Timing Driven Routing Tree Construction

Peishan Tu, Wing-Kai Chow, Evangeline F. Y. Young
Department of Computer Science and Engineering,
The Chinese University of Hong Kong, NT, Hong Kong

ABSTRACT

As technology scales down quickly, timing becomes more and more critical in modern designs. During placement and routing, a lot of techniques are applied to reduce circuit delay. A good timing driven routing tree construction can influence placement and routing significantly. As circuits become more and more complex, previous algorithms may not be efficient enough to be applied in modern designs. In this paper, we propose a new algorithm to construct timing-driven routing trees to trade off wirelength and timing. Our tree construction is based on properties of shortest path tree and minimum spanning tree. Experimental results show that our algorithm can obtain a smaller delay while keeping the wirelength short.

KEYWORDS

Timing Driven Routing Tree

ACM Reference format:
Peishan Tu, Wing-Kai Chow, Evangeline F. Y. Young. 2017. Timing Driven Routing Tree Construction. In *Proceedings of System Level Interconnect Prediction, Austin, TX, USA, June 17 2017 (SLIP17),* 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Previous Work

Timing is always a critical issue in the circuit design. Alpert *et al.* [2] surveyed several existing steiner tree construction methods considering timing and compared their performance of trade-off between total tree length and path length. Prim-Dijkstra algorithm [1], also called PD algorithm, considers tree edge weight as the combination of tree edge length and tree path length from a source node to an edge end node. PD algorithm achieves a better trade-off between total tree length and path length than others [2]. However, sometimes a bit larger edge weight with small edge length is also a good choice of PD algorithm such that the total tree length is reduced while the path length is not increased a lot. Bounded Radius Bounded Cost routing tree (BRBC) algorithm [5] iteratively connects a node, which violates path length constraint, to the source node directly by its shortest path and the work [8] performs an additional relaxation step to further improve the performance. It can produce a tree with the longest path length at most $(1 + \epsilon)$ times that of the shortest path tree and the total tree length at most $(1 + 2/\epsilon)$ times that of minimum spanning tree. All these algorithms above are based on geometry information, such as total tree length and path length, to build a timing driven routing tree. Besides, some consider Elmore delay model into their algorithms to construct a tree. Elmore Routing Tree (ERT) algorithm [3] iteratively picks a node that minimizes the maximum Elmore delay among all sinks. The work [2] shows that the actual total tree length of the tree produced by ERT algorithm is terribly large, which may not be suitable in practice. Moreover, it depends on extra timing information, such as resistance and capacitance. Rudolf [12] proposed a new delay bounded tree construction method which can obtain a tree with relative smaller delay. Given a shortest path tree, it traverses the tree to check whether a node violates capacitance constraint and connect a node in the subtree directly to the root, which has shortest path length from the source. Besides, such tree is also applied in the RC aware router [11]. Although delay is improved, it does not consider total tree length which is unfavourable in actual physical design. Another kind of algorithms tries to solve the minimum rectilinear steiner arborescence (MRSA) problem and any paths connected the source to a sink should be the shortest path. A-tree algorithm [6] solves such problem by growing or combining subtrees to build the arborescence. But the total tree length cannot be guaranteed. Stephan *et al.* [7] proposed an algorithm to construct a tree with delay bounded by $(1 + \epsilon) \cdot rat(v)$, where $rat(v)$ denotes required arrive time, and total tree length bounded by $(2 + \lceil log(\frac{2}{\epsilon}) \rceil)$ times that of its initial tree, which may be a minimum steiner tree with simple modification.

1.2 Our Contributions

In this paper, we present an algorithm to obtain a routing tree that can balance total tree length and path length from the root to each node. In all, the contributions of our work are:

(1) A graph with a significantly smaller number of edges is proposed in which the optimal solution of our problem still exists in it.
(2) Two graphs, upper bound graph UG and lower bound graph LG, are proposed. They own good properties of the path length for each node.
(3) An efficient algorithm called UGLG algorithm is proposed to build a tree that can balance tree length and path length based on UG and LG.
(4) A batch algorithm is shown to further improve the performance of UGLG algorithm such that the total tree length is further reduced while delay keeps small.
(5) We analyze different algorithms in the experiments and show that our algorithm can achieve a better trade-off between total tree length and maximum delay. The batch algorithm is also compared.

The remainder of the paper is organized as follows: Section 2 is about background. Our algorithm is described in section 3. In section 4, we introduce a batch algorithm. Abundant experimental results are shown in section 5. Then we conclude our contribution in section 6.

SLIP17, Austin, TX, USA
2017. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

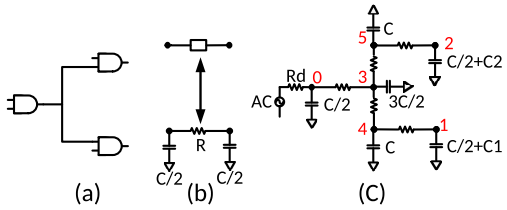

Figure 1: Elmore Delay Model. (a) A net with a driver and two sinks. (b) The wire is modeled by π type distribute RC delay model. (c) The RC delay model of (a). Sink 1 and 2 has a loading capacitance $C1$ and $C2$ respectively.

2 BACKGROUND

Given a net with sink $\{v_0, v_1, \ldots, v_i, \ldots, v_{|V|-1}\}$ and a source s with fixed positions, we need to construct a tree considering timing information. The distance between nodes is measured as $\ell_1 norm$.

2.1 RC Delay Model

Given a circuit shown in figure 1 (a), the wire is modeled by the π model as shown in figure 1 (b). The driver node 0 is modeled by a driver resistance R_d and the lumped downstream capacitance. The estimation of delay in our experiments is based on the π model and the Elmore Delay model [10].

