Tight Bounds for Online Tree Augmentation

David P. Williamson
Cornell University
davidpwilliamson@cornell.edu

Joint work with Joseph (Seffi) Naor (Technion)
and Seeun William Umboh (University of Sydney)

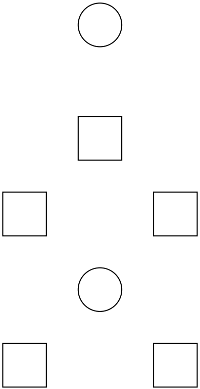
Paper to appear in ICALP 2019

June 3, 2019

Survivable Network Design Problem

Given an undirected network $G = (V, E)$, costs $c_e \geq 0$ for $e \in E$, source-sink pairs s_1-t_1,..., s_k-t_k, and requirements $r_1, \ldots, r_k$, find minimum-cost edges $F \subseteq E$ such that at least r_i edge-disjoint paths between s_i and t_i for $i = 1, \ldots, k$.

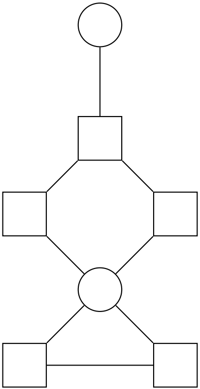
NP-hard even if $r_i = 1$, and $s_i = r$ for all i (Karp '72): Steiner tree problem



Survivable Network Design Problem

Given an undirected network $G = (V, E)$, costs $c_e \geq 0$ for $e \in E$, source-sink pairs s_1-t_1,. . ., s_k-t_k, and requirements $r_1, \ldots, r_k$, find minimum-cost edges $F \subseteq E$ such that at least r_i edge-disjoint paths between s_i and t_i for $i = 1, \ldots, k$.

NP-hard even if $r_i = 1$, and $s_i = r$ for all i (Karp '72): Steiner tree problem



Online Problems

If network requirements arrive over time, consider an *online* version of the problem. As each requirement arrives, must augment network to satisfy that requirement without knowledge of future requirements.

Online Problems

If network requirements arrive over time, consider an *online* version of the problem. As each requirement arrives, must augment network to satisfy that requirement without knowledge of future requirements.

Quality of algorithm determined by its *competitive ratio*: worst-case ratio over all inputs of cost of algorithm's solution to the minimum-cost solution for all requirements in the input.

Online Steiner Tree

An early case: Imase and Waxman (1991) give an $O(\log k)$-competitive algorithm for the online Steiner tree problem, which $r_i = 1$ and $s_i = r$ for all i. They also show that any algorithm must have competitive ratio at least $\Omega(\log k)$.

Online Steiner Tree

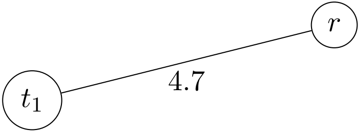
An early case: Imase and Waxman (1991) give an $O(\log k)$-competitive algorithm for the online Steiner tree problem, which $r_i = 1$ and $s_i = r$ for all i. They also show that any algorithm must have competitive ratio at least $\Omega(\log k)$.

Greedy algorithm: When next t_i arrives, buy a path to closest t_j for $j < i$ or to root r.

Online Steiner Tree

An early case: Imase and Waxman (1991) give an $O(\log k)$-competitive algorithm for the online Steiner tree problem, which $r_i = 1$ and $s_i = r$ for all i. They also show that any algorithm must have competitive ratio at least $\Omega(\log k)$.

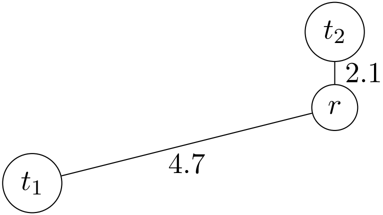
Greedy algorithm: When next t_i arrives, buy a path to closest t_j for $j < i$ or to root r.



Online Steiner Tree

An early case: Imase and Waxman (1991) give an $O(\log k)$-competitive algorithm for the online Steiner tree problem, which $r_i = 1$ and $s_i = r$ for all i. They also show that any algorithm must have competitive ratio at least $\Omega(\log k)$.

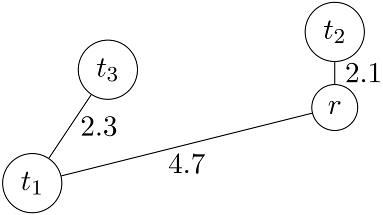
Greedy algorithm: When next t_i arrives, buy a path to closest t_j for $j < i$ or to root r.



Online Steiner Tree

An early case: Imase and Waxman (1991) give an $O(\log k)$-competitive algorithm for the online Steiner tree problem, which $r_i = 1$ and $s_i = r$ for all i. They also show that any algorithm must have competitive ratio at least $\Omega(\log k)$.

