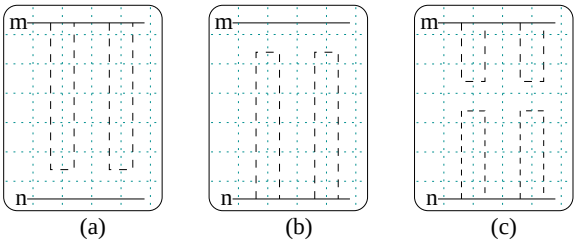

Figure 4: Parallel routing segments of net m and net n, together with allocated routing resources (indicated by dashed lines). (a) Resource allocation if $\lambda_{mS} > \lambda_{nS}$, (b) Resource allocation if $\lambda_{nS} > \lambda_{mS}$, (c) Desirable resource allocation if λ_{nS} is only slightly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

$\sum_{i\in\mathcal{N}}\sum_{e\in P_i}\epsilon(s^e)^2$ 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 Lagrangian multipliers. Intuitively, we want the penalty term to be ineffective in earlier iterations, but as the multiplier values start to converge to their optimal values, we want it to effectively dampen the oscillations. Note that the resulting behavior will be similar to the one illustrated in Figure 4(c). Also as a side effect, we had to eliminate the term $-\sum_{i\in\mathcal{N}}\lambda_{iS}L_i$ from function (2). The reason can be explained by using the example given in Figure 4. 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 Figure 4(b), but this time with a *snaking-like* behavior[5] instead of resource allocation. So, 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 (3), and the intuitive formula $\sum_{i\in\mathcal{N}}(\alpha_i L_i - \beta_i S_i)$, given in Section 2.1.

Another source of possible oscillations is due to the fact that we simultaneously route all nets using fixed Lagrangian multiplier values. As shown in Figure 5, 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 objective function (3). The reason for such a behavior is that the Lagrangian multipliers are 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 Figure 5(c)). If the solution in iteration k is as in Figure 5(a), λ_{mS} would be decreased, and λ_{nS} would be increased for the next iteration. So, the solution in iteration $k+1$ would be as in Figure 5(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 (3) without any modifications as follows:

$$\sum_{i\in\mathcal{N}}\sum_{e\in P_i}(1+\lambda_{iL}-\lambda_{iS}s^e+\epsilon(s^e)^2) \tag{4}$$

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

[5]The routing algorithm we use (Section 4) maximizes length if all the edge weights are negative.

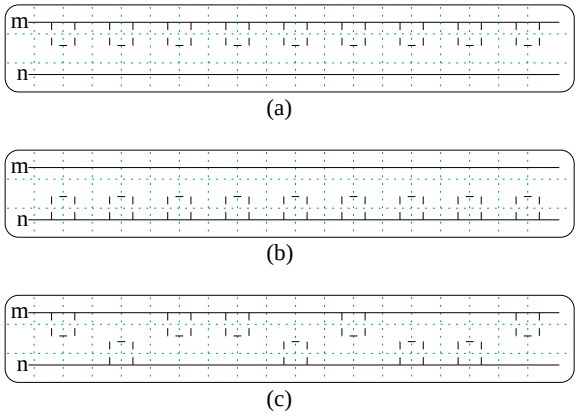

Figure 5: Parallel routing segments of net m and net n, together with allocated resources (indicated by dashed lines). (a) Resource allocation if $\lambda_{mS} > \lambda_{nS}$, (b) Resource allocation if $\lambda_{nS} > \lambda_{mS}$, (c) Desirable resource allocation if λ_{mS} is slightly larger than λ_{nS}.

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:

$$\lambda_{iL} = \alpha\lambda_{iL}^k + (1-\alpha)\lambda_{iL}^{k-1} \tag{5}$$

$$\lambda_{iS} = \alpha\lambda_{iS}^k + (1-\alpha)\lambda_{iS}^{k-1} \tag{6}$$

where α is a random number in the range $[0,1]$, and it is regenerated for each access to λ_{iL} and λ_{iS} values. Observe that such a smoothing is not expected to affect 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 Figures 5(a) and (b), the result is expected to turn out eventually as in Figure 5(c).