2.2 Problem Formulation

The analysis of the RC delay model [3] shows that the total tree length and the path length of a sink i are related to the delay of sink i.

A graph $G(V, E)$ consists of $|V|-1$ sinks and a source s. Any node $i \in V$ and $j \in V$ are connected. Given a user defined parameter α ($\alpha \geq 0$), a tree T with root s is constructed on G such that:

$$\begin{aligned} \text{minimize} \quad & \sum_{e_{ij} \in T} w_{ij} \\ & l_{si} <= (1+\alpha) \cdot D_i \quad \forall i \in N \end{aligned} \tag{1}$$

where e_{ij} is the edge between node i and node j and w_{ij} is the edge length of e_{ij}. l_{si} denotes the path length from s to sink i in T and D_i denotes the shortest path length from s to sink i in $G(V, E)$. A tree T is constructed such that the path length l_{si} of each sink i is bounded by $(1+\alpha)$ times the length of the shortest path from s to node i in $G(V, E)$ and the total tree length is minimized.

3 UGLG ALGORITHM

3.1 Edge Reduced Graph *ERG*

Definition 3.1. Edge Reduced Graph $ERG(V, E)$ Given a set of points V in the (R^2, ℓ_1) space, consider two points $i \in V$ and $j \in V$ with $x_i \leq x_j$. There exists an edge $e_{ij} \in E$ if and only if there is no point k at (x_k, y_k) such that $x_i \leq x_k \leq x_j$ and $y_i \leq y_k \leq y_j$ or $y_j \leq y_k \leq y_i$.

Theorem 3.2. *Given a set of points V in the (R^2, ℓ_1) space, there exists an optimal solution solu∗ of problem (1) in the corresponding Edge Reduced Graph $ERG(V, E)$.*

$ERG(V, E)$ is an undirected graph and the proof of theorem 3.2 is shown in the appendix. $ERG(V, E)$ can be constructed as follows:

$ERG(V, E)$ consists of $|V|$ points, which are sorted in ascending order of x. For each node i, a line is swept from it to the rightmost. There are two markers y^i_{max} and y^i_{min} of node i with initial value of $+\infty$ and $-\infty$. An edge e_{ij} will be added to E if $y^i_{min} < y_j < y^i_{max}$. Meanwhile y^i_{max} and y^i_{min} are updated according to equation 2.

$$\begin{aligned} y^i_{max} &= \max\{\min\{y^i_{max}, y_j\}, y_i\} \\ y^i_{min} &= \min\{\max\{y^i_{min}, y_j\}, y_i\} \end{aligned} \tag{2}$$

3.2 Upper Bound Graph UG and Lower Bound Graph LG

Definition 3.3. Lower Bound Graph $LG(V, E')$ Given a directed graph $G(V, E)$, a source node s and a parameter α, consider an edge e_{pq} with weight w_{pq} in $G(V, E)$. The lower bound graph $LG(V, E')$ is constructed such that the edge $e_{pq} \in E'$ iff e_{pq} satisfies

$$D_p + w_{pq} \leq (1+\alpha) \cdot D_q \tag{3}$$

where D_p and D_q are the shortest path length from s to p and to q in $G(V, E)$ respectively.

Definition 3.4. Upper Bound Graph $UG(V, E^*)$ Given a directed graph $G(V, E)$, a source node s and a parameter α, consider an edge e_{pq} with weight w_{pq} in $G(V, E)$. The upper bound graph $UG(V, E^*)$ is constructed such that the edge $e_{pq} \in E^*$ iff e_{pq} satisfies

$$(1+\alpha) \cdot D_p + w_{pq} \leq (1+\alpha) \cdot D_q \tag{4}$$

where D_p and D_q are the shortest path length from s to p and to q in $G(V, E)$ respectively.

According to the definition 3.3, we can conclude that the optimal solution of our problem must be in LG because LG only excludes edges whose existence will violate path length constraints. However, an MST in LG may not be a legal solution. In UG, all solutions are legal but the total tree length of an MST in UG will be always at least that of an MST in LG. According to these properties, our strategy is to build a legal MST in UG first and take advantages of the edges in LG to further optimize the solution.

In our method, an Edge Reduced Graph $ERG(V, E)$ is constructed first. According to the definitions 3.3 and 3.4, shortest path length D_i ($\forall i \in N$) could be obtained by the shortest path algorithm and two graphs LG and UG can then be built. Since all possible trees in UG are legal, we will choose to start with an MST in UG. However, the total tree length of the MST in UG may be very much larger than that of the optimal solution in ERG. The reason is that, due to the constraint in definition 3.4, UG may exclude some edges e_{pq} of small weight if the difference between the shortest path lengths of p and q is small. We thus need to add back those edges with small weight in LG to our tree to further optimize the total tree length. The advantage of using LG is that the number edges in LG are much lower than that in ERG. However, some of these edge adding steps will result in an illegal tree. If an edge is to be added, the path length at all the sinks should still be legal after the addition. Some edge will be needed to be removed from our tree after addition such that our solution can still be a tree. We handle edges in LG in a non-decreasing order of their edge weights.

The framework of our algorithm is shown in algorithm 1.

