A Note on a Recent Algorithm for Minimum Cut

Paweł Gawrychowski[1] Shay Mozes[2] Oren Weimann[3]

[1]University of Wrocław, Poland

[2]The Interdisciplinary Center Herzliya, Israel

[3]University of Haifa, Israel

Slides by Paweł Gawrychowski

(Global) Minimum Cut

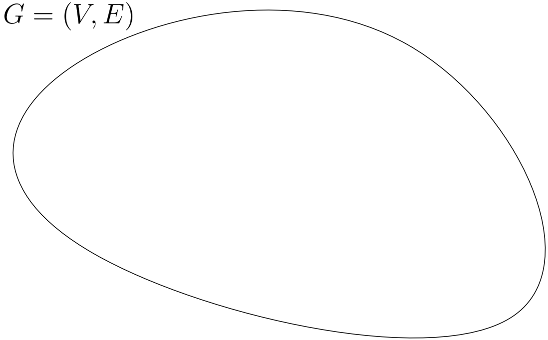
Input: undirected edge-weighted graph $G = (V, E)$
Output: nonempty $S \subset V$ minimizing the total weight of edges between S and $V \setminus S$

Solvable in polynomial time with $n - 1$ maximum flow computations.

(Global) Minimum Cut

Input: undirected edge-weighted graph $G = (V, E)$

Output: nonempty $S \subset V$ minimizing the total weight of edges between S and $V \setminus S$

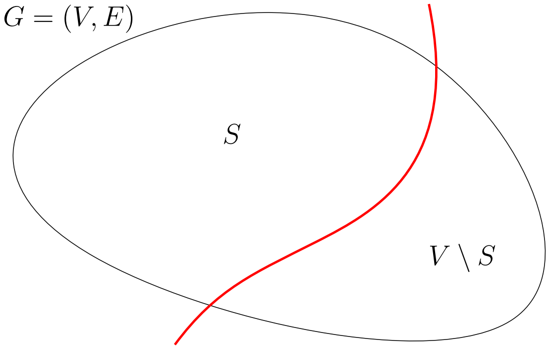


Solvable in polynomial time with $n - 1$ maximum flow computations.

(Global) Minimum Cut

Input: undirected edge-weighted graph $G = (V, E)$

Output: nonempty $S \subset V$ minimizing the total weight of edges between S and $V \setminus S$

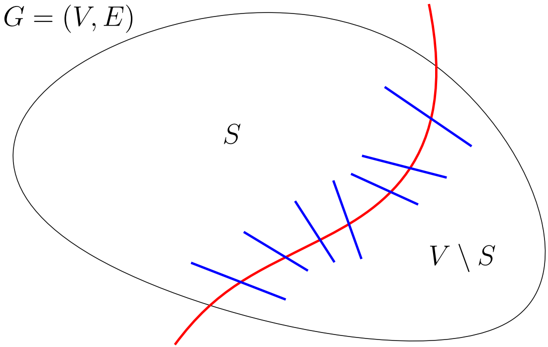


Solvable in polynomial time with $n - 1$ maximum flow computations.

(Global) Minimum Cut

Input: undirected edge-weighted graph $G = (V, E)$

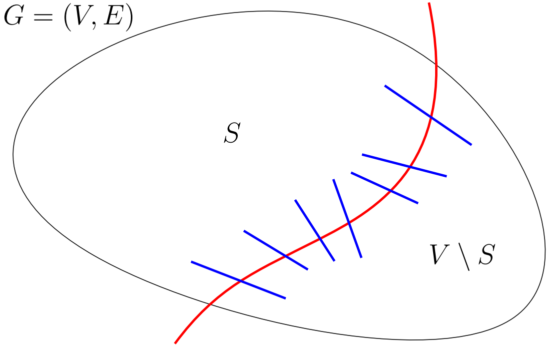
Output: nonempty $S \subset V$ minimizing the total weight of edges between S and $V \setminus S$



Solvable in polynomial time with $n - 1$ maximum flow computations.

(Global) Minimum Cut

Input: undirected edge-weighted graph $G = (V, E)$
Output: nonempty $S \subset V$ minimizing the total weight of edges between S and $V \setminus S$



Solvable in polynomial time with $n - 1$ maximum flow computations.

The fastest maximum flow algorithm works in $\mathcal{O}(nm)$ time, so applying it $n-1$ times results in $\mathcal{O}(n^2m)$ complexity. Is there a faster algorithm?

Nagamochi and Ibaraki 1992

In $\mathcal{O}(m + n\log n)$ time we can either find the global minimum cut or isolate an edge that doesn't cross it. This edge can then be contracted and the procedure repeated, resulting in $\mathcal{O}(mn + n^2\log n)$ complexity.

Karger and Stein 1996

A different method based on recursion and contracting a randomly chosen edge finds the global minimum cut in $\mathcal{O}(n^2\log^3 n)$ time whp.

Is there a more efficient algorithm for sparse graphs?

The fastest maximum flow algorithm works in $\mathcal{O}(nm)$ time, so applying it $n-1$ times results in $\mathcal{O}(n^2m)$ complexity. Is there a faster algorithm?

Nagamochi and Ibaraki 1992

In $\mathcal{O}(m+n\log n)$ time we can either find the global minimum cut or isolate an edge that doesn't cross it. This edge can then be contracted and the procedure repeated, resulting in $\mathcal{O}(mn+n^2\log n)$ complexity.

Karger and Stein 1996

A different method based on recursion and contracting a randomly chosen edge finds the global minimum cut in $\mathcal{O}(n^2\log^3 n)$ time whp.

Is there a more efficient algorithm for sparse graphs?

The fastest maximum flow algorithm works in $\mathcal{O}(nm)$ time, so applying it $n-1$ times results in $\mathcal{O}(n^2 m)$ complexity. Is there a faster algorithm?

Nagamochi and Ibaraki 1992

In $\mathcal{O}(m + n \log n)$ time we can either find the global minimum cut or isolate an edge that doesn't cross it. This edge can then be contracted and the procedure repeated, resulting in $\mathcal{O}(mn + n^2 \log n)$ complexity.

