Better Tradeoffs for Exact Distance Oracles in Planar Graphs

Paweł Gawrychowski[1] Shay Mozes[2] Oren Weimann[3]
Christian Wulff-Nilsen[4]

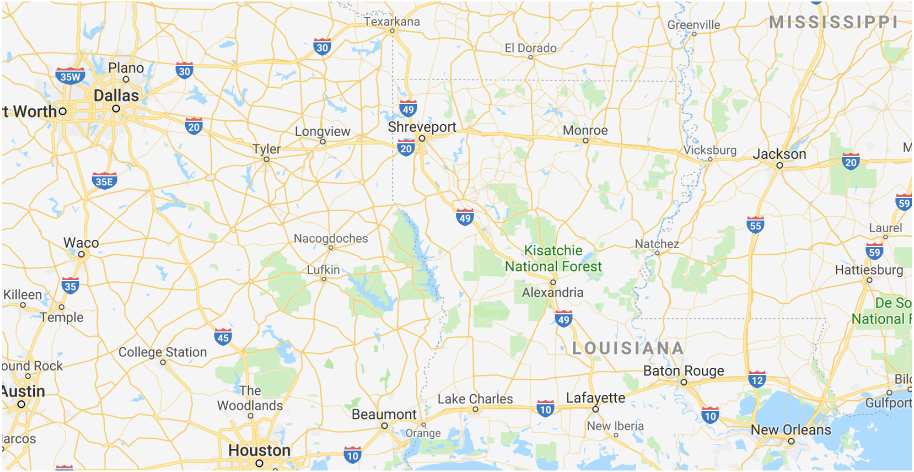
[1]University of Wrocław, Poland

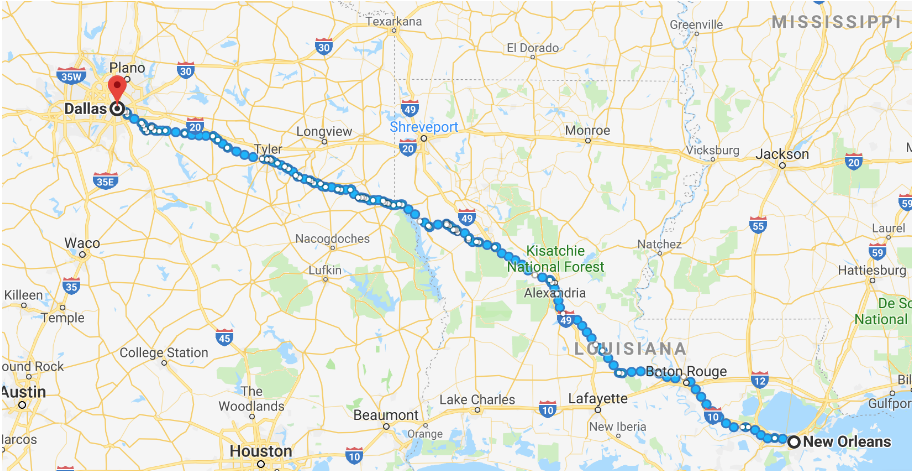
[2]Interdisciplinary Center Herzliya, Israel

[2]University of Haifa, Israel

University of Copenhagen, Denmark

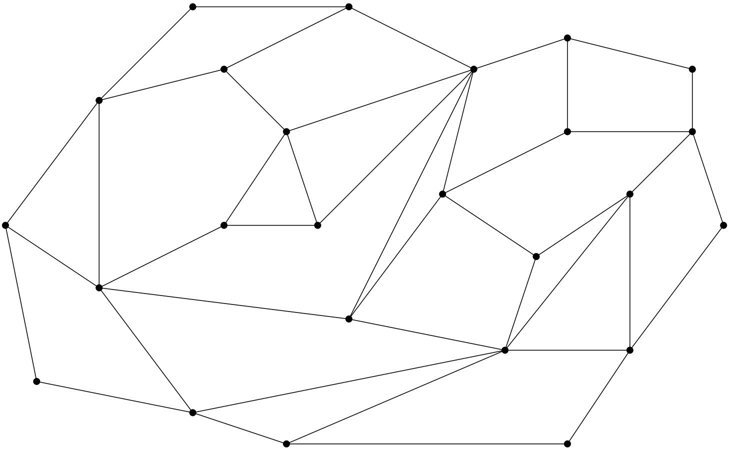
January 7, 2018


Texarkana
El Dorado
Greenville
MISSISSIPPI
Plano
Dallas
t Worth
Longview
Shreveport
Monroe
Vicksburg
Jackson
Tyler
Waco
Nacogdoches
Lufkin
Natchez
Kisatchie
National Forest
Alexandria
Laurel
Hattiesburg
Killeen
Temple
De So
National F
LOUISIANA
College Station
ound Rock
Austin
Baton Rouge
The
Woodlands
Lake Charles
Lafayette
Beaumont
Orange
New Iberia
New Orleans
Gulfport
arcos
Houston