Algorithm 1 Constructing A Timing Driven Routing Tree

Input: A source node s and a set of sinks $i \in V$
Output: A timing driven routing tree T'

```
Obtain edge reduced graph ERG(V, E)
Get shortest path length d_i for each sink i ∈ V
Obtain upper bound graph UG and lower bound graph LG
Get a minimum spanning tree T_M_UG on UG
Sort the edges in LG in non-decreasing order
Initialize our data structure
for e ∈ edges in LG do
    Try to add the edge e to T_M_UG
end for
T' ← T_M_UG
```

3.3 Data Structure

When an edge e_{pq} is being added, we need to check whether the path length at all the sinks in the subtree of node q are legal. A straightforward way is to visit all the nodes in the subtree once, which is time consuming. To reduce running time, a data structure is designed to keep information showing how tight the current path length of a node is. This slack information indicates how much the path length of node i can be increased before exceeding the bound.

At each node i, C_i, δ_i and L_i (a list of S_k^i where node k is in the subtree of node i) are stored. C_i denotes the current distance from the source s to node i. δ_i denotes the difference between C_i and O_i as shown in equation 5, where O_i is the origin distance from s to node i in T_{M_UG}, an MST in UG.

$$C_i + \delta_i = O_i \tag{5}$$

The term S_k^i denotes the slack information of node k in the subtree

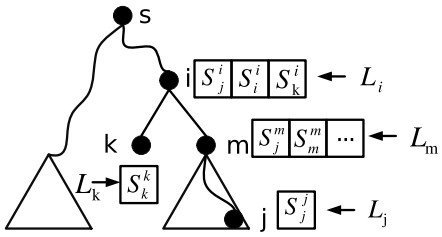

Figure 2: Slack Information

of node i. The slack information and L_i of node i are maintained as shown in figure 2. First, at node k, S_k^k is calculated by equation 6. If node k is a leaf, there is only one slack information in L_k. The slack information of node k should be passed to its parent node i and the slack information S_k^i of node k at node i is calculated by equation 7. For an internal node m, the number of slack information stored in L_m is equal to its number of children and the slack information in L_m is stored in a non-decreasing order. Only the smallest slack information S_j^m in L_m will be passed to its parent node i as S_j^i. After traversing the tree, L_i of each node i in the tree is built.

$$(1+\alpha)\cdot D_k = S_k^k + O_k = S_k^k + C_k + \delta_k \tag{6}$$

$$(1+\alpha)\cdot D_k = S_k^i + O_i - l_{ik} = S_k^i + C_i - l_{ik} + \delta_i = S_k^i + C_k + \delta_i \tag{7}$$

When we modify the tree structure, slack information of nodes should be added and deleted accordingly. The detail of modification of the slack information will be shown later.

3.4 Edge Adding Technique

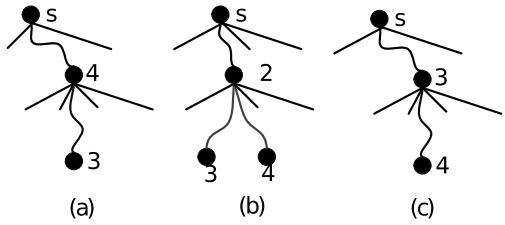

Figure 3: Examples of the possible relationships of node 3 and node 4

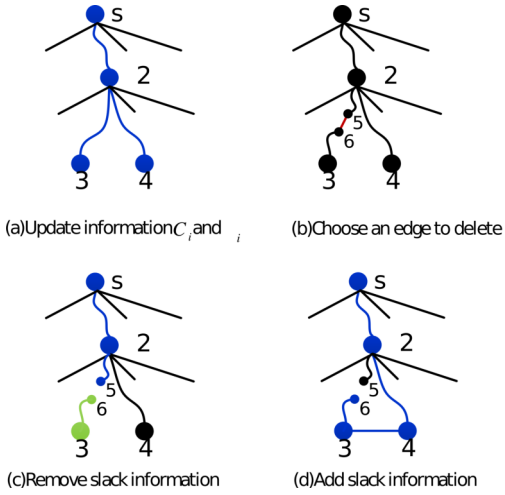

Figure 4: Examples of adding edge e_{43}

Consider a directed edge e_{43} from node 4 to node 3 and the possible relationships of these two nodes are shown in figure 3. The edge e_{43} cannot be added to the tree because node 3 is the ancestor of node 4 as shown in figure 3 (c). Otherwise, lazy update of C_i and δ_i is applied on each node i on two paths from s to node 3 and from s to node 4 (denoted in blue in figure 4 (a)). Next an edge e_{56} denote in red in the path from node 3 to node 2 will be chosen to be deleted (figure 4 (b)). Edge e_{56} is the edge with the largest edge length in the path from node 3 to node 2 such that there is no violation on the path length constraints after deletion. Then the smallest slack information S_k^6 in L_6 should be removed from L_i of node i on the blue path in figure 4 (c), if it exists in L_i (as S_k^i). Meanwhile, the smallest slack information in L_3 should also be removed from the L_i of node i on the green path in figure 4 (c). The last step is to add smallest slack information in L_6 to L_i of node i on the blue path of figure 4 (d). The tree structure is also modified simultaneously.

The whole process is summarized in algorithm 2.

Algorithm 2 Add Edge e_{pq} to Tree T

Input: an edge e_{pq} with weight w_{pq}, a parameter α, a tree T with a root s
Output: a tree T'

```
if Safe checking then
    Update C_i and δ_i of node i in two paths from s to p and q
    if Select an edge e_uv to delete then
        Remove slack information
        Add slack information
    end if
end if
T' ← T
```

3.4.1 Safe Checking. Given an edge e_{pq}, this step checks to ensure that node q is not the ancestor of node p.

3.4.2 Update Distance Information. Except for the first addition and deletion, distance information C_i and δ_i of node i may not be accurate because such information is not updated after modification of the tree structure. Hence, it will influence calculation on bound value of constraints. Hence, information C_i and δ_i of node i should be updated before adding or deleting edges. C_i of node i on the path from s to node p and s to node q is updated and δ_i is handled simultaneously by equation 5.