Karger and Stein 1996

A different method based on recursion and contracting a randomly chosen edge finds the global minimum cut in $\mathcal{O}(n^2 \log^3 n)$ time whp.

Is there a more efficient algorithm for sparse graphs?

The fastest maximum flow algorithm works in $\mathcal{O}(nm)$ time, so applying it $n-1$ times results in $\mathcal{O}(n^2m)$ complexity. Is there a faster algorithm?

Nagamochi and Ibaraki 1992

In $\mathcal{O}(m + n\log n)$ time we can either find the global minimum cut or isolate an edge that doesn't cross it. This edge can then be contracted and the procedure repeated, resulting in $\mathcal{O}(mn + n^2\log n)$ complexity.

Karger and Stein 1996

A different method based on recursion and contracting a randomly chosen edge finds the global minimum cut in $\mathcal{O}(n^2\log^3 n)$ time whp.

Is there a more efficient algorithm for sparse graphs?

Karger's Framework

In 1996, Karger announced a faster $\mathcal{O}(m \log^3 n)$ time algorithm finding the minimum cut whp. by solving $\mathcal{O}(\log n)$ independent instances of a more structured problem.

Minimum k-respecting cut

Given a spanning tree T, find the minimum cut crossed by exactly k of its edges.

Karger's Framework

In 1996, Karger announced a faster $\mathcal{O}(m \log^3 n)$ time algorithm finding the minimum cut whp. by solving $\mathcal{O}(\log n)$ independent instances of a more structured problem.

Minimum k-respecting cut

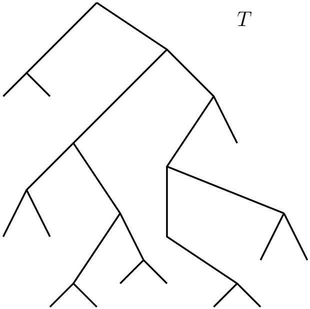
Given a spanning tree T, find the minimum cut crossed by exactly k of its edges.

Karger's Framework

In 1996, Karger announced a faster $\mathcal{O}(m \log^3 n)$ time algorithm finding the minimum cut whp. by solving $\mathcal{O}(\log n)$ independent instances of a more structured problem.

Minimum k-respecting cut

Given a spanning tree T, find the minimum cut crossed by exactly k of its edges.

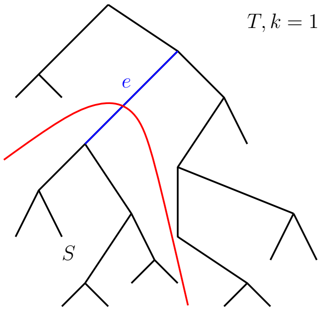


Karger's Framework

In 1996, Karger announced a faster $\mathcal{O}(m \log^3 n)$ time algorithm finding the minimum cut whp. by solving $\mathcal{O}(\log n)$ independent instances of a more structured problem.

Minimum k-respecting cut

Given a spanning tree T, find the minimum cut crossed by exactly k of its edges.

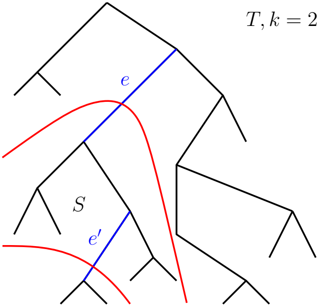


Karger's Framework

In 1996, Karger announced a faster $\mathcal{O}(m \log^3 n)$ time algorithm finding the minimum cut whp. by solving $\mathcal{O}(\log n)$ independent instances of a more structured problem.

Minimum k-respecting cut

Given a spanning tree T, find the minimum cut crossed by exactly k of its edges.



Karger's framework

The high-level structure of Karger's algorithm is as follows:

1. Find a collection $\mathcal{T}$ of $\mathcal{O}(\log n)$ trees such that whp the minimum cut 1- or 2-respects some $T \in \mathcal{T}$.
2. For every $T \in \mathcal{T}$, find the minimum 1-respecting cut.
3. For every $T \in \mathcal{T}$, find the minimum 2-respecting cut.

Finding $\mathcal{T}$

Uses the so-called (weighted) tree packing, building on a theorem by Nash-Williams, and can be implemented in $\mathcal{O}(m + n\log^3 n)$ time whp.

Karger's framework

The high-level structure of Karger's algorithm is as follows:

1. Find a collection $\mathcal{T}$ of $\mathcal{O}(\log n)$ trees such that whp the minimum cut 1- or 2-respects some $T \in \mathcal{T}$.
2. For every $T \in \mathcal{T}$, find the minimum 1-respecting cut.
3. For every $T \in \mathcal{T}$, find the minimum 2-respecting cut.

Finding $\mathcal{T}$

Uses the so-called (weighted) tree packing, building on a theorem by Nash-Williams, and can be implemented in $\mathcal{O}(m + n\log^3 n)$ time whp.

Karger's framework

The high-level structure of Karger's algorithm is as follows:

1. Find a collection $\mathcal{T}$ of $\mathcal{O}(\log n)$ trees such that whp the minimum cut 1- or 2-respects some $T \in \mathcal{T}$.
2. For every $T \in \mathcal{T}$, find the minimum 1-respecting cut.
3. For every $T \in \mathcal{T}$, find the minimum 2-respecting cut.

Finding $\mathcal{T}$

Uses the so-called (weighted) tree packing, building on a theorem by Nash-Williams, and can be implemented in $\mathcal{O}(m + n\log^3 n)$ time whp.

Karger's framework

The high-level structure of Karger's algorithm is as follows:

1. Find a collection $\mathcal{T}$ of $\mathcal{O}(\log n)$ trees such that whp the minimum cut 1- or 2-respects some $T \in \mathcal{T}$.
2. For every $T \in \mathcal{T}$, find the minimum 1-respecting cut.
3. For every $T \in \mathcal{T}$, find the minimum 2-respecting cut.

Finding $\mathcal{T}$

Uses the so-called (weighted) tree packing, building on a theorem by Nash-Williams, and can be implemented in $\mathcal{O}(m + n \log^3 n)$ time whp.