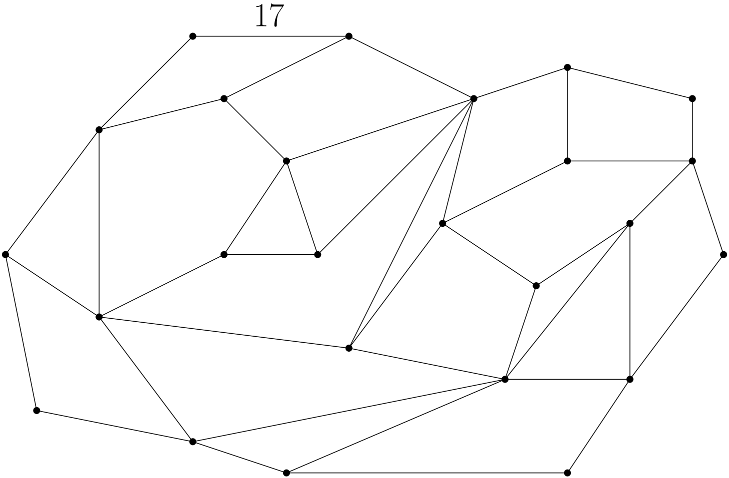
Texarkana
El Dorado
Greenville
MISSISSIPPI
Plano
Dallas
Shreveport
Longview
Monroe
Tyler
Vicksburg
Jackson
Nacogdoches
Natchez
Laurel
Waco
Lufkin
Kisatchie
National Forest
Hattiesburg
Killeen
Alexandria
Temple
LOUISIANA
College Station
Baton Rouge
ound Rock
Austin
The
Woodlands
Lake Charles
Lafayette
Beaumont
Orange
New Iberia
New Orleans
Houston
Gulfpor

Goal



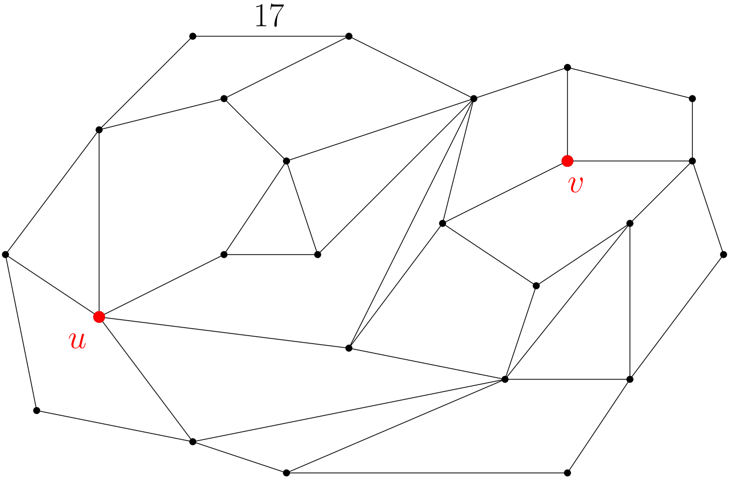
Preprocess an n-vertex planar graph $G = (V, E)$ with nonnegative arc lengths, so that given any $u, v \in V$ we can compute $d(u, v)$ efficiently.

Goal



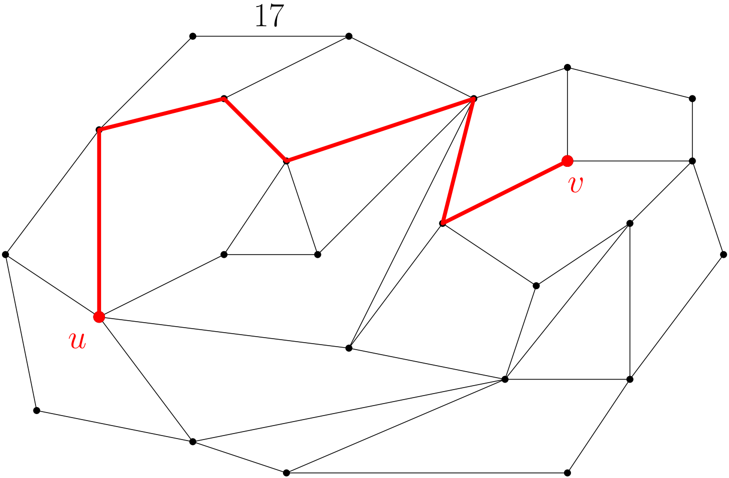
Preprocess an n-vertex planar graph $G = (V, E)$ with nonnegative arc lengths, so that given any $u, v \in V$ we can compute $d(u, v)$ efficiently.