Greedy algorithm: When next t_i arrives, buy a path to closest t_j for $j < i$ or to root r.



A Quick Analysis

Let c_i be cost algorithm pays to connect t_i when it arrives. Let Z_j be set of indices i with $c_i \in [2^j, 2^{j+1})$.

Algorithm's cost is then

$$\sum_j \sum_{i \in Z_j} c_j \leq \sum_j 2^{j+1} |Z_j|.$$

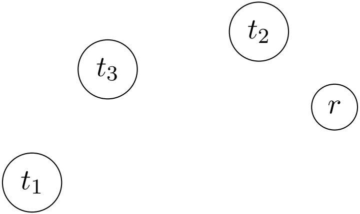
A Quick Analysis

Lemma

For any j, $OPT \geq 2^{j-1}|Z_j|$.

Proof.

Cost of path between any pair of vertices in Z_j is at least 2^j.
Put disjoint balls of radius 2^{j-1} around each point in Z_j. $\square$



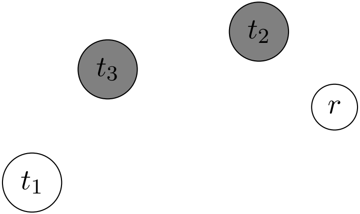
A Quick Analysis

Lemma

For any j, $OPT \geq 2^{j-1}|Z_j|$.

Proof.

Cost of path between any pair of vertices in Z_j is at least 2^j. Put disjoint balls of radius 2^{j-1} around each point in Z_j. □



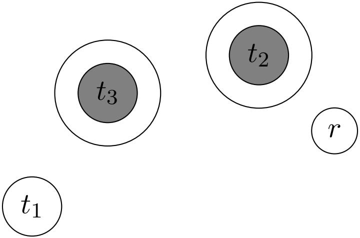
A Quick Analysis

Lemma

For any j, $OPT \geq 2^{j-1}|Z_j|$.

Proof.

Cost of path between any pair of vertices in Z_j is at least 2^j. Put disjoint balls of radius 2^{j-1} around each point in Z_j. □



A Quick Analysis

Algorithm's cost at most $\sum_j 2^{j+1}|Z_j|$, OPT $\geq 2^{j-1}|Z_j|$ for all j.

A Quick Analysis

Algorithm's cost at most $\sum_j 2^{j+1}|Z_j|$, OPT $\geq 2^{j-1}|Z_j|$ for all j.

If ℓ highest index such that $Z_\ell \neq \emptyset$, then:

$$
\begin{aligned}
2^{\ell+1}|Z_\ell| &\leq 4 \cdot \mathrm{OPT} \\
2^{\ell}|Z_{\ell-1}| &\leq 4 \cdot \mathrm{OPT} \\
&\vdots \\
2^{\ell-\lceil \log_2 k \rceil+1}|Z_{\ell-\lceil \log_2 k \rceil}| &\leq 4 \cdot \mathrm{OPT} \\
\sum_{j<\ell-\lceil \log_2 k \rceil} 2^{j+1}|Z_j| &\leq \frac{2^\ell}{k} \sum_j |Z_j| \leq 2^\ell \leq 2 \cdot \mathrm{OPT}
\end{aligned}
$$

A Quick Analysis

Algorithm's cost at most $\sum_j 2^{j+1}|Z_j|$, OPT $\geq 2^{j-1}|Z_j|$ for all j.

If ℓ highest index such that $Z_\ell \neq \emptyset$, then:

$$\begin{aligned}
2^{\ell+1}|Z_\ell| &\leq 4 \cdot \mathrm{OPT} \\
2^{\ell}|Z_{\ell-1}| &\leq 4 \cdot \mathrm{OPT} \\
&\vdots \\
2^{\ell-\lceil \log_2 k \rceil+1}|Z_{\ell-\lceil \log_2 k \rceil}| &\leq 4 \cdot \mathrm{OPT} \\
\sum_{j<\ell-\lceil \log_2 k \rceil} 2^{j+1}|Z_j| &\leq \frac{2^\ell}{k} \sum_j |Z_j| \leq 2^\ell \leq 2 \cdot \mathrm{OPT}
\end{aligned}$$

Summing the inequalities together, we get that the algorithm's cost is at most $O(\log k)$OPT.

Higher Connectivities

$O(\log k)$-competitive algorithm known for $r_i = 1$, arbitrary s_i-t_i pairs (Berman, Coulston 1997), other types of connectivity (Qian, Umboh, W 2018), node-weighted problems (Hajiaghayi, Liaghat, Panigrahi 2013).

Higher Connectivities

$O(\log k)$-competitive algorithm known for $r_i = 1$, arbitrary s_i-t_i pairs (Berman, Coulston 1997), other types of connectivity (Qian, Umboh, W 2018), node-weighted problems (Hajiaghayi, Liaghat, Panigrahi 2013).