3. 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. For simplicity of the presentation, we will give the graph model in case the *preferred direction*(see Section 2.1) 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 is defined to contain three subnodes: , , and s . Each subnode corresponds to a different state of in terms of the direction of the incoming edge. Namely, , , and s define the cases in which the incoming edge to is *upwards*, *downwards*, and *straight*, respectively. Figure 6 illustrates this graph model with an example. Here, supernodes , , and correspond to three neighboring routing grid cells, where and are right and down neighbors of , respectively. All eleven edges are illustrated separately with the corresponding physical explanation. For instance, the edge $s \rightarrow s$ corresponds to the case where the incoming edge to is straight, and the connection from to is also straight. Or, the edge $\rightarrow$ corresponds to the case where the incoming edge to is upwards, and the connection from to is through allocating some of the top grid cells. Note

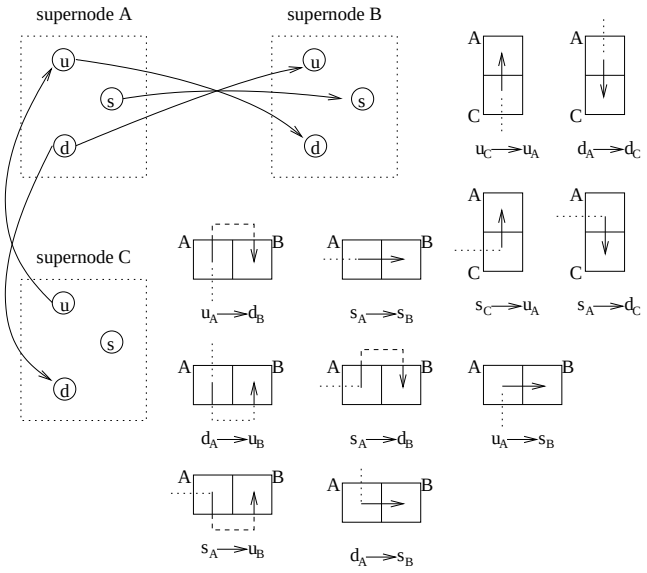

Figure 6: Three supernodes together with their subnodes are displayed on the upper left corner. Only five of the eleven edges are drawn in the big picture for clarity. All the eleven edges between supernodes , , and are illustrated separately on the right.

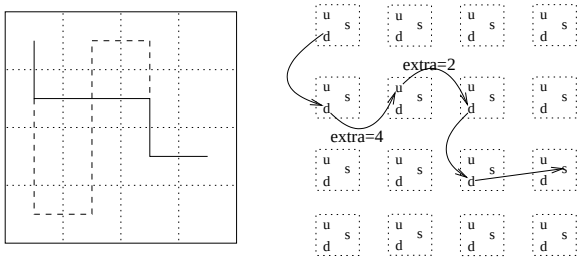

Figure 7: (a) An example routing segment, where allocated grid cells are shown with dashed lines, (b) The corresponding path in our graph model.

that in this case, the direction of the incoming edge to (from) is regarded as downwards, assuming that the allocated grid cells will be used for snaking later.

One point to observe in Figure 6 is that resource allocation is possible only through the edges $\rightarrow$, $\rightarrow$, $s \rightarrow$, and $s \rightarrow$. This guarantees that all the allocated grid cells during min-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 be discussed in Section 4.

Figure 7 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 part (b) means that four extra grid cells are allocated around this edge. Observe that a total of six grid cells is allocated in part (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.

ROUTE-ALL-NETS(Inputs: λ_{iL}, λ_{iS} values 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

Figure 8: Low level algorithm description

4. ROUTING NETS

In this section we describe the technique we use to route all the nets $i \in \mathcal{N}$, given fixed λ_{iL} and λ_{iS} values. Our algorithm is based on the Pathfinder negotiated congestion algorithm [7, 1, 2] originally proposed for FPGA routing problem. 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 will end up using it.

In one iteration of this algorithm, each routing grid cell has a fixed congestion cost value. The problem here is to find the best path and resource allocation for each net i, based on fixed congestion costs and fixed λ_{iL}, λ_{iS} values. For this, we will model the routing grid as a graph using the model described in Section 3. To calculate the edge weights, we incorporate congestion costs into the Lagrangian objective function (4) defined in Section 2.3. Specifically for a net i, the weight of edge e will be calculated as:

$$ei \quad t(e) = 1 + \lambda_{iL} - \lambda_{iS} s^e + \epsilon (s^e)^2 + \quad ^e \qquad (7)$$

where s^e is the number of extra grid cells allocated around this edge, and ^{e} is the total congestion cost of the grid cells *occupied* by this edge. As described in Section 3, some types of edges are not suitable for resource allocation. If e is such an edge, then s^e is set to zero, and ^{e} is set to the sum of the congestion costs of the two grid cells connected by this edge. Otherwise, s^e is selected so as to minimize $ei \quad t(e)$ in Equation 7. Note that increasing s^e means allocating more grid cells, hence possibly increasing ^{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.

After setting edge weights, the next step is to find the minimum cost path for net i. Note that shorter nets would automatically prefer the routes where they can allocate enough resources around. On the other hand, longer nets would probably not be detoured despite congestion costs, because λ_{iL} would dominate Equation 7 for small or moderate congestion levels.