Karger's framework

The high-level structure of Karger's algorithm is as follows:

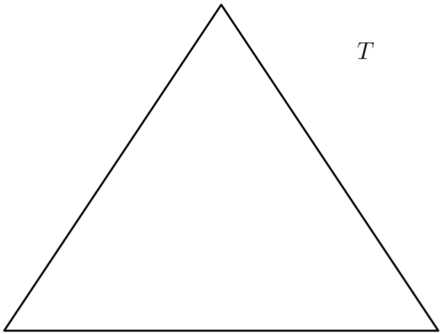
1. Find a collection $\mathcal{T}$ of $\mathcal{O}(\log n)$ trees such that whp the minimum cut 1- or 2-respects some $T \in \mathcal{T}$.
2. For every $T \in \mathcal{T}$, find the minimum 1-respecting cut.
3. For every $T \in \mathcal{T}$, find the minimum 2-respecting cut.

Finding $\mathcal{T}$

Uses the so-called (weighted) tree packing, building on a theorem by Nash-Williams, and can be implemented in $\mathcal{O}(m + n\log^3 n)$ time whp.

Minimum 1-Respecting Cut

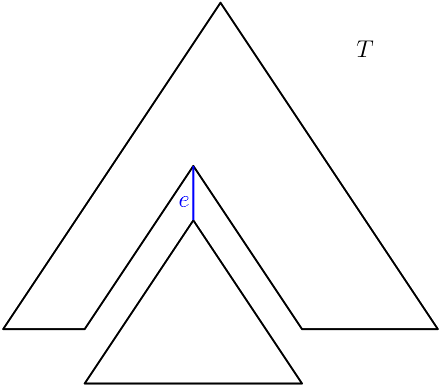
Karger showed that this is actually fairly easy:



This information can be propagated in $\mathcal{O}(m)$ overall time.

Minimum 1-Respecting Cut

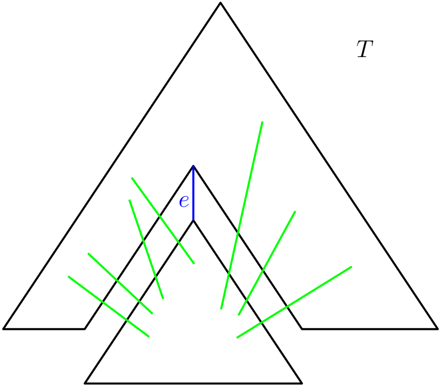
Karger showed that this is actually fairly easy:



This information can be propagated in $\mathcal{O}(m)$ overall time.

Minimum 1-Respecting Cut

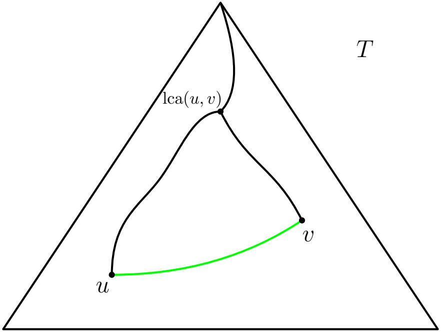
Karger showed that this is actually fairly easy:



This information can be propagated in $\mathcal{O}(m)$ overall time.

Minimum 1-Respecting Cut

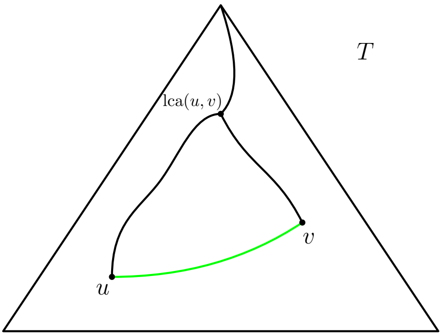
Karger showed that this is actually fairly easy:



This information can be propagated in $\mathcal{O}(m)$ overall time.

Minimum 1-Respecting Cut

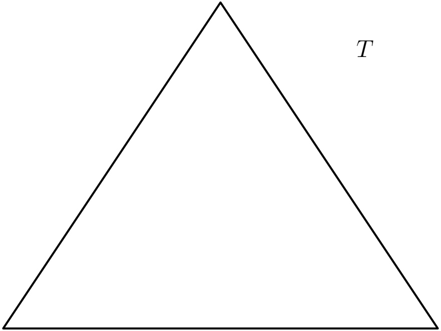
Karger showed that this is actually fairly easy:



This information can be propagated in $\mathcal{O}(m)$ overall time.

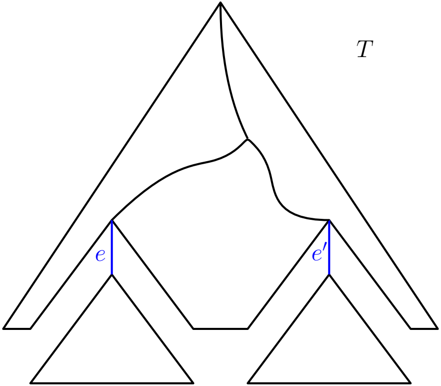
Minimum 2-Respecting Cut

Sought e and e' can either be independent or descendant.



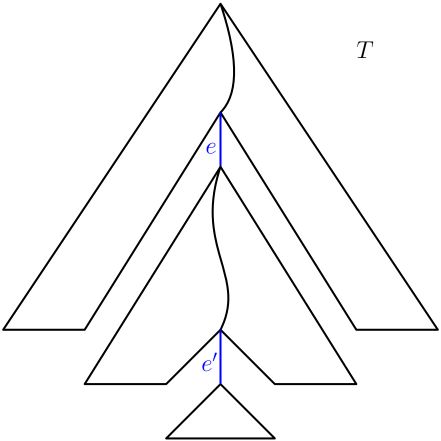
Minimum 2-Respecting Cut

Sought e and e' can either be independent or descendant.



Minimum 2-Respecting Cut

Sought e and e' can either be independent or descendant.



Minimum 2-Respecting Cut

For both cases, Karger's algorithm operates in $\mathcal{O}(\log n)$ phases that iteratively contracting paths (called *boughs*) consisting of nodes with exactly one child terminating at a leaf. Each phase is implemented with link-cut trees in $\mathcal{O}(m \log n)$ time.