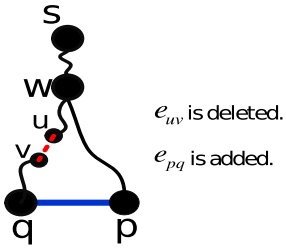

Figure 5: Illustration on UGLG algorithm

3.4.3 Select an edge e_{uv} to delete. As shown in figure 5, assume node w is the lowest common ancestor of node p and node q. An edge e_{uv} on the path from node q to node w is selected to be deleted by satisfying two conditions. Condition 1 is that all the path lengths of the nodes in tree T should remain legal after adding e_{pq} and deleting e_{uv}. If inequality 10 holds for node v, no path length constraint is violated, where new_v denotes the new path length from s to node v by adding edge e_{pq}. Assume S_k^v is the first value in L_v which means the gap between new path length and the bound value of node k is the smallest in the subtree of node v. Inequality 8 should hold where new_k is the new path length from s to node k. By applying equation 7, inequality 9 and inequality 10 can be obtained. The other condition is that the edge length of e_{uv} is maximum among all the edges satisfying condition 1 on the path from p to w.

$$(1+\alpha)D_k > new_k = new_v - l_{vk} \tag{8}$$

$$S_k^v + C_v + \delta_v - l_{vk} > new_v - l_{vk} \tag{9}$$

$$S_k^v + C_v + \delta_v > new_v \tag{10}$$

If there is no such edge satisfying both conditions, e_{pq} will not be added and another edge is considered to be added.

3.4.4 Remove Slack Information. As shown in figure 5, when e_{uv} is deleted from the tree, the smallest slack information S_k^v should be removed from L_i of node i on the path from u to s, if S_k^i exists. Similarly, the smallest slack information S_m^q in L_q should be deleted from L_i of node i on the path from q to v (q excluded).

3.4.5 Add Slack Information. As shown in figure 5, the first slack information S_k^i of each child node i on the path from node v to s should be passed to its parent node p as S_k^p by equation 11.

$$S_k^i + \delta_i = S_k^p + \delta_p \tag{11}$$

3.5 Complexity

In algorithm 1, the construction of ERG requires $O(|V|^2)$ and the graph UG and LG is built in $O(|V| + |E|)$. The shortest path tree algorithm and the minimum spanning tree algorithms requires $O(|E| + |V|log|V|)$. Sorting takes $O(|V|log|V|)$. Traversing all the edges is in $O(|E|)$. Besides, modification on L_i takes $O(logd)$, where d denotes the degree of the tree. Considering adding an edge, we should trace a path in $O(logN)$ and make modification on L_i. Running time of algorithm 2 is in $O(|E|log|V|logd)$. Taking all into consideration, our algorithm takes $max\{O(N^2), O(|E|log|V|logd)\}$.

3.6 Rectilinearization

In order to further improve wirelength, we apply the technique of removal overlapping edges [1]. It compares each pair of adjacent edges and estimates a reduced cost according to their bounding box. The pairs maximum cost reduction will be processed to remove overlapped edges.

4 BATCH ALGORITHM

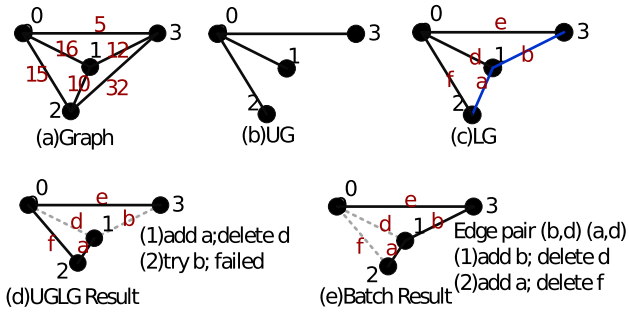

Figure 6: An Example

The UGLG algorithm only considers to add the smallest weight edge each time. However, adjusting the order of edges may reduce more total tree length and a case that illustrate such situation is shown in figure 6 (a), where red numbers denote edge length. Besides, α is 1 for the case. An MST of UG is obtained in figure 6 (b) and edges denoted in blue in figure 6 (c) can be added to the MST. UGLG algorithm can produce a tree shown in figure 6 (d). However,

if we try to add the edge b first and the edge a next, the corresponding result shown in figure 6(e) will have a smaller total tree length. In addition, more iterations may also bring a better solution because some edges may fail to add but these edges can be added after tree structure is modified. In order to produce a better solution, a batch algorithm is developed.

Given a tree T, first pairs of edges (e_a, e_d) are obtained by the UGLG algorithm, where e_a is considered to be added and e_d is to be deleted. The pairs are sorted by *pair_weight*, which is defined by

$$pair\_weight = \Delta w - \Delta path \tag{12}$$

where Δw denotes the gain in the total tree length $|e_d| - |e_a|$ and $\Delta path$ denotes the increase in path length. After sorting, the edge is added according to the new order based on UGLG algorithm. Noted that when we add the edge, the corresponding edge to be deleted is decided by UGLG algorithm and it may not be as same as the one we stored in the pair. We repeat such process until there is no more edges to be added or it reaches the maximum iteration time.

Other factors, such as capacitance, can be taken into consideration in the *pair_weight* in order to further optimize the problem.