For online survivable network design, Gupta, Krishnaswamy, and Ravi (2012) show a randomized $O(r_{\max} \log^3 n)$-competitive algorithm, where $n = |V|$ in input graph, $r_{\max} = \max_i r_i$.

Higher Connectivities

$O(\log k)$-competitive algorithm known for $r_i = 1$, arbitrary s_i-t_i pairs (Berman, Coulston 1997), other types of connectivity (Qian, Umboh, W 2018), node-weighted problems (Hajiaghayi, Liaghat, Panigrahi 2013).

For online survivable network design, Gupta, Krishnaswamy, and Ravi (2012) show a randomized $O(r_{\max} \log^3 n)$-competitive algorithm, where $n = |V|$ in input graph, $r_{\max} = \max_i r_i$.

Question

Can we do better? Better competitive ratio? Deterministic algorithm?

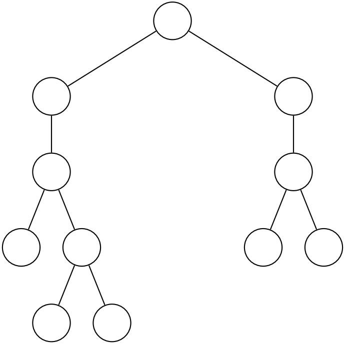
Tree Augmentation Problem

The minimal, interesting variant of online survivable network design for which we do not have an $O(\log n)$-competitive algorithm: online tree augmentation.

Tree Augmentation Problem

The minimal, interesting variant of online survivable network design for which we do not have an $O(\log n)$-competitive algorithm: online tree augmentation.

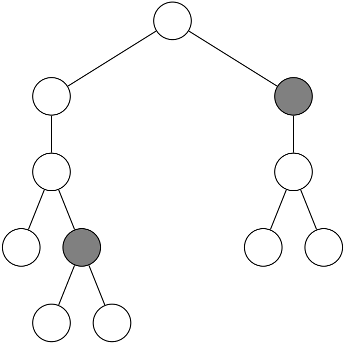
Given a spanning tree T on a node set V, and a set $L \subseteq \binom{V}{2}$ of *links*, cost $c(\ell)$ for link $\ell \in L$. Requests (s_i, t_i) arrive over time; find minimum-cost $F \subseteq L$ such that for each i, there are at least two edge-disjoint paths between s_i and t_i in $T \cup F$ for all i.



Tree Augmentation Problem

The minimal, interesting variant of online survivable network design for which we do not have an $O(\log n)$-competitive algorithm: online tree augmentation.

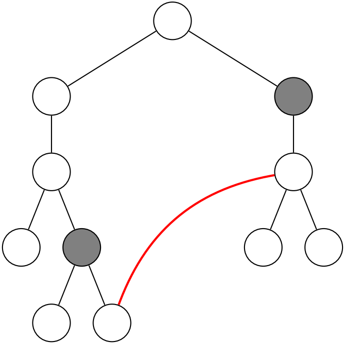
Given a spanning tree T on a node set V, and a set $L \subseteq \binom{V}{2}$ of *links*, cost $c(\ell)$ for link $\ell \in L$. Requests (s_i, t_i) arrive over time; find minimum-cost $F \subseteq L$ such that for each i, there are at least two edge-disjoint paths between s_i and t_i in $T \cup F$ for all i.



Tree Augmentation Problem

The minimal, interesting variant of online survivable network design for which we do not have an $O(\log n)$-competitive algorithm: online tree augmentation.

Given a spanning tree T on a node set V, and a set $L \subseteq \binom{V}{2}$ of *links*, cost $c(\ell)$ for link $\ell \in L$. Requests (s_i, t_i) arrive over time; find minimum-cost $F \subseteq L$ such that for each i, there are at least two edge-disjoint paths between s_i and t_i in $T \cup F$ for all i.



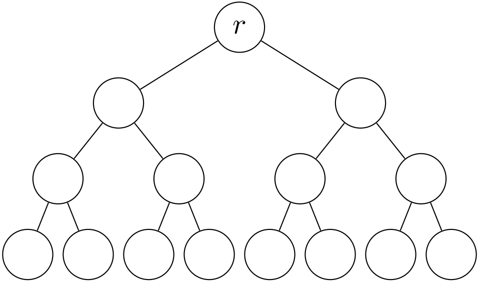
Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

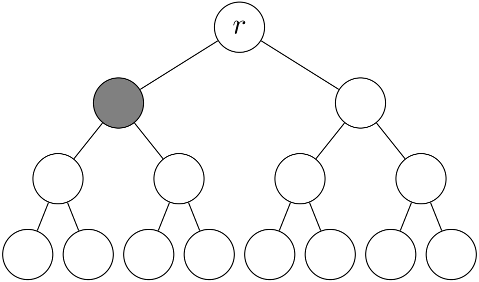
Complete binary tree, links of cost 1 from each leaf to the root, all requests have $s_i = r$.



Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

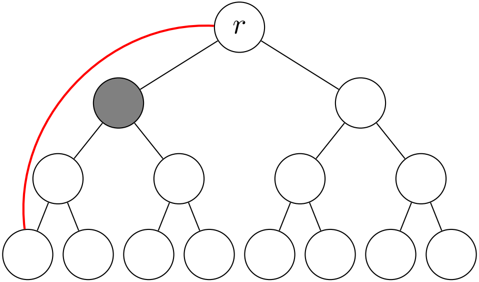
Complete binary tree, links of cost 1 from each leaf to the root, all requests have $s_i = r$.



Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

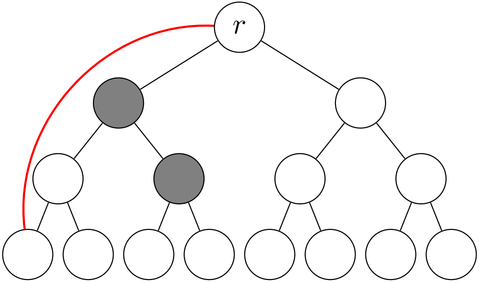
Complete binary tree, links of cost 1 from each leaf to the root, all requests have $s_i = r$.



Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

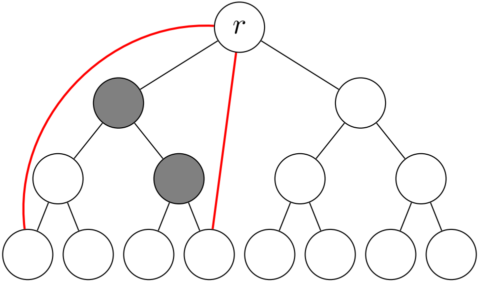
Complete binary tree, links of cost 1 from each leaf to the root, all requests have $s_i = r$.



Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

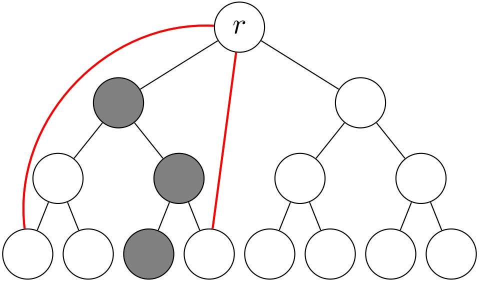
Complete binary tree, links of cost 1 from each leaf to the root, all requests have $s_i = r$.



Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

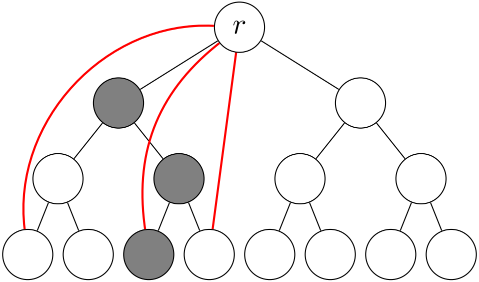
Complete binary tree, links of cost 1 from each leaf to the root, all requests have $s_i = r$.



Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

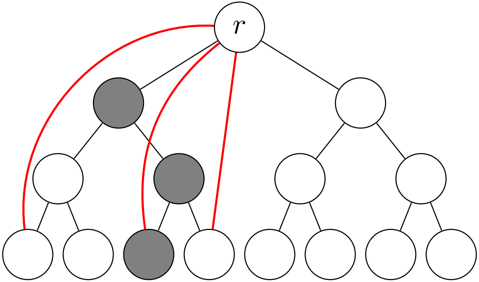
Complete binary tree, links of cost 1 from each leaf to the root, all requests have $s_i = r$.



Lower Bound

Gupta, Krishnaswamy, and Ravi show an $\Omega(\log n)$ lower bound on the competitive ratio.

Complete binary tree, links of cost 1 from each leaf to the root, all requests have $s_i = r$.



Optimal only buys last link, algorithm must buy $\log_2 n - 1$ links.

Our Result

Theorem (Naor, Umboh, W 2019)

There is a deterministic $O(\log n)$-competitive algorithm for the online tree augmentation problem.

Our Result

Theorem (Naor, Umboh, W 2019)

There is a deterministic $O(\log n)$-competitive algorithm for the online tree augmentation problem.

