# 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

June 17, 2016

# Outline

# Outline

# Introduction

Timing driven tree construction in routing:

- As technology scales down, a more effective routing tree construction approach is needed.

Existing works:

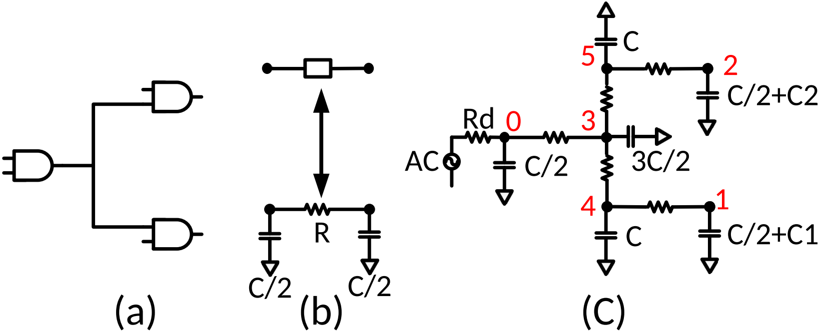
- Path length and total wirelength trade off e.g. PD and BRBC
- Elmore delay considered e.g. ERT algorithm
- Minimum rectilinear steiner arborescence (MRSA) construction

# Our Contributions

- A graph with a significantly smaller number of edges edge reduced graph $ERG$ is proposed.
- Two graphs, upper bound graph $UG$ and lower bound graph $LG$, are proposed.
- An efficient algorithm called UGLG algorithm is proposed.
- A batch algorithm is shown to further improve the performance of UGLG algorithm.
- 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.

# Outline

# RC Delay Model




Elmore Delay Model. (a) a net with a driver and two sinks. (b) We use $\pi$ type distribute RC delay model. (c) The RC delay model of (a).

# Problem Formulation

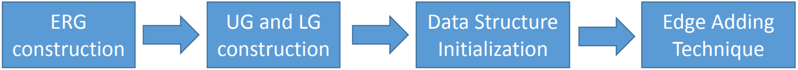
A graph $G(V, E)$ consists of $|V| - 1$ sinks and a source $s$. Any node $i \in V$ and $j \in V$ are connected. Given a user defined parameter $\alpha$ $(\alpha \geq 0)$, a tree $T$ with root $s$ is constructed on $G$ such that:

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

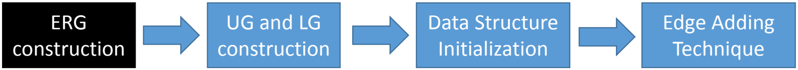
- $e_{ij}$ is the edge between node $i$ and node $j$
- $w_{ij}$ is the edge length of $e_{ij}$
- $l_{si}$ denotes the path length from $s$ to sink $i$ in $T$
- $D_i$ denotes the shortest path length from $s$ to sink $i$ in $G(V, E)$.

# Outline

# The Algorithm-Overview



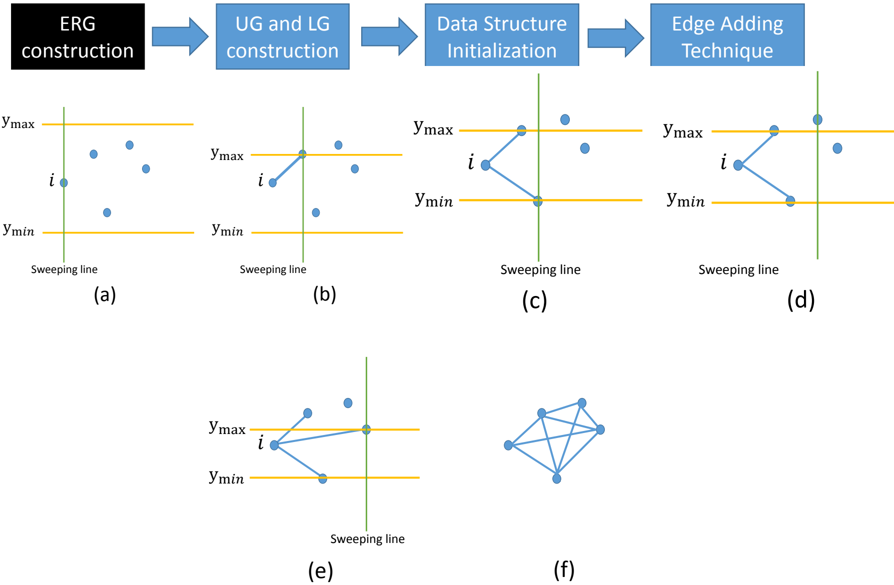
# The Algorithm-Edge Reduced Graph (ERG)



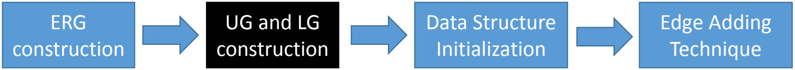

## Definition

*Edge Reduced Graph* $ERG(V, E)$ Given a set of points $V$ in the $(R^2, \ell_1)$ 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 $y_j \leq y_k \leq y_i$ .