5 EXPERIMENTAL RESULTS

5.1 Experiments with Different Algorithms

Table 1: Benchmark Information

# pins	sb18	sb16	sb4	sb10	sb1	sb3	sb5	sb7
[0, 10)	730495	969721	772680	1842288	1174480	1167280	1069712	1831245
[10, 20)	24472	17228	16855	31289	23310	34991	18163	62510
[20, 30)	10887	7327	8724	13826	11180	15447	7624	27485
[30, 40)	5060	5348	3755	9495	5842	6131	4671	11038
[40, 50)	619	264	485	1201	879	1095	625	1641
[50, ∞)	9	14	14	20	19	35	30	26
total	771542	999902	802513	1898119	1215710	1224979	1100825	1933945

ICCAD2015 [9] provides timing driven designs and their net information is shown in table 1. Nets are classified by their pin number and number of nets in each category is displayed in the table. Meanwhile, placement results, with respect to short displacement constraint, produced by the winning 1st placer in the contest are used. Besides, we set driver resistance 4ohm. In the experiments, several algorithms, such as PD [1], FLUTE [4], BONN [12] and ours are compared. Except FLUTE, others are run several times using different parameters. As shown in figure 7, wirelength and maximum delay are normalized by the result of FLUTE. The result of FLUTE is the point denoted as a green circle. Bonn is implemented by us and same parameters in [12] are selected in the experiments. It can always produce a result with relatively small delay but will lose in wirelength. Results produced by PD algorithm with parameters $c \in [0, 1]$ can achieve a good trade off between wirelength and maximum delay. However, our algorithm can minimize total wirelength further compared with PD. In order to show the effect of the algorithms, a ratio r is defined to estimate how well the result is, which is defined by $r = \frac{\text{improvement of delay}}{\text{loss of wirelength}}$. The results with best maximum delay of each algorithm on each benchmark are selected, which are shown in table 2. Bonn can achieve 9.39% gain on maximum delay but lose 138.36% wirelength on average. Maximum delay of PD is 3.48% of FLUTE and the loss of wirelength is 20.37%. Ours is 4.48% and −14.67% respectively. Considering ratio, average ratio of ours is 0.3 and meanwhile PD is 0.175. Ours is around 4s slower.

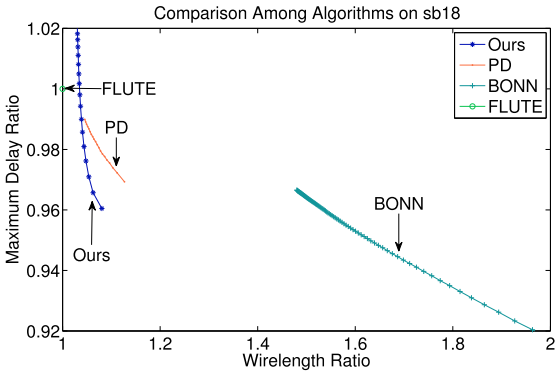

Figure 7: Comparision Among Algorithms

5.2 Comparison with Removal Overlapping Technique

We also compare PD and ours by using the overlapping removal technique, called PD-Steiner and UGLG-steiner respectively. The results are shown in table 3. PD improves 0.319 with respect of r and our improvement is 0.56 on r.

5.3 Comparison with Batch Algorithm

Batch algorithm are also performed by using parameters $\alpha \in (0, 1]$ on ICCAD2015 benchmarks. The default number of iteration is 5. Shown in figure 8, the results of the batch algorithms and the UGLG algorithm on different benchmarks are shown. We can see from the figure that the batch algorithm can achieve a little bit better result compared with the UGLG algorithm. The runtime is increased by around $1 - 2$ s for these benchmarks.

In figure 9, the proposed techniques, removal overlapping and batch algorithm, are compared.The results of 8 benchmarks with different techniques are shown in the figure. We pick results with smallest maximum delay value among different parameters by applying different techniques. Maximum delay ratio and wirelength ratio are estimated based on results of FLUTE. Batch algorithm improves a little wirelength and overlapping removal technique reduces wirelength a lot.

6 CONCLUSION

Optimizing timing can improve the performance of current physical design. Based on the RC delay model, geometry relationship of the circuit is used to model timing. The optimization objective is to minimize the total wirelength while each path length is within a certain range. After *ERG* is constructed, UGLG algorithm obtains an MST in *UG*. It iteratively adds an edge to *MST* to optimize the solution. A batch algorithm is also proposed to further improve the performance. The experimental results show that our algorithm can have better performance.