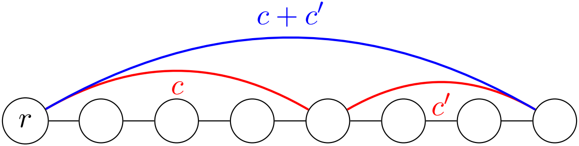
Main ingredients:

1. An algorithm for paths
2. Decomposition of trees into paths
3. A refined path algorithm

Ingredient 1: An Algorithm for Paths

Suppose tree T is a path P, all requests are *rooted*: (r, t_i). Assume WLOG:

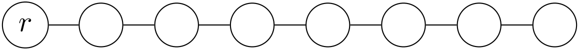
- no nonrooted links exist;



Ingredient 1: An Algorithm for Paths

Suppose tree T is a path P, all requests are *rooted*: (r, t_i).
Assume WLOG:

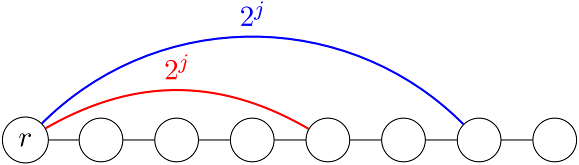
- no nonrooted links exist;
- link costs are 2^j;



Ingredient 1: An Algorithm for Paths

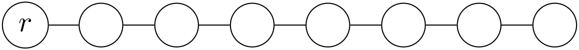
Suppose tree T is a path P, all requests are *rooted*: (r, t_i).
Assume WLOG:

- no nonrooted links exist;
- link costs are 2^j;
- at most one link of cost 2^j.



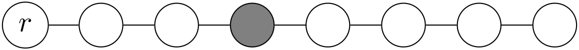
Ingredient 1: An Algorithm for Paths

Algorithm: Given request (r, t_i) not already covered, buy cheapest link (r, v) that covers request.



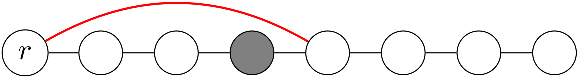
Ingredient 1: An Algorithm for Paths

Algorithm: Given request (r, t_i) not already covered, buy cheapest link (r, v) that covers request.



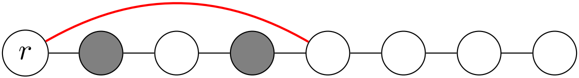
Ingredient 1: An Algorithm for Paths

Algorithm: Given request (r, t_i) not already covered, buy cheapest link (r, v) that covers request.



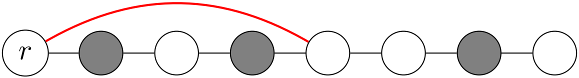
Ingredient 1: An Algorithm for Paths

Algorithm: Given request (r, t_i) not already covered, buy cheapest link (r, v) that covers request.



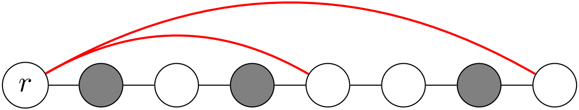
Ingredient 1: An Algorithm for Paths

Algorithm: Given request (r, t_i) not already covered, buy cheapest link (r, v) that covers request.



Ingredient 1: An Algorithm for Paths

Algorithm: Given request (r, t_i) not already covered, buy cheapest link (r, v) that covers request.



Ingredient 1: An Algorithm for Paths

Theorem

The algorithm is $O(1)$-competitive.

Proof.

Factor of 2 for rounding link costs up to nearest power of 2.

Algorithm buys at most one link of cost 2^j for each j. Consider request (r, t_i) such that cheapest link that covers request is 2^ℓ for ℓ maximum. Then

$$\mathrm{OPT} \geq 2^\ell,$$

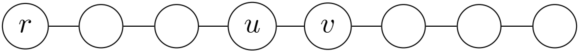
while algorithm pays at most

$$2^\ell + 2^{\ell-1} + 2^{\ell-2} + \cdots = 2^{\ell+1} \leq 2 \cdot \mathrm{OPT}.$$

□

Ingredient 1: An Algorithm for Paths

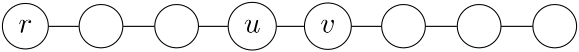
What about non-rooted requests? Assume:



Ingredient 1: An Algorithm for Paths

What about non-rooted requests? Assume:

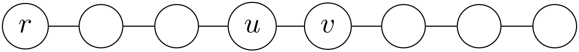
- All link costs 2^j;



Ingredient 1: An Algorithm for Paths

What about non-rooted requests? Assume:

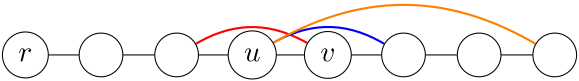
- All link costs 2^j;
- Requests are only of edges $(u, v) \in P$;



Ingredient 1: An Algorithm for Paths

What about non-rooted requests? Assume:

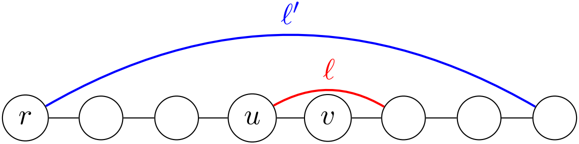
- All link costs 2^j;
- Requests are only of edges $(u, v) \in P$;
- At most two links of cost 2^j contain any edge $(u, v) \in P$;



Ingredient 1: An Algorithm for Paths

What about non-rooted requests? Assume:

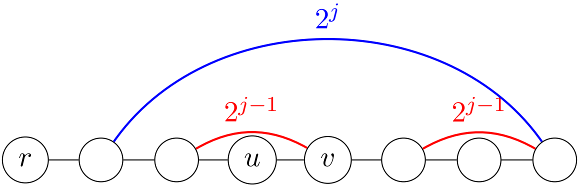
- All link costs 2^j;
- Requests are only of edges $(u, v) \in P$;
- At most two links of cost 2^j contain any edge $(u, v) \in P$;
- Any link ℓ' containing a link ℓ has strictly greater cost.