# The Algorithm-Edge Reduced Graph (ERG)



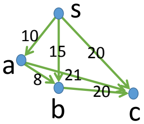
# The Algorithm-UG and LG




## Lower Bound Graph $LG(V, E')$

- edge $e_{pq} \in E'$ iff $e_{pq}$ satisfies

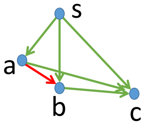
$$D_p + w_{pq} \leq (1 + \alpha) \cdot D_q \qquad (2)$$




$\alpha = 0.5$

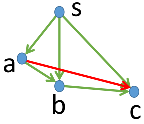
$D_a = 10$
$D_b = 15$
$D_c = 20$

(a)




$w_{ab} = 8$
$D_a + w_{ab} \leq 1.5 D_b$
$10 + 8 \leq 22.5$
$e_{ab}$ *in LG*

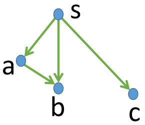
(b)




$w_{ac} = 21$
$D_a + w_{ac} \geq 1.5 D_c$
$10 + 21 \geq 30$
$e_{ac}$ *not in LG*

*Similarly,*
$e_{bc}$ *not in LG*

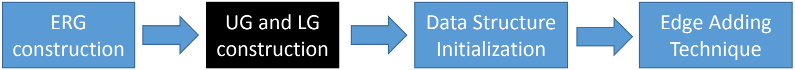
(c)




LG is obtained

(d)

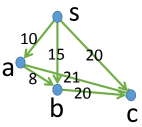
# The Algorithm-UG and LG




## Upper Bound Graph $UG(V, E^*)$

- edge $e_{pq} \in E^*$ iff $e_{pq}$ satisfies

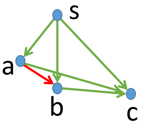
$$(1+\alpha) \cdot D_p + w_{pq} \leq (1+\alpha) \cdot D_q \quad (3)$$




$\alpha = 0.5$

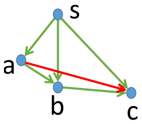
$D_a = 10$
$D_b = 15$
$D_c = 20$

(a)




$w_{ab} = 8$
$1.5D_a + w_{ab} \geq 1.5D_b$
$15 + 8 \geq 22.5$
$e_{ab}$ *not in LG*

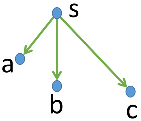
(b)




$w_{ac} = 21$
$1.5D_a + w_{ac} \geq 1.5D_c$
$15 + 21 \geq 30$
$e_{ac}$ *not in LG*

*Similarly,*
$e_{bc}$ *not in LG*

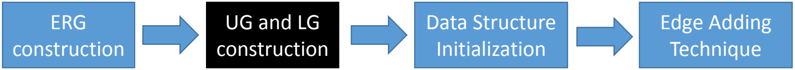
(c)




UG is obtained

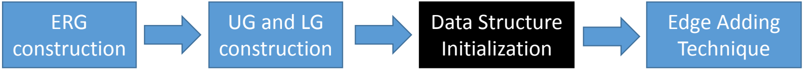
(d)

# The Algorithm-UG and LG



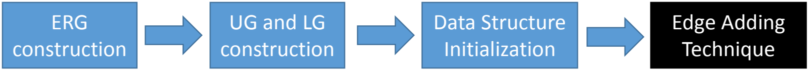

- Get shortest path length $D_i$ for each sink $i \in V$
- Obtain upper bound graph $UG$ and lower bound graph $LG$
- Get a minimum spanning tree $T_{M\_UG}$ on $UG$
- Sort the edges in $LG$ in non-decreasing order

# The Algorithm-Data Structure

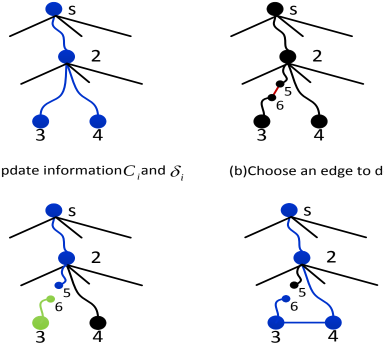



- each node keeps more information
- speed up the algorithm
- initialized at the beginning
- updated during the process