Karger 1996

Minimum 2-respecting cut can be found in $\mathcal{O}(m \log^2 n)$ time.

The overall time complexity is $\mathcal{O}(m \log^3 n + n \log^3 n) = \mathcal{O}(m \log^3 n)$.

Minimum 2-Respecting Cut

For both cases, Karger's algorithm operates in $\mathcal{O}(\log n)$ phases that iteratively contracting paths (called *boughs*) consisting of nodes with exactly one child terminating at a leaf. Each phase is implemented with link-cut trees in $\mathcal{O}(m \log n)$ time.

Karger 1996

Minimum 2-respecting cut can be found in $\mathcal{O}(m \log^2 n)$ time.

The overall time complexity is $\mathcal{O}(m \log^3 n + n \log^3 n) = \mathcal{O}(m \log^3 n)$.

Minimum 2-Respecting Cut

For both cases, Karger's algorithm operates in $\mathcal{O}(\log n)$ phases that iteratively contracting paths (called *boughs*) consisting of nodes with exactly one child terminating at a leaf. Each phase is implemented with link-cut trees in $\mathcal{O}(m \log n)$ time.

Karger 1996

Minimum 2-respecting cut can be found in $\mathcal{O}(m \log^2 n)$ time.

The overall time complexity is $\mathcal{O}(m \log^3 n + n \log^3 n) = \mathcal{O}(m \log^3 n)$.

Recent Developments

Henzinger, Rao, and Wang 2017

A faster $\mathcal{O}(m \log^2 n (\log \log n)^2)$ deterministic time algorithm for simple unweighted graphs.

Ghaffari, Nowicki and Thorup 2020

An even faster $\mathcal{O}(\min\{m + n \log^3 n, m \log n\})$ randomised time algorithm for simple unweighted graphs.

Bhardwaj, Lovett, and Sandlund 2020

A different take on the minimum 2-respecting cut based on top trees instead of link-cut trees, and iterating over the edges of T guided by heavy-light decomposition. Also $\mathcal{O}(m \log^2 n)$ time, but simpler.

Recent Developments

Henzinger, Rao, and Wang 2017

A faster $\mathcal{O}(m \log^2 n (\log \log n)^2)$ deterministic time algorithm for simple unweighted graphs.

Ghaffari, Nowicki and Thorup 2020

An even faster $\mathcal{O}(\min\{m + n \log^3 n, m \log n\})$ randomised time algorithm for simple unweighted graphs.

Bhardwaj, Lovett, and Sandlund 2020

A different take on the minimum 2-respecting cut based on top trees instead of link-cut trees, and iterating over the edges of T guided by heavy-light decomposition. Also $\mathcal{O}(m \log^2 n)$ time, but simpler.

Recent Developments

Henzinger, Rao, and Wang 2017

A faster $\mathcal{O}(m \log^2 n (\log \log n)^2)$ deterministic time algorithm for simple unweighted graphs.

Ghaffari, Nowicki and Thorup 2020

An even faster $\mathcal{O}(\min\{m + n \log^3 n, m \log n\})$ randomised time algorithm for simple unweighted graphs.

Bhardwaj, Lovett, and Sandlund 2020

A different take on the minimum 2-respecting cut based on top trees instead of link-cut trees, and iterating over the edges of T guided by heavy-light decomposition. Also $\mathcal{O}(m \log^2 n)$ time, but simpler.

Recent Developments, Continued

Gawrychowski, Mozes, and Weimann 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n)$ time.

Mukhopadhyay and Nanongkai 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n + n \log^3 n)$ randomised time.

Time complexity of the former dominates that of the latter, but the latter has the following advantages:

1. uses a nice structural property of minimum 2-respecting cut,
2. extends to the cut-query and the streaming model,
3. can be seen as a reduction to 2D orthogonal range counting/sampling.

By plugging in appropriate structures, the time complexity for unweighted multigraphs becomes $\mathcal{O}(m\sqrt{\log n} + n \log^4 n)$.

Recent Developments, Continued

Gawrychowski, Mozes, and Weimann 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n)$ time.

Mukhopadhyay and Nanongkai 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n + n \log^3 n)$ randomised time.

Time complexity of the former dominates that of the latter, but the latter has the following advantages:

1. uses a nice structural property of minimum 2-respecting cut,
2. extends to the cut-query and the streaming model,
3. can be seen as a reduction to 2D orthogonal range counting/sampling.

By plugging in appropriate structures, the time complexity for unweighted multigraphs becomes $\mathcal{O}(m\sqrt{\log n} + n \log^4 n)$.

Recent Developments, Continued

Gawrychowski, Mozes, and Weimann 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n)$ time.

Mukhopadhyay and Nanongkai 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n + n \log^3 n)$ randomised time.

Time complexity of the former dominates that of the latter, but the latter has the following advantages:

1. uses a nice structural property of minimum 2-respecting cut,
2. extends to the cut-query and the streaming model,
3. can be seen as a reduction to 2D orthogonal range counting/sampling.

By plugging in appropriate structures, the time complexity for unweighted multigraphs becomes $\mathcal{O}(m\sqrt{\log n} + n \log^4 n)$.

Recent Developments, Continued

Gawrychowski, Mozes, and Weimann 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n)$ time.

Mukhopadhyay and Nanongkai 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n + n \log^3 n)$ randomised time.

Time complexity of the former dominates that of the latter, but the latter has the following advantages:

1. uses a nice structural property of minimum 2-respecting cut,
2. extends to the cut-query and the streaming model,
3. can be seen as a reduction to 2D orthogonal range counting/sampling.

By plugging in appropriate structures, the time complexity for unweighted multigraphs becomes $\mathcal{O}(m\sqrt{\log n} + n \log^4 n)$.

Recent Developments, Continued

Gawrychowski, Mozes, and Weimann 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n)$ time.

Mukhopadhyay and Nanongkai 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n + n \log^3 n)$ randomised time.

Time complexity of the former dominates that of the latter, but the latter has the following advantages:

1. uses a nice structural property of minimum 2-respecting cut,
2. extends to the cut-query and the streaming model,
3. can be seen as a reduction to 2D orthogonal range counting/sampling.