Table 2: Comparison Among Algorithms

FLUTE											
sb18	8.32	0.00%	6.97	0.00%	9.23	0.00%	5.77E+07	0.00%	18.19	0.00%	-
sb16	10.75	0.00%	9.86	0.00%	11.42	0.00%	9.36E+07	0.00%	18.14	0.00%	-
sb4	7.99	0.00%	6.74	0.00%	8.84	0.00%	7.16E+07	0.00%	16.60	0.00%	-
sb10	14.02	0.00%	12.89	0.00%	14.88	0.00%	2.06E+08	0.00%	34.26	0.00%	-
sb1	7.12	0.00%	5.96	0.00%	7.98	0.00%	9.61E+07	0.00%	26.23	0.00%	-
sb3	8.57	0.00%	7.09	0.00%	9.59	0.00%	1.15E+08	0.00%	28.30	0.00%	-
sb5	10.27	0.00%	8.30	0.00%	11.92	0.00%	1.08E+08	0.00%	22.26	0.00%	-
sb7	6.45	0.00%	5.13	0.00%	7.26	0.00%	1.40E+08	0.00%	42.82	0.00%	-
Average	9.18	0.00%	7.87	0.00%	10.14	0.00%	1.11E+08	0.00%	25.85	0.00%	-
PD											
Benchmarks	AD	imprv.	MIND	imprv.	MAXD	imprv.	WL	imprv.	Runtime	imprv.	r
sb18	7.79	6.43%	6.453	7.38%	8.783	4.80%	7.65E+07	-32.56%	9.86814	45.76%	0.147
sb16	10.4	3.21%	9.487	3.83%	11.17	2.18%	1.04E+08	-11.56%	11.5992	36.06%	0.189
sb4	7.56	5.29%	6.347	5.89%	8.514	3.74%	8.45E+07	-18.08%	9.69289	41.61%	0.207
sb10	13.7	2.54%	12.56	2.63%	14.6	1.85%	2.32E+08	-12.76%	20.843	39.17%	0.145
sb1	6.77	4.86%	5.679	4.70%	7.694	3.61%	1.14E+08	-18.50%	16.0013	38.99%	0.195
sb3	8.05	6.07%	6.564	7.44%	9.195	4.10%	1.43E+08	-24.56%	16.0357	43.34%	0.167
sb5	9.8	4.54%	7.843	5.55%	11.57	2.93%	1.24E+08	-15.08%	14.6127	34.37%	0.194
sb7	6	6.90%	4.666	9.07%	6.928	4.60%	1.82E+08	-29.83%	26.149	38.93%	0.154
Average	8.76	4.98%	7.45	5.81%	9.81	3.48%	1.32E+08	-20.37%	15.60	39.78%	0.175
BONN											
Benchmarks	AD	imprv.	MIND	imprv.	MAXD	imprv.	WL	imprv.	Runtime	imprv.	r
sb18	6.84	17.82%	5.864	15.82%	7.962	13.70%	1.81E+08	-214.36%	21.429	-17.79%	0.064
sb16	9.82	8.65%	8.875	10.03%	10.86	4.87%	1.69E+08	-80.22%	22.4913	-23.99%	0.061
sb4	6.86	14.10%	5.792	14.13%	7.994	9.62%	1.59E+08	-122.63%	20.3612	-22.66%	0.078
sb10	13.1	6.63%	12.03	6.68%	14.2	4.55%	3.80E+08	-84.72%	40.9977	-19.66%	0.054
sb1	6.26	12.10%	5.311	10.87%	7.303	8.50%	2.17E+08	-126.18%	31.6514	-20.68%	0.067
sb3	7.16	16.47%	5.983	15.62%	8.466	11.71%	3.02E+08	-164.13%	34.42	-21.63%	0.071
sb5	9.02	12.21%	7.075	14.80%	11.08	7.01%	2.14E+08	-98.45%	27.1845	-22.10%	0.071
sb7	5.09	21.06%	4.087	20.37%	6.162	15.15%	4.44E+08	-216.19%	52.0292	-21.52%	0.070
Average	8.02	13.63%	6.88	13.54%	9.25	9.39%	2.58E+08	-138.36%	31.32	-21.25%	0.067
OURS											
Benchmarks	AD	imprv.	MIND	imprv.	MAXD	imprv.	WL	imprv.	Runtime	imprv.	r
sb18	7.67	7.86%	6.381	8.40%	8.614	6.63%	7.08E+07	-22.78%	12.8826	-30.55%	0.291
sb16	10.4	3.20%	9.508	3.62%	11.15	2.34%	1.02E+08	-9.05%	14.5204	-25.18%	0.259
sb4	7.51	5.92%	6.33	6.14%	8.427	4.72%	8.10E+07	-13.16%	12.2056	-25.92%	0.359
sb10	13.6	2.70%	12.56	2.61%	14.56	2.19%	2.25E+08	-9.46%	26.4404	-26.86%	0.231
sb1	6.76	5.06%	5.69	4.51%	7.651	4.14%	1.09E+08	-13.36%	19.0409	-19.00%	0.310
sb3	7.95	7.24%	6.531	7.89%	9.031	5.81%	1.34E+08	-17.35%	20.5394	-28.09%	0.335
sb5	9.8	4.61%	7.857	5.38%	11.53	3.27%	1.20E+08	-11.26%	17.8537	-22.18%	0.290
sb7	5.89	8.59%	4.624	9.91%	6.773	6.74%	1.70E+08	-20.91%	33.0358	-26.34%	0.322
Average	8.70	5.65%	7.43	6.06%	9.72	4.48%	1.26E+08	-14.67%	19.56	-25.51%	0.300

Note:AD, MIND and MAXD denote average delay, minimum delay and maximum delay. The unit of delay is ps and the unit of wirelength (WL) is um. The unit of running time is second.

REFERENCES

[1] C. J. Alpert, T. Hu, J. Huang, A. B. Kahng, and D. Karger. Prim-dijkstra tradeoffs for improved performance-driven routing tree design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(7):890–896, 1995.

[2] C. J. Alpert, A. B. Kahng, C. Sze, and Q. Wang. Timing-driven steiner trees are (practically) free. In *Proceedings of the 43rd annual Design Automation Conference*, pages 389–392. ACM, 2006.

[3] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins. Near-optimal critical sink routing tree constructions. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(12):1417–1436, 1995.

[4] C. Chu and Y.-C. Wong. Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(1):70–83, 2008.

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

[6] J. Cong, K.-S. Leung, and D. Zhou. Performance-driven interconnect design based on distributed rc delay model. In *Design Automation, 1993. 30th Conference on*, pages 606–611. IEEE, 1993.