# The Algorithm-Edge Adding Technique




For $e \in$ edges in $LG$, try to add the edge $e$ to $T_{M\_UG}$

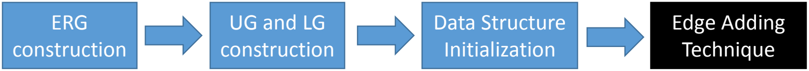



(a)Update information $C_i$ and $\delta_i$ (b)Choose an edge to delete

(c)Remove slack information (d)Add slack information

Examples of adding edge $e_{43}$

# The Algorithm-Edge Adding Technique



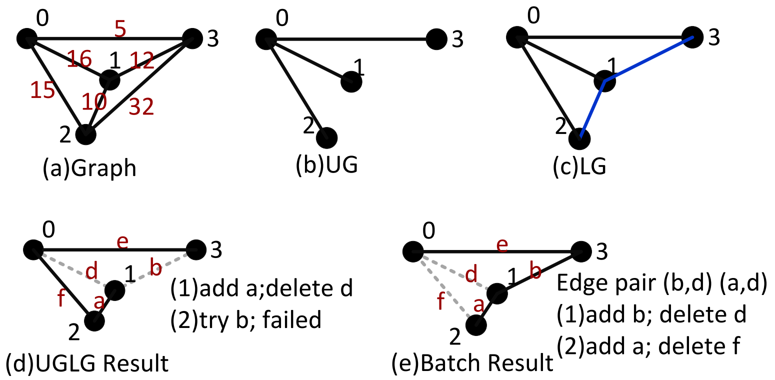

- Safe checking
- Update $C_i$ and $\delta_i$ of node $i$ in two paths from $s$ to $p$ and $q$
- an edge $e_{uv}$ to delete
  - Remove slack information
  - Add slack information
- $T' \leftarrow T$

# The Algorithm-Rectilinearization

It compares each pair of adjacent edges and estimates a reduced cost according to their bounding box. The pairs giving the maximum cost reduction will be processed to remove overlapped edges.

# The Algorithm-Batch Algorithm

$$\mathit{cost\_reduction} = \Delta w - \Delta path \tag{4}$$



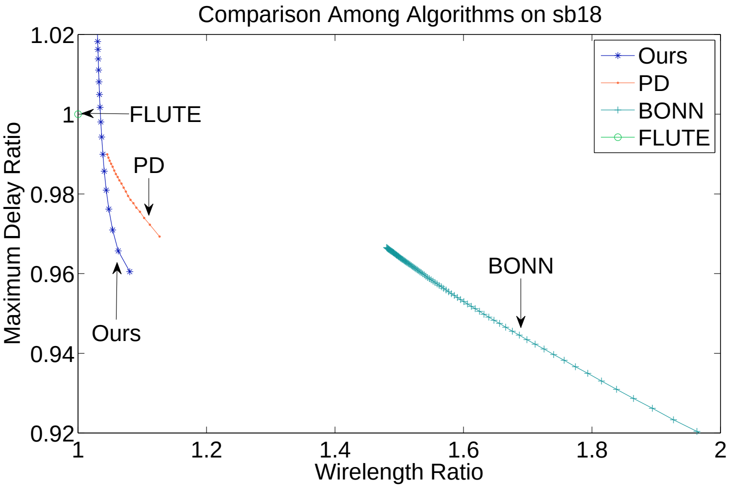
# Outline

# Results-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 |
| $[50, \infty)$ | 9 | 14 | 14 | 20 | 19 | 35 | 30 | 26 |
| total | 771542 | 999902 | 802513 | 1898119 | 1215710 | 1224979 | 1100825 | 1933945 |

# Results

## Comparision Among Algorithms



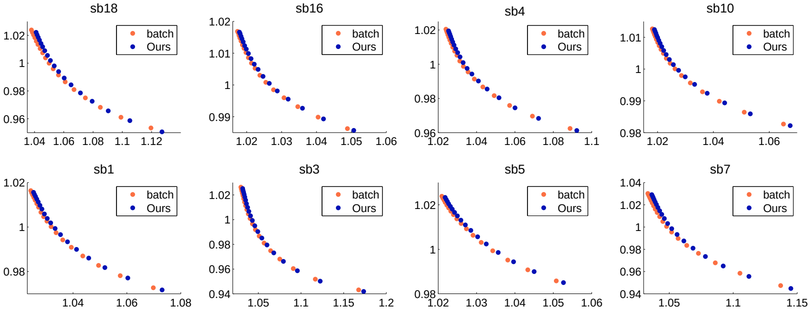
# Results

## 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 |

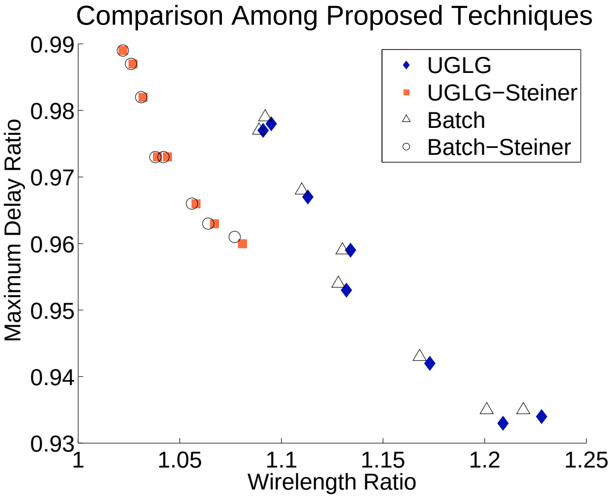
# Results

## Comparison with Batch Algorithm



# Results

Comparison Among Proposed Techniques



# Outline

# Conclusions

- $ERG$ is constructed with smaller edges.
- $UG$ and $LG$ owns good timing properties.
- data structure is designed for efficiency.
- two techniques overlap removal and batch algorithm are used.
- Results have better qualities.

Thanks!