Ingredient 1: An Algorithm for Paths

What about non-rooted requests? Assume:

- All link costs 2^j;
- Requests are only of edges $(u, v) \in P$;
- At most two links of cost 2^j contain any edge $(u, v) \in P$;
- Any link ℓ' containing a link ℓ has strictly greater cost.
- Any link of cost 2^j contains at most 2^k disjoint links of cost 2^{j-k}.



Ingredient 1: An Algorithm for Paths

Algorithm: If request $(u, v) \in P$ not covered, buy (two) cheapest link(s) covering (u, v).

Let Z_j be the set of links of cost 2^j bought by algorithm, so that algorithm's cost is

$$\sum_j 2^j |Z_j|.$$

Ingredient 1: An Algorithm for Paths

Algorithm: If request $(u, v) \in P$ not covered, buy (two) cheapest link(s) covering (u, v).

Let Z_j be the set of links of cost 2^j bought by algorithm, so that algorithm's cost is

$$\sum_j 2^j |Z_j|.$$

Claim

$$OPT \geq \frac{1}{2} \cdot 2^j \cdot |Z_j|.$$

Ingredient 1: An Algorithm for Paths

Algorithm: If request $(u, v) \in P$ not covered, buy (two) cheapest link(s) covering (u, v).

Let Z_j be the set of links of cost 2^j bought by algorithm, so that algorithm's cost is

$$\sum_j 2^j |Z_j|.$$

Claim

$$OPT \geq \frac{1}{2} \cdot 2^j \cdot |Z_j|.$$

Theorem

The algorithm is $O(\log n)$-competitive.

Ingredient 1: An Algorithm for Paths

Algorithm: If request $(u, v) \in P$ not covered, buy (two) cheapest link(s) covering (u, v).

Let Z_j be the set of links of cost 2^j bought by algorithm, so that algorithm's cost is

$$\sum_j 2^j |Z_j|.$$

Claim

$$OPT \geq \frac{1}{2} \cdot 2^j \cdot |Z_j|.$$

Theorem

The algorithm is $O(\log n)$-competitive.

Proof.

Essentially the same as for online Steiner tree. □

Ingredient 1: An Algorithm for Paths

Can get $O(\log n)$-competitive algorithm using primal-dual/dual-fitting arguments.

Ingredient 1: An Algorithm for Paths

Can get $O(\log n)$-competitive algorithm using primal-dual/dual-fitting arguments.

Theorem (Naor, Umboh, W 2019)

Any deterministic algorithm for the online path augmentation problem has competitive ratio $\Omega(\log n)$.

Improves on a result of Meyerson (2005) of $\Omega(\log n / \log\log n)$.

Ingredient 2: Tree Decomposition

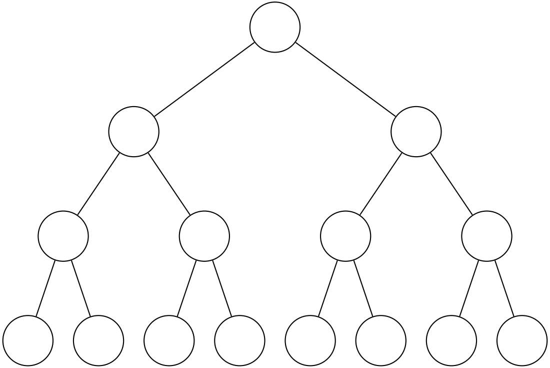
Theorem (Sleator, Tarjan (1983))

Any rooted tree T can be decomposed into disjoint paths $\mathcal{P}$ such that each path in $\mathcal{P}$ is rooted (has an LCA closest to root), any path in T intersects at most $O(\log n)$ paths in $\mathcal{P}$.