By plugging in appropriate structures, the time complexity for unweighted multigraphs becomes $\mathcal{O}(m\sqrt{\log n} + n \log^4 n)$.

Recent Developments, Continued

Gawrychowski, Mozes, and Weimann 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n)$ time.

Mukhopadhyay and Nanongkai 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n + n \log^3 n)$ randomised time.

Time complexity of the former dominates that of the latter, but the latter has the following advantages:

1. uses a nice structural property of minimum 2-respecting cut,
2. extends to the cut-query and the streaming model,
3. can be seen as a reduction to 2D orthogonal range counting/sampling.

By plugging in appropriate structures, the time complexity for unweighted multigraphs becomes $\mathcal{O}(m\sqrt{\log n} + n \log^4 n)$.

Recent Developments, Continued

Gawrychowski, Mozes, and Weimann 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n)$ time.

Mukhopadhyay and Nanongkai 2020

Minimum 2-respecting cut in $\mathcal{O}(m \log n + n \log^3 n)$ randomised time.

Time complexity of the former dominates that of the latter, but the latter has the following advantages:

1. uses a nice structural property of minimum 2-respecting cut,
2. extends to the cut-query and the streaming model,
3. can be seen as a reduction to 2D orthogonal range counting/sampling.

By plugging in appropriate structures, the time complexity for unweighted multigraphs becomes $\mathcal{O}(m\sqrt{\log n} + n \log^4 n)$.

Our Contribution

We simplify and streamline the minimum 2-respecting cut algorithm Mukhopadhyay and Nanongkai to work in only $\mathcal{O}(m \log n + n \log^2 n)$ deterministic time.

Our implementation can be seen as a reduction to just 2D orthogonal counting. This allows us to obtain the following new bounds for the mimimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs, for any $\epsilon > 0$.

Our Contribution

We simplify and streamline the minimum 2-respecting cut algorithm Mukhopadhyay and Nanongkai to work in only $\mathcal{O}(m \log n + n \log^2 n)$ deterministic time.

Our implementation can be seen as a reduction to just 2D orthogonal counting. This allows us to obtain the following new bounds for the mimimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs, for any $\epsilon > 0$.

Our Contribution

We simplify and streamline the minimum 2-respecting cut algorithm Mukhopadhyay and Nanongkai to work in only $\mathcal{O}(m \log n + n \log^2 n)$ deterministic time.

Our implementation can be seen as a reduction to just 2D orthogonal counting. This allows us to obtain the following new bounds for the mimimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs, for any $\epsilon > 0$.

Our Contribution

We simplify and streamline the minimum 2-respecting cut algorithm Mukhopadhyay and Nanongkai to work in only $\mathcal{O}(m \log n + n \log^2 n)$ deterministic time.

Our implementation can be seen as a reduction to just 2D orthogonal counting. This allows us to obtain the following new bounds for the mimimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs, for any $\epsilon > 0$.

Mukhopadhyay and Nanongkai's framework

Partition the edges of T into edge-disjoint heavy paths such that any root-to-leaf path intersects with at most $\log n$ heavy paths.

Now we have two cases:

1. e, e' belong to the same heavy path,
2. e, e' belong to different heavy paths.

Mukhopadhyay and Nanongkai's framework

Partition the edges of T into edge-disjoint heavy paths such that any root-to-leaf path intersects with at most $\log n$ heavy paths.

Now we have two cases:

1. e, e' belong to the same heavy path,
2. e, e' belong to different heavy paths.

Mukhopadhyay and Nanongkai's framework

Partition the edges of T into edge-disjoint heavy paths such that any root-to-leaf path intersects with at most $\log n$ heavy paths.

Now we have two cases:

1. e, e' belong to the same heavy path,
2. e, e' belong to different heavy paths.

Mukhopadhyay and Nanongkai's framework

Partition the edges of T into edge-disjoint heavy paths such that any root-to-leaf path intersects with at most $\log n$ heavy paths.

Now we have two cases:

1. e, e' belong to the same heavy path,
2. e, e' belong to different heavy paths.

e, e' belong to the same heavy path

Let $M[i,j]$ be the weight of the cut determined by e_i and e_j.

M is a partial Monge matrix

For any $i \neq j$, $M[i,j] - M[i,j+1] \geq M[i+1,j] - M[i+1,j+1]$.

Klawe and Kleitman showed how to find the minimum in such an array in $\mathcal{O}(\ell \cdot \alpha(\ell))$ inspections, where ℓ is the length of the path. This sums to $\mathcal{O}(n \cdot \alpha(n))$ inspections.

e, e' belong to the same heavy path

Let $M[i,j]$ be the weight of the cut determined by e_i and e_j.

M is a partial Monge matrix

For any $i \neq j$, $M[i,j] - M[i,j+1] \geq M[i+1,j] - M[i+1,j+1]$.

Klawe and Kleitman showed how to find the minimum in such an array in $\mathcal{O}(\ell \cdot \alpha(\ell))$ inspections, where ℓ is the length of the path. This sums to $\mathcal{O}(n \cdot \alpha(n))$ inspections.

e, e' belong to the same heavy path

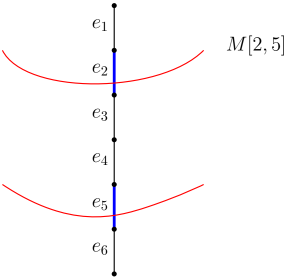
Let $M[i, j]$ be the weight of the cut determined by e_i and e_j.

M is a partial Monge matrix

For any $i \neq j$, $M[i, j] - M[i, j+1] \geq M[i+1, j] - M[i+1, j+1]$.

Klawe and Kleitman showed how to find the minimum in such an array in $\mathcal{O}(\ell \cdot \alpha(\ell))$ inspections, where ℓ is the length of the path. This sums to $\mathcal{O}(n \cdot \alpha(n))$ inspections.

e, e' belong to the same heavy path



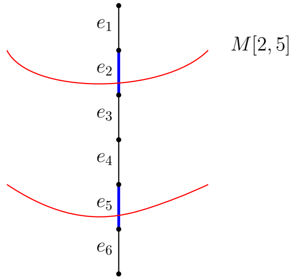
Let $M[i,j]$ be the weight of the cut determined by e_i and e_j.