Goal



Preprocess an n-vertex planar graph $G = (V, E)$ with nonnegative arc lengths, so that given any $u, v \in V$ we can compute $d(u, v)$ efficiently.

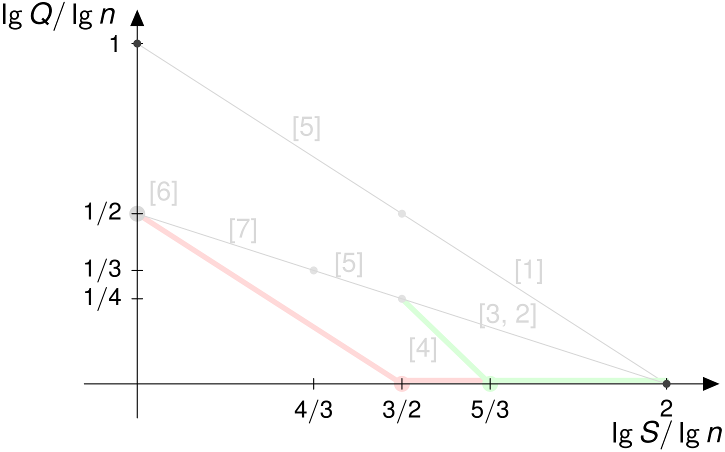
Goal



Preprocess an n-vertex planar graph $G = (V, E)$ with nonnegative arc lengths, so that given any $u, v \in V$ we can compute $d(u, v)$ efficiently.

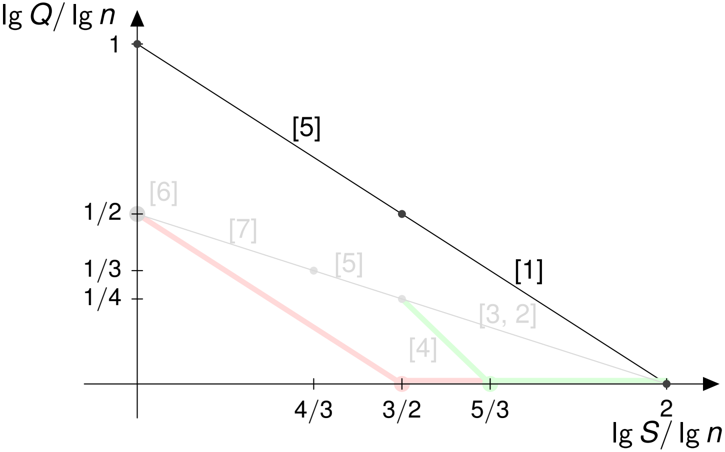
Previous work

The trade-off between the query time Q and the size S of the structure:



Previous work

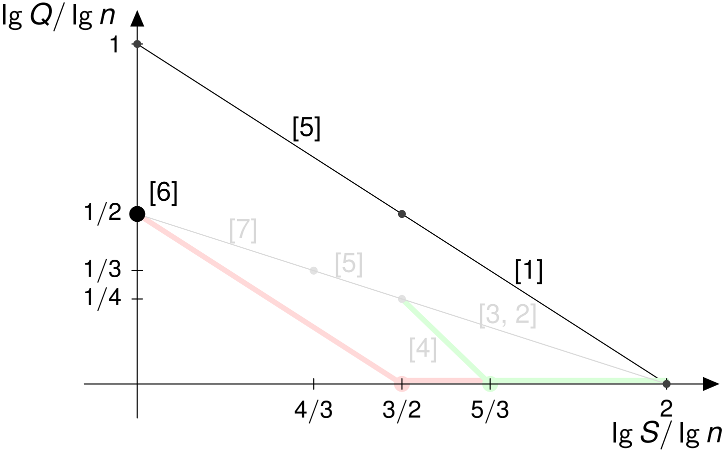
The trade-off between the query time Q and the size S of the structure:



Djidjev [5] and Arikati et al. [1] achieved $Q = O(n^2/S^2)$.

Previous work

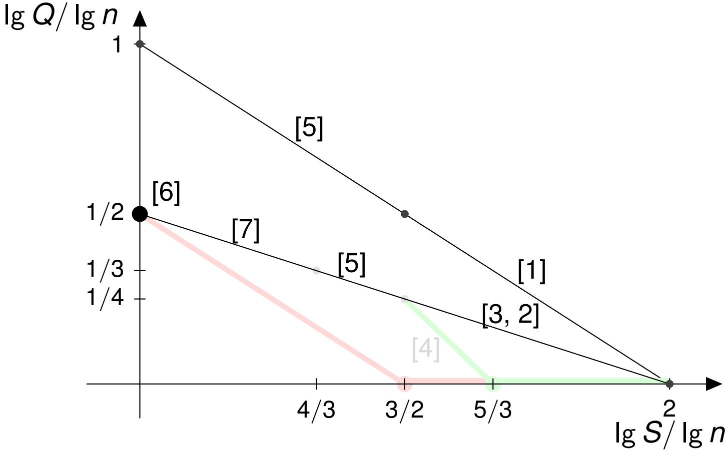
The trade-off between the query time Q and the size S of the structure:



Fakcharoenphol and Rao [6] show that $S = \tilde{O}(n)$ and $Q = \tilde{O}(\sqrt{n})$ is possible.