Ingredient 2: Tree Decomposition

Theorem (Sleator, Tarjan (1983))

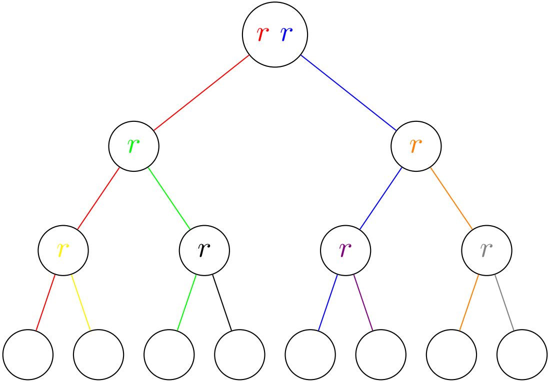
Any rooted tree T can be decomposed into disjoint paths $\mathcal{P}$ such that each path in $\mathcal{P}$ is rooted (has an LCA closest to root), any path in T intersects at most $O(\log n)$ paths in $\mathcal{P}$.



Ingredient 2: Tree Decomposition

Theorem (Sleator, Tarjan (1983))

Any rooted tree T can be decomposed into disjoint paths $\mathcal{P}$ such that each path in $\mathcal{P}$ is rooted (has an LCA closest to root), any path in T intersects at most $O(\log n)$ paths in $\mathcal{P}$.

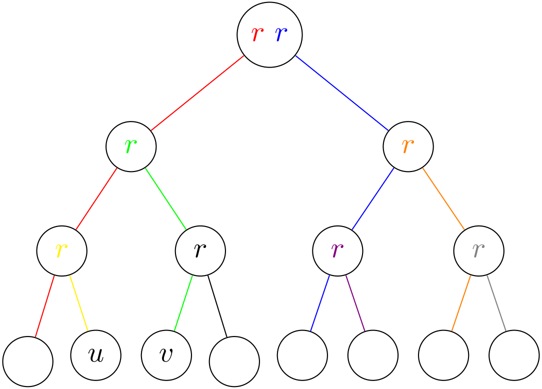


Ingredient 2: Tree Decomposition

Let $\mathcal{P}$ be a decomposition of tree T into rooted paths.

Definition

The *projection* of a link (u, v) on to a rooted path $P \in \mathcal{P}$ is the link whose endpoints are the endpoints of $P \cap T(u, v)$, where $T(u, v)$ is the u-v path in T.

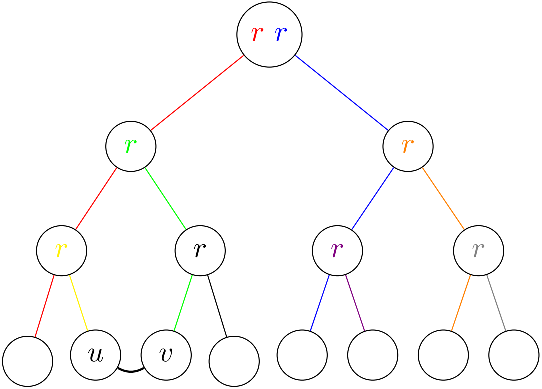


Ingredient 2: Tree Decomposition

Let $\mathcal{P}$ be a decomposition of tree T into rooted paths.

Definition

The *projection* of a link (u, v) on to a rooted path $P \in \mathcal{P}$ is the link whose endpoints are the endpoints of $P \cap T(u, v)$, where $T(u, v)$ is the u-v path in T.

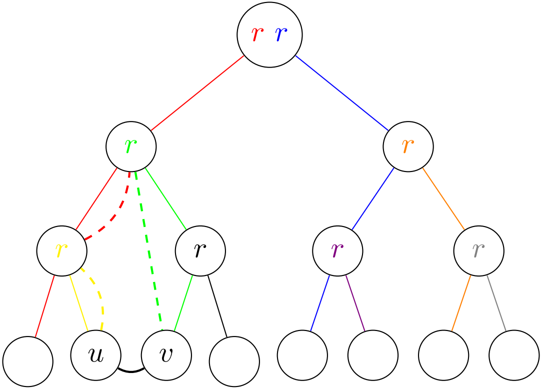


Ingredient 2: Tree Decomposition

Let $\mathcal{P}$ be a decomposition of tree T into rooted paths.

Definition

The *projection* of a link (u, v) on to a rooted path $P \in \mathcal{P}$ is the link whose endpoints are the endpoints of $P \cap T(u, v)$, where $T(u, v)$ is the u-v path in T.



Algorithm Idea

Assume WLOG each request (s_i, t_i) is an edge of the tree.

Algorithm Idea

Assume WLOG each request (s_i, t_i) is an edge of the tree.

Idea:

- When request $(s_i, t_i) \in T$ arrives, run an online algorithm for path $P \in \mathcal{P}$ such that $(s_i, t_i) \in P$.

Algorithm Idea

Assume WLOG each request (s_i, t_i) is an edge of the tree.

Idea:

- When request $(s_i, t_i) \in T$ arrives, run an online algorithm for path $P \in \mathcal{P}$ such that $(s_i, t_i) \in P$.
- Consider projections of all links ℓ on to P, and buy ℓ if online algorithm buys the projected link.

Algorithm Idea

Assume WLOG each request (s_i, t_i) is an edge of the tree.