M is a partial Monge matrix

For any $i \neq j$, $M[i,j] - M[i,j+1] \geq M[i+1,j] - M[i+1,j+1]$.

Klawe and Kleitman showed how to find the minimum in such an array in $\mathcal{O}(\ell \cdot \alpha(\ell))$ inspections, where ℓ is the length of the path. This sums to $\mathcal{O}(n \cdot \alpha(n))$ inspections.

e, e' belong to the same heavy path



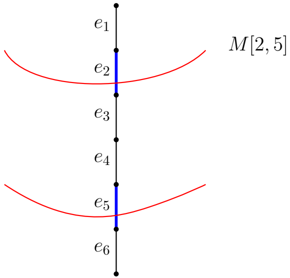
Let $M[i,j]$ be the weight of the cut determined by e_i and e_j.

M is a partial Monge matrix

For any $i \neq j$, $M[i,j] - M[i,j+1] \geq M[i+1,j] - M[i+1,j+1]$.

Klawe and Kleitman showed how to find the minimum in such an array in $\mathcal{O}(\ell \cdot \alpha(\ell))$ inspections, where ℓ is the length of the path. This sums to $\mathcal{O}(n \cdot \alpha(n))$ inspections.

e, e' belong to the same heavy path



Let $M[i,j]$ be the weight of the cut determined by e_i and e_j.

M is a partial Monge matrix

For any $i \neq j$, $M[i,j] - M[i,j+1] \geq M[i+1,j] - M[i+1,j+1]$.

Klawe and Kleitman showed how to find the minimum in such an array in $\mathcal{O}(\ell \cdot \alpha(\ell))$ inspections, where ℓ is the length of the path. This sums to $\mathcal{O}(n \cdot \alpha(n))$ inspections.

e, e' belong to different heavy paths

One could similarly form a Monge matrix for every pair of heavy paths. However, this would be too slow.

All such edges e' form a single path from the root to some node c_e.

If the minimum cut is determined by independent edges e, e' then e is cross-interested in e' and vice versa.

e, e' belong to different heavy paths

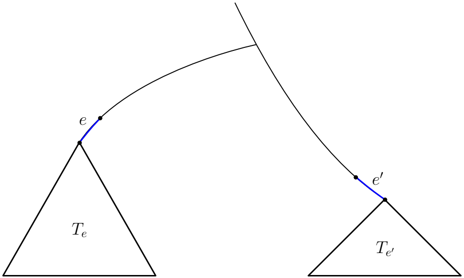
An edge e is cross-interested in an edge $e' \notin T_e$ if more than half of the edge weight going out T_e goes into $T_{e'}$.

All such edges e' form a single path from the root to some node c_e.

If the minimum cut is determined by independent edges e, e' then e is cross-interested in e' and vice versa.

e, e' belong to different heavy paths

An edge e is cross-interested in an edge $e' \notin T_e$ if more than half of the edge weight going out T_e goes into $T_{e'}$.

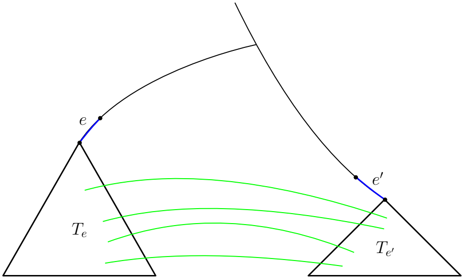


All such edges e' form a single path from the root to some node c_e.

If the minimum cut is determined by independent edges e, e' then e is cross-interested in e' and vice versa.

e, e' belong to different heavy paths

An edge e is cross-interested in an edge $e' \notin T_e$ if more than half of the edge weight going out T_e goes into $T_{e'}$.

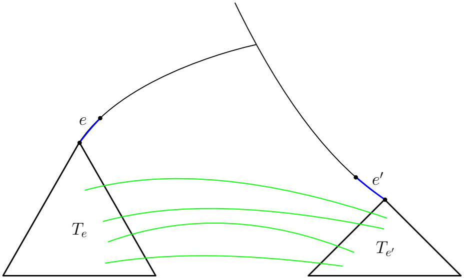


All such edges e' form a single path from the root to some node c_e.

If the minimum cut is determined by independent edges e, e' then e is cross-interested in e' and vice versa.

e, e' belong to different heavy paths

An edge e is cross-interested in an edge $e' \notin T_e$ if more than half of the edge weight going out T_e goes into $T_{e'}$.

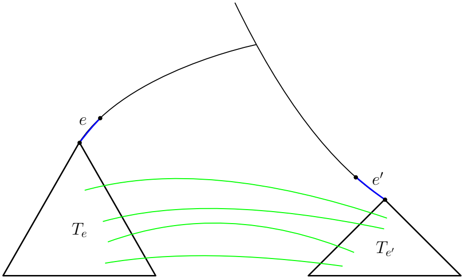


All such edges e' form a single path from the root to some node c_e.

If the minimum cut is determined by independent edges e, e' then e is cross-interested in e' and vice versa.

e, e' belong to different heavy paths

An edge e is cross-interested in an edge $e' \notin T_e$ if more than half of the edge weight going out T_e goes into $T_{e'}$.



All such edges e' form a single path from the root to some node c_e.