Previous work

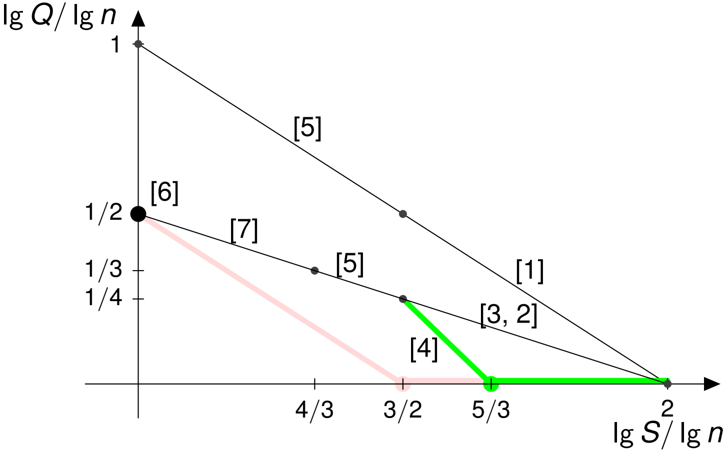
The trade-off between the query time Q and the size S of the structure:



This has been extended to $Q = \tilde{O}(n/\sqrt{S})$ for essentially the whole range of S in a series of papers.

Previous work

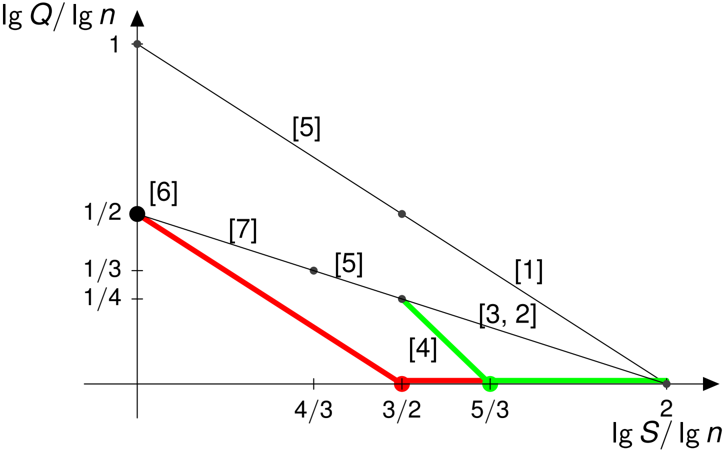
The trade-off between the query time Q and the size S of the structure:



Last year, Cohen-Addad, Dahlgaard, and Wulff-Nilsen [4] showed that this is not optimal, and $S = O(n^{5/3})$ with $Q = O(\log n)$ is possible.

Previous work

The trade-off between the query time Q and the size S of the structure:



We improve this to $S = O(n^{1.5})$ and $Q = O(\log n)$.

Main result

For any $S \in [n, n^2]$, we construct an oracle of size S that answers an exact distance query in $Q = \tilde{O}(\max\{1, n^{1.5}/S\})$ time.

At the heart of the above construction is a structure with $S = O(n^{1.5})$ and $Q = O(\log n)$ that, similarly to the result of Cohen-Addad et al., uses the Voronoi diagram technique introduced by Cabello.

Main result

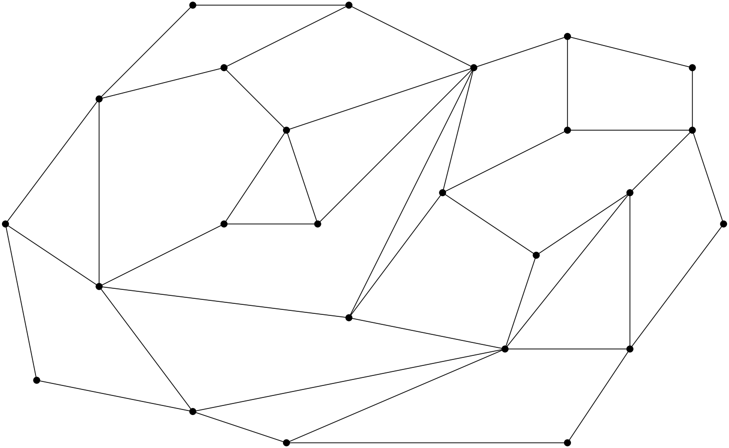
For any $S \in [n, n^2]$, we construct an oracle of size S that answers an exact distance query in $Q = \tilde{O}(\max\{1, n^{1.5}/S\})$ time.