Idea:

- When request $(s_i, t_i) \in T$ arrives, run an online algorithm for path $P \in \mathcal{P}$ such that $(s_i, t_i) \in P$.
- Consider projections of all links ℓ on to P, and buy ℓ if online algorithm buys the projected link.

Decomposition tells us each link projects onto $O(\log n)$ paths $P \in \mathcal{P}$.

Together with our $O(\log n)$-competitive online path augmentation algorithm, this gives an $O(\log^2 n)$-competitive algorithm for online tree augmentation.

Algorithm Idea

Assume WLOG each request (s_i, t_i) is an edge of the tree.

Idea:

- When request $(s_i, t_i) \in T$ arrives, run an online algorithm for path $P \in \mathcal{P}$ such that $(s_i, t_i) \in P$.
- Consider projections of all links ℓ on to P, and buy ℓ if online algorithm buys the projected link.

Decomposition tells us each link projects onto $O(\log n)$ paths $P \in \mathcal{P}$.

Together with our $O(\log n)$-competitive online path augmentation algorithm, this gives an $O(\log^2 n)$-competitive algorithm for online tree augmentation.

How can we do better?

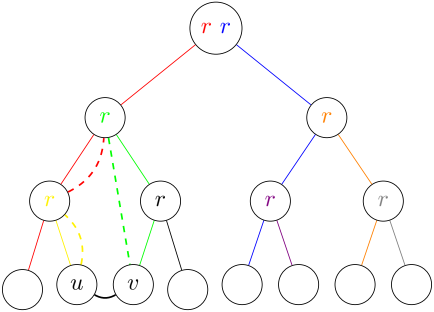
Ingredient 2: Tree Decomposition

Definition

A projection of a link (u, v) on to a rooted path P is *rooted* if one endpoint of the projection is the root of the path P.

Lemma

For any given link (u, v), its projection on to all but one path $P \in \mathcal{P}$ is rooted.



Ingredient 3: Refined Path Algorithm

Definition

An online algorithm for path augmentation is *nice* if for any feasible solution F^* it produces a solution of cost at most

$$O(1)c(R^*) + O(\log n)c(S^*),$$

where R^* are the rooted links in F^* and S^* are the non-rooted links in F^*.

Ingredient 3: Refined Path Algorithm

Definition

An online algorithm for path augmentation is *nice* if for any feasible solution F^* it produces a solution of cost at most

$$O(1)c(R^*) + O(\log n)c(S^*),$$

where R^* are the rooted links in F^* and S^* are the non-rooted links in F^*.

Theorem

Given a deterministic nice algorithm for online path augmentation, we get a deterministic $O(\log n)$-competitive algorithm for online tree augmentation.

Proof Sketch

Theorem

Given a deterministic nice algorithm for online path augmentation, we get a deterministic $O(\log n)$-competitive algorithm for online tree augmentation.

Proof.

For feasible solution F^* for the tree augmentation problem, let R_P^* be links of F^* whose projections on to $P \in \mathcal{P}$ are rooted, S_P^* be links of F^* that have projections on to P are non-rooted. Then cost of algorithm's solution is at most

$$\sum_{P \in \mathcal{P}} (O(1)c(R_P^*) + O(\log n)c(S_P^*)) \leq O(\log n)c(F^*).$$

$\square$

The Rest

Some amount of work needed to get all of the ideas to work together.

Open Questions

Recall that Gupta, Krishnaswamy, Ravi (2012) give a $O(r_{\max} \log^3 n)$-competitive algorithm for online survivable network design.

Open Questions

Recall that Gupta, Krishnaswamy, Ravi (2012) give a $O(r_{\max}\log^3 n)$-competitive algorithm for online survivable network design.

- Is the linear dependence on $r_{\max}$ necessary?

Open Questions

Recall that Gupta, Krishnaswamy, Ravi (2012) give a $O(r_{\max} \log^3 n)$-competitive algorithm for online survivable network design.

- Is the linear dependence on $r_{\max}$ necessary?
- Are the polylogs necessary?

Open Questions

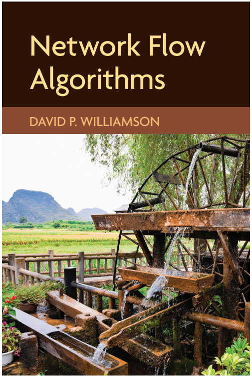
Recall that Gupta, Krishnaswamy, Ravi (2012) give a $O(r_{\max} \log^3 n)$-competitive algorithm for online survivable network design.

- Is the linear dependence on $r_{\max}$ necessary?
- Are the polylogs necessary?
- Is there an $O(\log n)$-competitive algorithm in the case $r_{\max} = 2$?

Other Work

I also spent the semester finishing a book, to be published by Cambridge this fall.

Online PDF available at
`www.networkflowalgs.com/book.pdf`.



Thanks for your attention.