If the minimum cut is determined by independent edges e, e' then e is cross-interested in e' and vice versa.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P$, $e' \in Q$, Q is interesting for e and P is interesting for e':
    1. extract $P' \subseteq P$ consisting of edges interested in Q,
    2. extract $Q' \subseteq Q$ consisting of edges interested in P,
    3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P, e' \in Q$, Q is interesting for e and P is interesting for e':
   1. extract $P' \subseteq P$ consisting of edges interested in Q,
   2. extract $Q' \subseteq Q$ consisting of edges interested in P,
   3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P, e' \in Q$, Q is interesting for e and P is interesting for e':
    1. extract $P' \subseteq P$ consisting of edges interested in Q,
    2. extract $Q' \subseteq Q$ consisting of edges interested in P,
    3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P, e' \in Q$, Q is interesting for e and P is interesting for e':
   1. extract $P' \subseteq P$ consisting of edges interested in Q,
   2. extract $Q' \subseteq Q$ consisting of edges interested in P,
   3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P, e' \in Q$, Q is interesting for e and P is interesting for e':
   1. extract $P' \subseteq P$ consisting of edges interested in Q,
   2. extract $Q' \subseteq Q$ consisting of edges interested in P,
   3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P, e' \in Q$, Q is interesting for e and P is interesting for e':
   1. extract $P' \subseteq P$ consisting of edges interested in Q,
   2. extract $Q' \subseteq Q$ consisting of edges interested in P,
   3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P$, $e' \in Q$, Q is interesting for e and P is interesting for e':
   1. extract $P' \subseteq P$ consisting of edges interested in Q,
   2. extract $Q' \subseteq Q$ consisting of edges interested in P,
   3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P$, $e' \in Q$, Q is interesting for e and P is interesting for e':
    1. extract $P' \subseteq P$ consisting of edges interested in Q,
    2. extract $Q' \subseteq Q$ consisting of edges interested in P,
    3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

e, e' belong to different heavy paths

Similar notion for the case of descendant edges e, e'.

High-level structure of the algorithm:

1. Identify c_e for every e.
2. For every e, identify $\mathcal{O}(\log n)$ interesting heavy paths containing cross-interesting edges e'.
3. For every pair of heavy paths P, Q such that, for some edges $e \in P, e' \in Q$, Q is interesting for e and P is interesting for e':
   1. extract $P' \subseteq P$ consisting of edges interested in Q,
   2. extract $Q' \subseteq Q$ consisting of edges interested in P,
   3. form a $|P'| \times |Q'|$ Monge matrix and find the minimum with SMAWK using $\mathcal{O}(|P'| + |Q'|)$ inspections.

We need $\sum(|P'| + |Q'|) = \mathcal{O}(n \log n)$ inspections, plus $\mathcal{O}(n \log^2 n)$ time for the bookkeeping, assuming that we know c_e for every e.

Preprocessing

To check if e is cross-interested in e', or to compute the total weight of the cut determined by e, e', we use the following tool:

Chazelle 1988

Collection of N weighted points can be preprocesses in $\mathcal{O}(N \log N)$ time and space, so that the total weight of all points in any axis-aligned rectangle can be computed in $\mathcal{O}(\log N)$ time.

We identify the nodes with their visiting time in the postorder traversal of T. Then, every edge (u, v) naturally becomes a weighted point in the plane. We preprocess them in $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$ time.

Both queries translate into a constant number of queries about the total weight of points in axis-aligned rectangles.

Preprocessing

To check if e is cross-interested in e', or to compute the total weight of the cut determined by e, e', we use the following tool:

Chazelle 1988

Collection of N weighted points can be preprocesses in $\mathcal{O}(N \log N)$ time and space, so that the total weight of all points in any axis-aligned rectangle can be computed in $\mathcal{O}(\log N)$ time.

We identify the nodes with their visiting time in the postorder traversal of T. Then, every edge (u, v) naturally becomes a weighted point in the plane. We preprocess them in $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$ time.

Both queries translate into a constant number of queries about the total weight of points in axis-aligned rectangles.

Preprocessing

To check if e is cross-interested in e', or to compute the total weight of the cut determined by e, e', we use the following tool:

Chazelle 1988

Collection of N weighted points can be preprocesses in $\mathcal{O}(N \log N)$ time and space, so that the total weight of all points in any axis-aligned rectangle can be computed in $\mathcal{O}(\log N)$ time.

We identify the nodes with their visiting time in the postorder traversal of T. Then, every edge (u, v) naturally becomes a weighted point in the plane. We preprocess them in $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$ time.

Both queries translate into a constant number of queries about the total weight of points in axis-aligned rectangles.

Finding c_e for every e

Recall that all edges e' cross-interesting for e form a path from the root to c_e, and we have a mechanism for checking in $\mathcal{O}(\log n)$ time if a given edge e' is cross-interesting for e. Instead of random sampling, we use the following tool to identify c_e:

Centroid decomposition

Choose a centroid node $v \in T$ such that every connected component of $T \setminus \{v\}$ consists of at most $|T|/2$ nodes. Recurse on the connected components of $T \setminus \{v\}$.

Centroid decomposition of T can be constructed in $\mathcal{O}(n \log n)$ time.

Finding c_e for every e

Recall that all edges e' cross-interesting for e form a path from the root to c_e, and we have a mechanism for checking in $\mathcal{O}(\log n)$ time if a given edge e' is cross-interesting for e. Instead of random sampling, we use the following tool to identify c_e:

Centroid decomposition

Choose a centroid node $v \in T$ such that every connected component of $T \setminus \{v\}$ consists of at most $|T|/2$ nodes. Recurse on the connected components of $T \setminus \{v\}$.

Centroid decomposition of T can be constructed in $\mathcal{O}(n \log n)$ time.

Finding c_e for every e

Recall that all edges e' cross-interesting for e form a path from the root to c_e, and we have a mechanism for checking in $\mathcal{O}(\log n)$ time if a given edge e' is cross-interesting for e. Instead of random sampling, we use the following tool to identify c_e:

Centroid decomposition

Choose a centroid node $v \in T$ such that every connected component of $T \setminus \{v\}$ consists of at most $|T|/2$ nodes. Recurse on the connected components of $T \setminus \{v\}$.

Centroid decomposition of T can be constructed in $\mathcal{O}(n \log n)$ time.

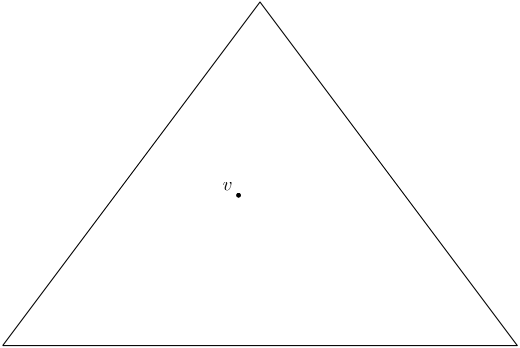
Finding c_e for every e

Let T be binary (otherwise, replace high-degree nodes with small binary trees). To find c_e, traverse the centroid decomposition of T:

1. Check if e_1, e_2, e_3 are cross-interesting for e.
2. Continue in the appropriate connected component of $T \setminus \{v\}$.
3. $\log n$ steps, each in $\mathcal{O}(\log n)$ time.