At the heart of the above construction is a structure with $S = O(n^{1.5})$ and $Q = O(\log n)$ that, similarly to the result of Cohen-Addad et al., uses the Voronoi diagram technique introduced by Cabello.

Basic recursion

Miller

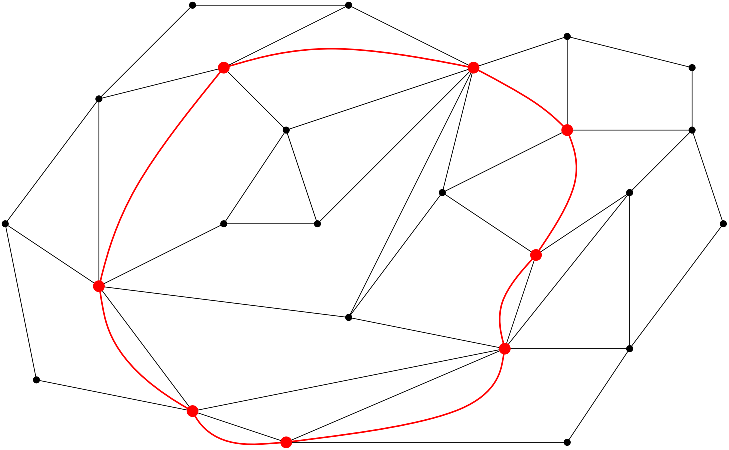
There always exists a Jordan curve separator of size $O(\sqrt{n})$ such that there are at most $\frac{2}{3}n$ nodes on its inside/outside.



Basic recursion

Miller

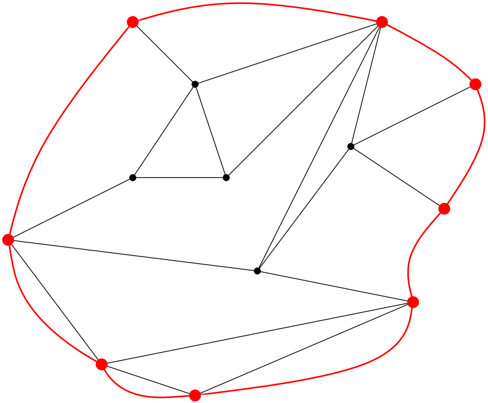
There always exists a Jordan curve separator of size $O(\sqrt{n})$ such that there are at most $\frac{2}{3}n$ nodes on its inside/outside.



Basic recursion

Miller

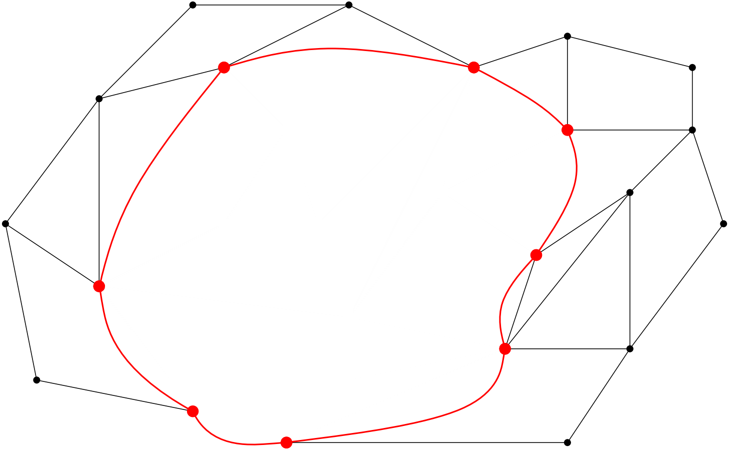
There always exists a Jordan curve separator of size $O(\sqrt{n})$ such that there are at most $\frac{2}{3}n$ nodes on its inside/outside.



Basic recursion

Miller

There always exists a Jordan curve separator of size $O(\sqrt{n})$ such that there are at most $\frac{2}{3}n$ nodes on its inside/outside.



Basic recursion

- Recursively build an exact distance oracle for the inside.
- Recursively build an exact distance oracle for the outside.
- Build a structure that that can be used to find the shortest path from u to v that visits at least one node of the separator.