Closer examination of the edges illustrated in Figure 6 would reveal that our graph is in fact a DAG (directed acyclic graph). It is known that minimum cost path problem can be solved for a weighted DAG in linear time[5].

The overall method described in this section is summarized in Figure 8.

5. GENERALIZING THE MODELS

The models and algorithms in the previous sections mainly focus on routing a single bus on a single layer. However, it is straightforward to extend these ideas to more general cases.

For a multilayer layout, the graph model proposed in Section 3, and the weight calculation scheme given by equation (7) can be applied for each layer independently. However, the main difference here is in modeling interlayer connections. Assume that grid cells and 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 and . 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 equation (7) for this purpose as follows:

$$ei \quad t(e_{\ i}) = (\lambda_{iL} \quad _i + \ ^e) \quad via\_ena\ t \qquad (8)$$

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 $_i$ to model such difference. Furthermore, since via connections are typically undesired, the constant $via\_ena\ t$ is used to avoid using these edges unless they are really necessary.

Also, we can extend these models for routing multiple buses simultaneously. For this, we need to modify the original formulation (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:

$$\begin{aligned} \text{minimize} \quad & \sum_{i \in \mathcal{N}} L_i \\ \text{subject to:} \quad & \\ & \forall i, \quad L_i \leq T_i \\ & \forall i, \quad L_i + S_i \geq T_i \end{aligned} \qquad (9)$$

where T_i and T_i are the upper and lower bounds for net i. Note that such a modification in constraints would only effect the update schedule of Lagrangian multipliers. Rest of the models and algorithms given for the single bus case can be used without a change.

6. EXPERIMENTAL RESULTS

In this section, we have compared our scheme with a commonly used greedy approach. For this purpose, we have written a program that performs routing (with no extra grid allocation) first, and then greedily performs snaking for each net starting from the shortest one in postprocessing.

We have implemented all these algorithms in C++, and we have performed our experiments on an Intel Pentium 4 2.4Ghz system with 512MB memory, and a Linux operating system.

Figure 9 illustrates a sample outcome of our algorithm[6]. Here, there are various nets that are routed with *almost* the same path length. Specifically, we have set the constant Δ in objective function (1) to 1 in our experiments[7]. In accordance with this constraint, the difference between the minimum and maximum path lengths is only one grid cell in the solution of Figure 9. Observe that snaking could be performed even in the dense areas of the layout.

Table 1: Properties of test problems

Test Problem	Vertical Spacing avg	Vertical Spacing stdev	Manhattan Dist. avg	Manhattan Dist. stdev	number of layers
B1	3.15	2.91	117	6.63	single
B2	3.22	2.13	115	13.82	single
B3	2.53	1.99	115	13.96	single
B4	2.32	1.78	115	7.30	single
B5	3.97	1.84	116	7.23	two
B6	3.39	2.59	116	14.41	two
B7	2.34	1.39	116	7.19	two
B8	2.27	1.53	116	11.52	two
B9	1.94	1.36	116	5.80	two
B10	1.85	1.15	117	15.91	two

Table 2: Routing results on test problems

Test Problem	GREEDY SNAKING minL	maxL	stdev	time	LR BASED ROUTING minL	maxL	stdev	time
B1	120	142	3.71	1:45	141	142	0.59	4.53
B2	97	163	11.51	0:50	158	159	0.52	3:44
B3	105	159	9.80	2:40	164	165	0.50	6.06
B4	103	146	7.07	1:56	130	131	0.56	12:06
B5	130	131	0.50	0:15	130	131	0.50	0:15
B6	103	149	5.65	0:14	146	147	0.54	0:44
B7	109	131	2.67	0:14	132	133	0.50	2:50
B8	99	141	6.82	0:14	142	143	0.50	5:27
B9	107	128	3.64	0:14	132	133	0.61	8:49
B10	93	149	9.64	0:14	152	153	0.50	11:50

We have also performed experiments on test problems properties of which are summarized in Table 1. 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 80-100 nets, and the objective is to route them and match their lengths. Note also that, the underlying grid sizes are between 1 0 00 and 1 0 00, depending on the problem size.

We have executed both the greedy algorithm mentioned before and the Lagrangian relaxation (LR) based routing algorithm on these test problems. The comparison of the results are given in Table 2. Here, *minL, maxL, 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. Also, the execution times of these algorithms are given under columns *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 (see the algorithm given in Figure 3) to effectively find the solution that satisfies length constraints. Due to these multiple iterations, the execution time increases; nevertheless the appropriate solution is obtained eventually.

[6] 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 postprocessing.

[7] The length of a route can be extended only by an even number of grid cells. So, 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 1 grid cell difference.