Table 3: Overlapping Removal

PD-Steiner											
Benchmarks	AD	imprv.	MIND	imprv.	MAXD	imprv.	WL	imprv.	Runtime	imprv.	r
sb18	8.08	2.90%	6.93	0.45%	8.94	3.07%	6.50E+07	-12.69%	12.03	33.89%	0.242
sb16	10.63	1.06%	9.84	0.27%	11.29	1.13%	9.66E+07	-3.23%	13.17	27.40%	0.351
sb4	7.81	2.19%	6.72	0.31%	8.64	2.28%	7.60E+07	-6.18%	11.88	28.43%	0.369
sb10	13.86	1.08%	12.87	0.21%	14.70	1.22%	2.14E+08	-4.18%	25.29	26.20%	0.292
sb1	6.94	2.49%	5.92	0.65%	7.77	2.60%	1.02E+08	-6.68%	19.62	25.21%	0.390
sb3	8.34	2.68%	7.05	0.62%	9.33	2.71%	1.25E+08	-9.11%	20.52	27.48%	0.297
sb5	10.09	1.73%	8.27	0.36%	11.72	1.68%	1.13E+08	-4.76%	15.57	30.05%	0.352
sb7	6.27	2.76%	5.11	0.35%	7.06	2.79%	1.56E+08	-10.85%	28.91	32.47%	0.257
Average	9.00	2.11%	7.84	0.40%	9.93	2.18%	1.18E+08	-7.21%	18.37	28.89%	0.319
OURS-Steiner											
Benchmarks	AD	imprv.	MIND	imprv.	MAXD	imprv.	WL	imprv.	Runtime	imprv.	r
sb18	8.01	3.80%	6.86	1.57%	8.86	3.95%	6.23E+07	-8.06%	14.78	18.73%	0.490
sb16	10.63	1.08%	9.83	0.35%	11.29	1.13%	9.57E+07	-2.24%	15.28	15.77%	0.504
sb4	7.78	2.58%	6.69	0.81%	8.61	2.67%	7.43E+07	-3.89%	13.84	16.63%	0.687
sb10	13.86	1.14%	12.85	0.32%	14.69	1.27%	2.11E+08	-2.73%	28.92	15.59%	0.466
sb1	6.93	2.63%	5.91	0.85%	7.76	2.75%	1.00E+08	-4.36%	23.70	9.66%	0.630
sb3	8.28	3.37%	6.99	1.38%	9.26	3.43%	1.21E+08	-5.83%	24.44	13.63%	0.589
sb5	10.08	1.85%	8.25	0.61%	11.71	1.77%	1.11E+08	-3.21%	20.65	7.26%	0.551
sb7	6.21	3.73%	5.06	1.44%	6.99	3.73%	1.50E+08	-6.66%	34.74	18.85%	0.559
Average	8.97	2.52%	7.81	0.92%	9.90	2.59%	1.16E+08	-4.62%	22.04	14.52%	0.560

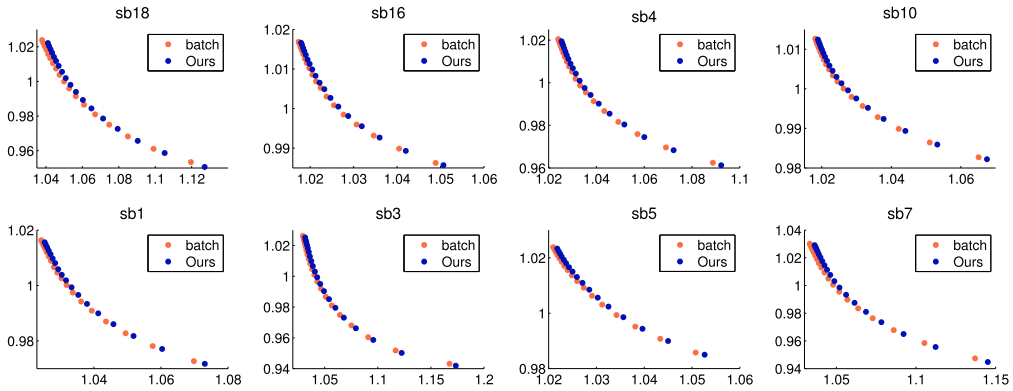

Figure 8: Comparison with Batch Algorithm

[7] S. Held and D. Rotter. Shallow-light steiner arborescences with vertex delays. In *Integer Programming and Combinatorial Optimization*, pages 229–241. Springer, 2013.

[8] S. Khuller, B. Raghavachari, and N. Young. Balancing minimum spanning trees and shortest-path trees. *Algorithmica*, 14(4):305–321, 1995.

[9] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan. Iccad-2015 cad contest in incremental timing-driven placement and benchmark suite. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 921–926. IEEE Press, 2015.

[10] J. Rubinstein, P. Penfield Jr, and M. A. Horowitz. Signal delay in rc tree networks. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 2(3):202–211, 1983.

[11] R. Scheifele. Rc-aware global routing. In *Proceedings of the 35th International Conference on Computer-Aided Design*, page 21. ACM, 2016.

[12] R. Scheifele. Steiner trees with bounded rc-delay. *Algorithmica*, 78(1):86–109, 2017.

APPENDIX

THEOREM .1. *Given a set of points in* (R^2, ℓ_1), *there exists an optimal solution* $solu*$ *of problem* (1) *in the corresponding Edge Reduced Graph ERG.*

PROOF. Denote source node as s. We will show that if edge pq in $solu*$ does not exist in G, we can find alternative edges for pq to obtain a better solution. In $solu*$, the tree edge may have 8 directions shown in figure 10. In the proof, we only consider direction 1 while 3, 5 and 7 are similar and others can be explained by adjacent direction's proof. As a base case, there are four situations shown in figure 10(I)(II)(III)(VI). It shows there is only one node r located inside the bounding box of p and q.