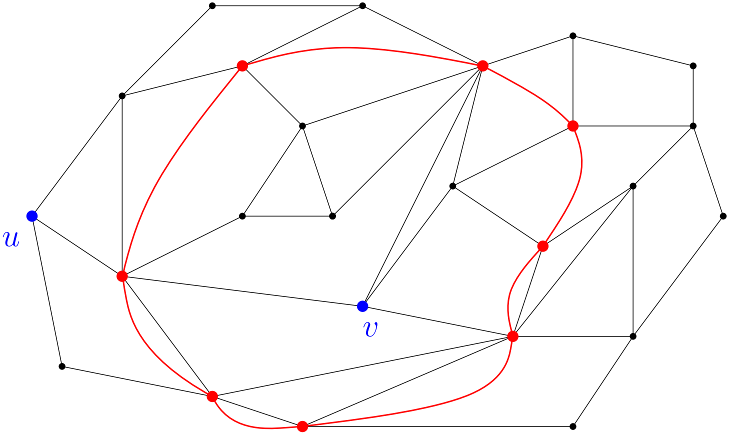
Basic recursion

- Recursively build an exact distance oracle for the inside.
- Recursively build an exact distance oracle for the outside.
- Build a structure that that can be used to find the shortest path from u to v that visits at least one node of the separator.

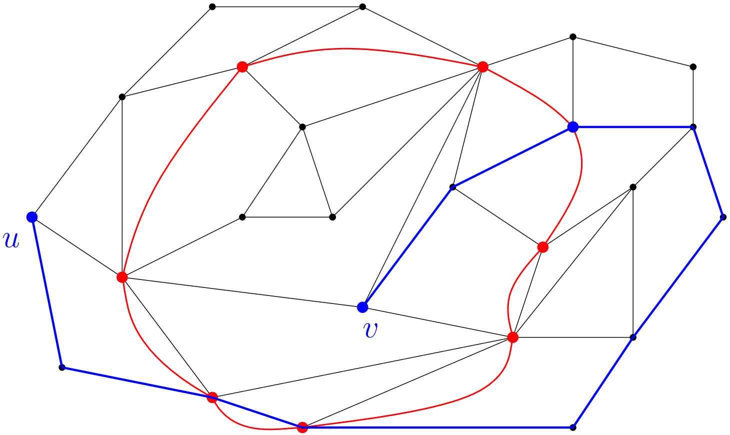
Basic recursion

- Recursively build an exact distance oracle for the inside.
- Recursively build an exact distance oracle for the outside.
- Build a structure that that can be used to find the shortest path from u to v that visits at least one node of the separator.

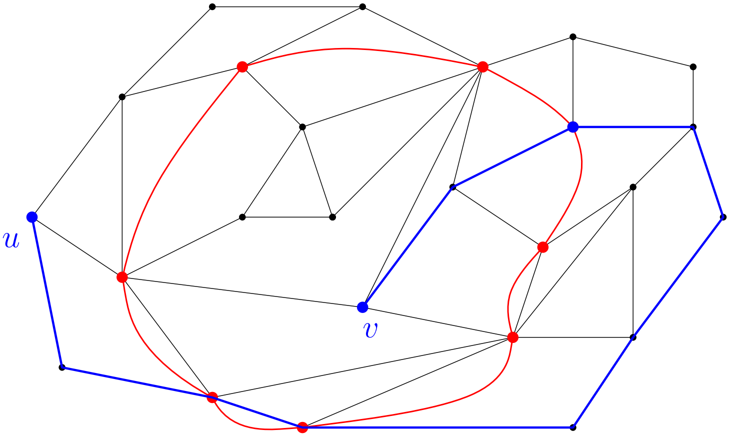
Basic recursion



Basic recursion

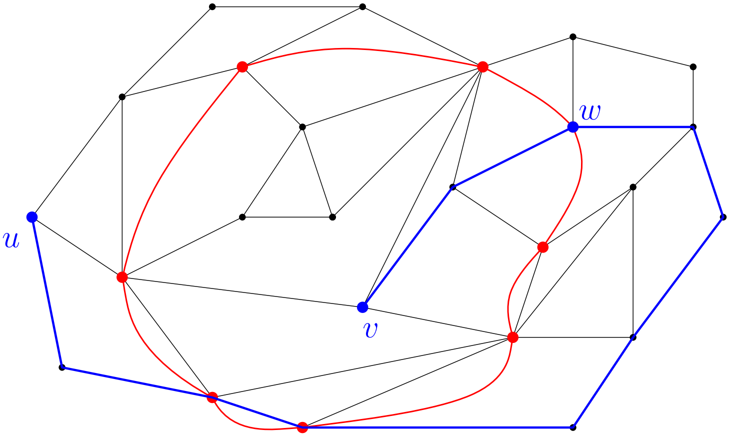


Basic recursion



Shortest path can cross the separator multiple times!

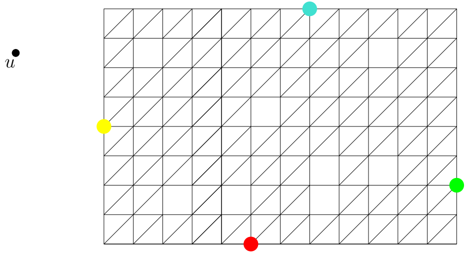
Basic recursion



Find $w \in Sep$ minimising $d_G(u, w) + d_{in}(w, v)$.