Finding c_e for every e

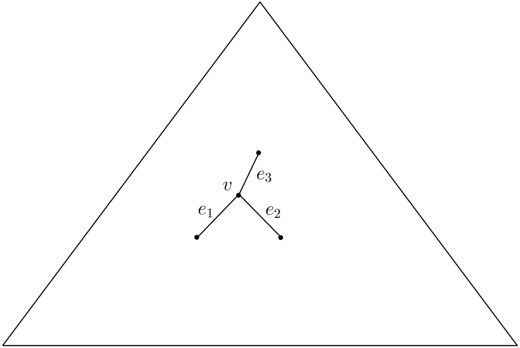
Let T be binary (otherwise, replace high-degree nodes with small binary trees). To find c_e, traverse the centroid decomposition of T:



1. Check if e_1, e_2, e_3 are cross-interesting for e.
2. Continue in the appropriate connected component of $T \setminus \{v\}$.
3. $\log n$ steps, each in $\mathcal{O}(\log n)$ time.

Finding c_e for every e

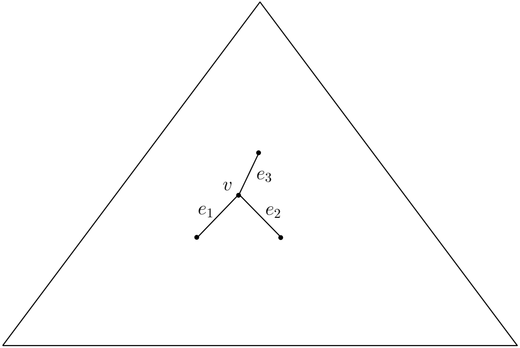
Let T be binary (otherwise, replace high-degree nodes with small binary trees). To find c_e, traverse the centroid decomposition of T:



1. Check if e_1, e_2, e_3 are cross-interesting for e.
2. Continue in the appropriate connected component of $T \setminus \{v\}$.
3. $\log n$ steps, each in $\mathcal{O}(\log n)$ time.

Finding c_e for every e

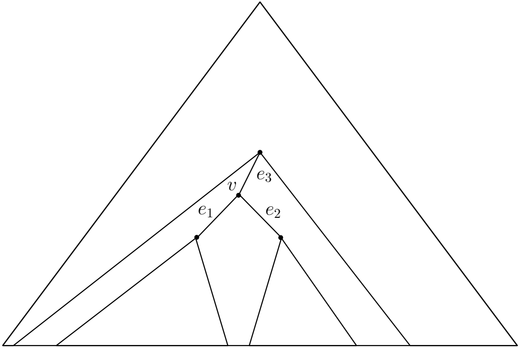
Let T be binary (otherwise, replace high-degree nodes with small binary trees). To find c_e, traverse the centroid decomposition of T:



1. Check if e_1, e_2, e_3 are cross-interesting for e.
2. Continue in the appropriate connected component of $T \setminus \{v\}$.
3. $\log n$ steps, each in $\mathcal{O}(\log n)$ time.

Finding c_e for every e

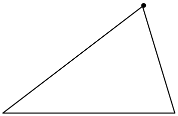
Let T be binary (otherwise, replace high-degree nodes with small binary trees). To find c_e, traverse the centroid decomposition of T:



1. Check if e_1, e_2, e_3 are cross-interesting for e.
2. Continue in the appropriate connected component of $T \setminus \{v\}$.
3. $\log n$ steps, each in $\mathcal{O}(\log n)$ time.

Finding c_e for every e

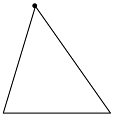
Let T be binary (otherwise, replace high-degree nodes with small binary trees). To find c_e, traverse the centroid decomposition of T:



1. Check if e_1, e_2, e_3 are cross-interesting for e.
2. Continue in the appropriate connected component of $T \setminus \{v\}$.
3. $\log n$ steps, each in $\mathcal{O}(\log n)$ time.

Finding c_e for every e

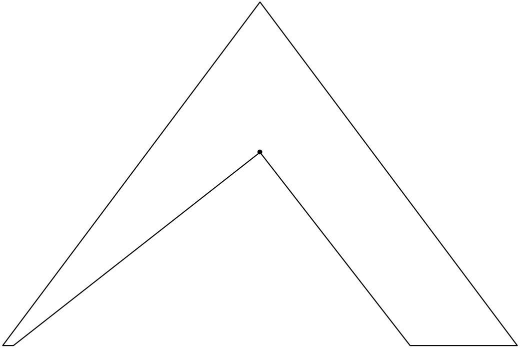
Let T be binary (otherwise, replace high-degree nodes with small binary trees). To find c_e, traverse the centroid decomposition of T:



1. Check if e_1, e_2, e_3 are cross-interesting for e.
2. Continue in the appropriate connected component of $T \setminus \{v\}$.
3. $\log n$ steps, each in $\mathcal{O}(\log n)$ time.

Finding c_e for every e

Let T be binary (otherwise, replace high-degree nodes with small binary trees). To find c_e, traverse the centroid decomposition of T:



1. Check if e_1, e_2, e_3 are cross-interesting for e.
2. Continue in the appropriate connected component of $T \setminus \{v\}$.
3. $\log n$ steps, each in $\mathcal{O}(\log n)$ time.

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!

The approach of Mukhopadhyay and Nanongkai for finding minimum 2-respecting cut can be implemented in $\mathcal{O}(m \log n + n \log^2 n)$ time (without randomisation).

By plugging in appropriate data structures, we can also obtain the following new results for the minimum cut problem:

1. $\mathcal{O}(m \log^{3/2} n + n \log^3 n)$ for unweighted multigraphs.
2. $\mathcal{O}(m \log n + n^{1+\epsilon})$ for weighted graphs,

1. Is there a way to further simplify this approach to remove the $n \log^2 n$?
2. Are similar speedups possible for dense graphs?
3. Is there a near-linear time deterministic algorithm?

Thank you!