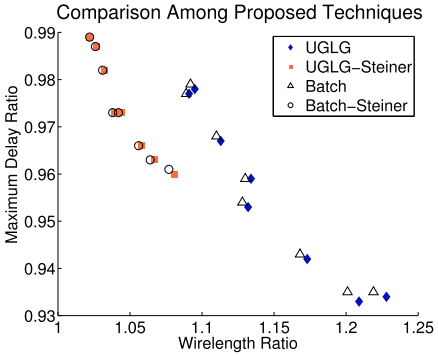

Figure 9: Comparison Among Proposed Techniques

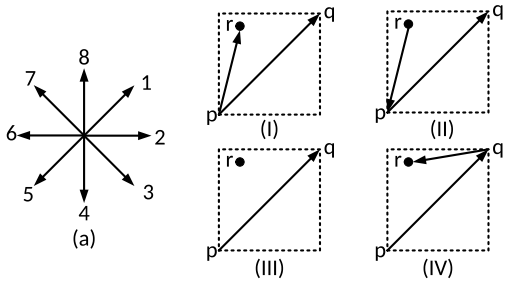

Figure 10: Proof

Base case 1: Replace pq with rq to get $solu'$. l_{sr}, l_{sp} and l_{sq} do not change, where $l'_{sq} = l_{sp} + w_{pr} + w_{rq} = l_{sp} + w_{pq} = l_{sq}$. The total weight is decreased by w_{pr}. Therefore $solu'$ is better.

Base Case 2: Replace pq with rq to get $solu'$. l_{sr} and l_{sp} are same. l_{sq} is smaller, where $l'_{sq} = l_{sr} + w_{rq} < l_{sr} + w_{rp} + w_{pq} = l_{sq}$. The total weight is reduced by w_{pr}. Therefore $solu'$ is better.

Base Case 3: node r has no connection with p and q.

(1) Replace pq with pr and rq to get $solu'$ and delete the incoming edge ir to r in $solu*$. l_{sp} and l_{sq} are same. $l'_{sr} = l_{sp} + w_{pr}$. The total weight is smaller by w_{ir}.
(2) We can also replace pq by rq. l_{sp} and l_{sr} are same. $l'_{sq} = l_{sr} + w_{rq}$. The total weight is smaller by w_{pr}.

If pq is in $solu*$, 1) does not hold and then $l'_{sr} = l_{sp} + w_{pr} > (1+\alpha)D_r \geq l_{sr}$. Moreover, 2) does not hold and $l'_{sq} = l_{sr} + w_{rq} > (1+\alpha)D_q \geq l_{sq} = l_{sp} + w_{pq}$. However, these two condition are contradicted. Because

$$\begin{aligned} l'_{sq} = l_{sr} + w_{rq} &> l_{sq} = l_{sp} + w_{pq} \\ l_{sr} + w_{rq} &> l_{sp} + w_{pr} + w_{rq} \\ l_{sr} &> l_{sp} + w_{pr} \end{aligned} \tag{13}$$

Therefore, either 1) or 2) must hold and the solution is better than $solu*$.

Base case 4: We delete the edge qr and replace pq by pr and rq. l_{sp} and l_{sq} are same. and l_{sr} is smaller because $l'_{sr} = l_{sp} + w_{pr} < l_{sr} = l_{sp} + w_{pq} + w_{qr} = l_{sp} + w_{pr} + 2w_{rq}$. The total weight is smaller by w_{rq}.

In addition, if any nodes located outside the bounding box of p, q form a tree path p to r, r to p or q to r, we can transform such case into base cases.

- path p to r: We can transform it into case 1 by adding an edge pr and delete the incoming edge from r. The total weight is reduced by this deleting edge.
- path r to p: We could add an edge rp and delete edge e which is connected with r in the path r to p. It can be proved in case 2 and the total weight is also reduced.
- path q to r: It can be modified by adding an edge qr and remove the edge connected with r in the path. It is proved in case 4 that a better solution can be found.

Hence we can prove that the edge pq which is not in G can not be in the $solu*$ when there is one node inside bounding box of pq.

Assume there is an edge pq in $solu*$ which is not in the graph G. There are n nodes $v_1, \cdots, v_n$ lying in the bounding box of the p and q. pq can not exist in $solu*$.

When there are $n+1$ nodes $v_1, \cdots, v_{n+1}$ inside the bounding box of the p and q and they are sorted in x ascending order. The edge $pv_1, \cdots, v_iv_{i+1}, \cdots, v_{n+1}q$ must be in the G. We define set $S = \{v_1, \cdots, v_{n+1}, p, q\}$ and set $S' = \{v_1, \cdots, v_{n+1}\}$.

Considering the edge pv_{n+1}, there are less than or equal n nodes inside the bounding box p, v_{n+1}. According to assumption, edge pv_{n+1} cannot in $solu*$. We can also find alternative edges in G to get a better solution.

Case 1: node v_{n+1} has an incoming edge lying in the path p to v_{n+1}. Considering nodes p, q and v_{n+1}, it is proved in base case 1. But it may also introduce the non existing edge pv_{n+1}, which is solved in assumption.

Case 2: There is a path $v_{n+1}p$. Considering nodes p, q and v_{n+1}, it is proved in base case 2. The edge pv_{n+1} may be added and it is explained by assumption.

Case 3: node v_{n+1} has no connection with nodes p, q in the $solu*$. Considering nodes p, q and v_{n+1}, we can prove pq not in $solu*$ according to base case 3. But it may introduce the edge $p_{v_{n+1}}$ which is not in G and edge pv_{n+1} is considered in assumption.

Case 4: A path from node q to node v_{n+1} exists in the $solu*$. Considering nodes p, q and v_{n+1}, base case 4 proved it. And also edge $p_v n + 1$ is explained in assumption.

The proof is completed. □