Single step

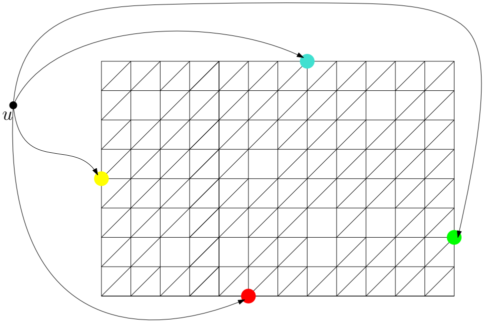
Consider the inside of graph and a fixed node u.



1. For technical reasons, triangulate.
2. For every $w \in Sep$ define $\omega(w) = d_G(u, w)$.
3. Construct the Voronoi diagram.
4. ... finding w reduces to point location!

Single step

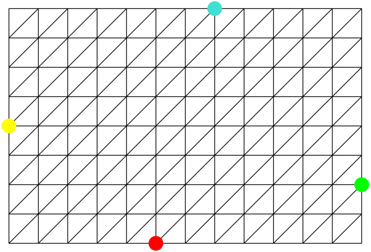
Consider the inside of graph and a fixed node u.



1. For technical reasons, triangulate.
2. For every $w \in Sep$ define $\omega(w) = d_G(u, w)$.
3. Construct the Voronoi diagram.
4. ... finding w reduces to point location!

Single step

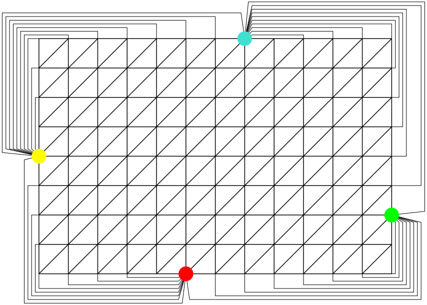
Consider the inside of graph and a fixed node u.



1. For technical reasons, triangulate.
2. For every $w \in Sep$ define $\omega(w) = d_G(u, w)$.
3. Construct the Voronoi diagram.
4. ... finding w reduces to point location!

Single step

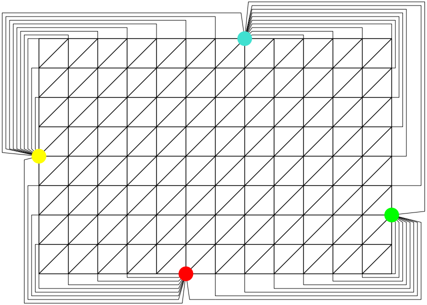
Consider the inside of graph and a fixed node u.



1. For technical reasons, triangulate.
2. For every $w \in Sep$ define $\omega(w) = d_G(u, w)$.
3. Construct the Voronoi diagram.
4. ... finding w reduces to point location!

Single step

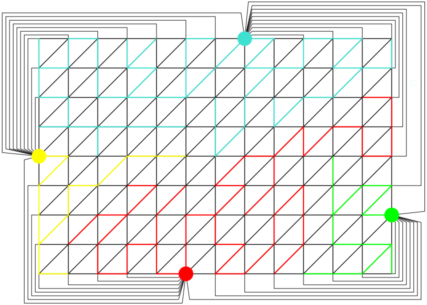
Consider the inside of graph and a fixed node u.



1. For technical reasons, triangulate.
2. For every $w \in Sep$ define $\omega(w) = d_G(u, w)$.
3. Construct the Voronoi diagram.
4. ... finding w reduces to point location!

Single step

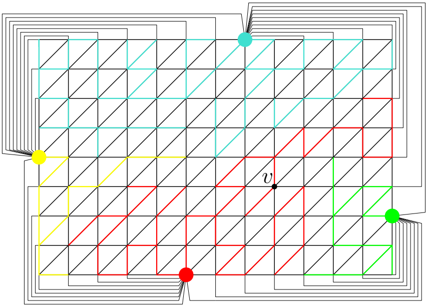
Consider the inside of graph and a fixed node u.



1. For technical reasons, triangulate.
2. For every $w \in Sep$ define $\omega(w) = d_G(u, w)$.
3. Construct the Voronoi diagram.
4. ... finding w reduces to point location!

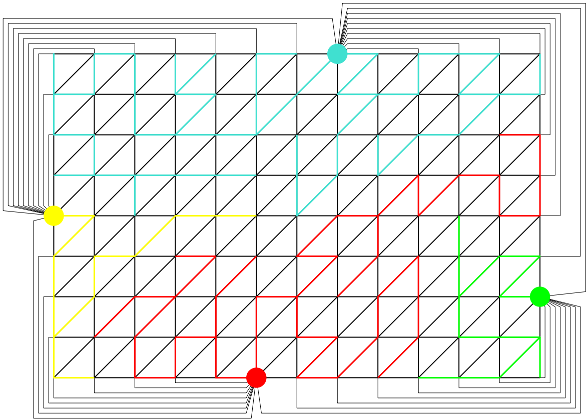
Single step

Consider the inside of graph and a fixed node u.