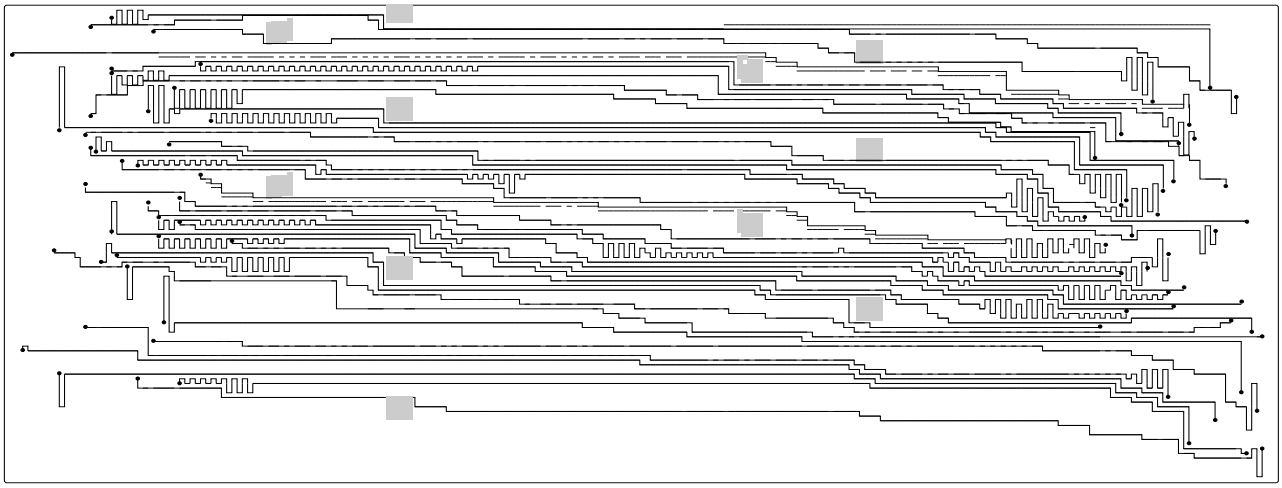

Figure 9: A sample routing solution using Lagrangian relaxation based resource allocation.

7. CONCLUSIONS

We have proposed an algorithm for routing nets within minimum and maximum length bounds. The focus of the paper is mainly on the application of length matching for a group of nets belonging to a bus structure. However, we have shown that it is straightforward to extend the proposed model for more general problems. Our experiments have indicated 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.

8. REFERENCES

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

[2] V. Betz and J. Rose. Vpr: A new packing, placement and routing tool for FPGA research. In *Proc. of the 7th International Workshop on Field-Programmable Logic*, pages 213–222, 1997.

[3] K. D. Boese, J. Cong, A. B. Kahng, K. S. Leung, and D. Zhou. On highspeed VLSI interconnects: Analysis and design. In *Proc. Asia–Pacific Conf. Circuits Syst.*, pages 35–40, 1992.

[4] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Provably good performance-driven global routing. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 11(6):739–752, 1992.

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

[6] 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. of Design Automation Conference*, pages 133–136, 1984.

[7] C. Ebeling, L. McMurchie, S. A. Hauck, and S. Burns. Placement and routing tools for the triptych FPGA. *IEEE Trans. on VLSI*, pages 473–482, 1995.

[8] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[9] M. L. Fisher. An applications oriented guide to Lagrangian relaxation. *Interfaces*, 15(2):10–21, 1985.

[10] A. M. Geoffrion. Lagrangian relaxation and its uses in integer programming. *Mathematical Programming*, 2:82–114, 1974.

[11] X. Guan, P. B. Luh, and L. Zhang. Nonlinear approximation method in Lagrangian relaxation-based algorithms for hydrothermal scheduling. *IEEE Transactions on Power Systems*, 10(2):772–778, 1995.

[12] X. H. Guan, Q. Z. Zhai, and F. Lai. New Lagrangian relaxation based algorithm for resource scheduling with homogeneous subproblems. *Journal of Optimization Theory and Applications*, 113(1):65–82, 2002.

[13] M. H. Held, P. Wolfe, and H. D. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974.

[14] E. Kuh, M. A. B. Jackson, and M. Marek-Sadowska. Timing-driven routing for building block layout. In *Proc. IEEE Intl. Symposium on Circuits and Systems*, pages 518–519, 1987.

[15] S. Prastjutrakul and W. J. Kubitz. A timing-driven global router for custom chip design. In *Proc. of IEEE Intl. Conf. on Computer-Aided Design*, pages 48–51, 1990.

[16] A. Renaud. Daily generation management at electricite de france: Form planning toward real time. *IEEE Transactions on Automatic Control*, 38(7):1080–1093, 1993.

[17] L. W. Ritchey. Busses: What are they and how do they work? *Printed Circuit Design Magazine*, 2000.

[18] 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. on Power Systems*, 10(3):1294–1301, 1995.