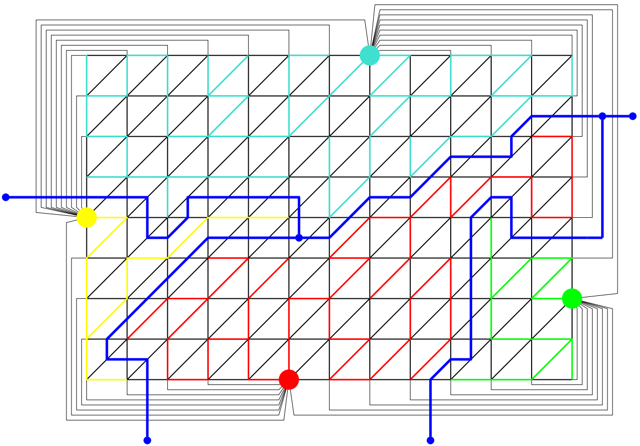


1. For technical reasons, triangulate.
2. For every $w \in Sep$ define $\omega(w) = d_G(u, w)$.
3. Construct the Voronoi diagram.
4. ... finding w reduces to point location!

Voronoi diagram

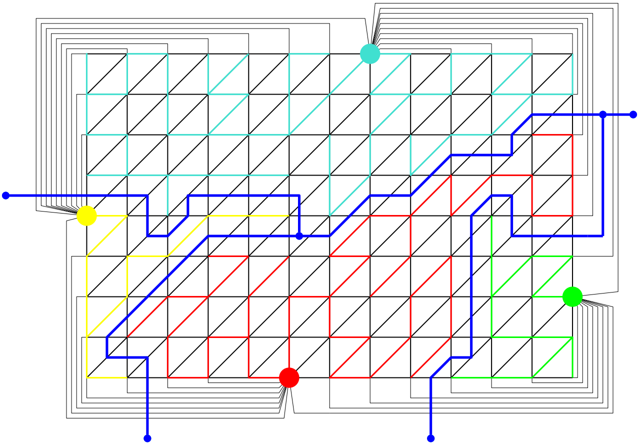


Voronoi diagram



Look at the dual and create many copies of the node corresponding to the external face.

Voronoi diagram



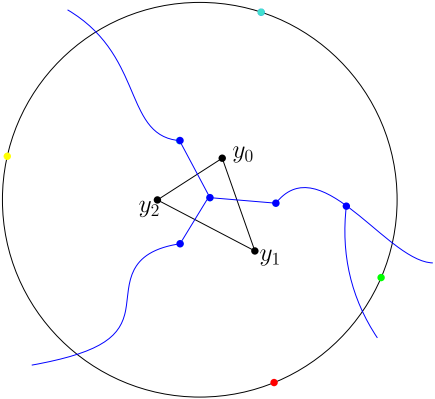
Because all sites are adjacent to the external face, the diagram can be described by a tree on $O(|Sep|)$ nodes.

Point location

Any tree on k nodes contains a centroid node u such that every component of $T \setminus \{u\}$ is of size $\frac{2}{3}k$.

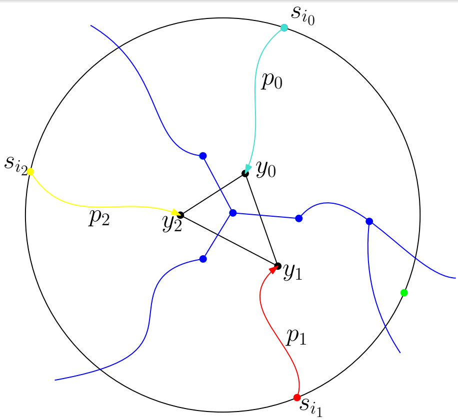
Point location

Any tree on k nodes contains a centroid node u such that every component of $T \setminus \{u\}$ is of size $\frac{2}{3}k$.



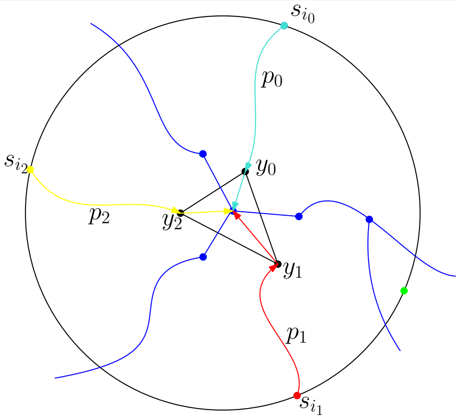
Point location

Any tree on k nodes contains a centroid node u such that every component of $T \setminus \{u\}$ is of size $\frac{2}{3}k$.

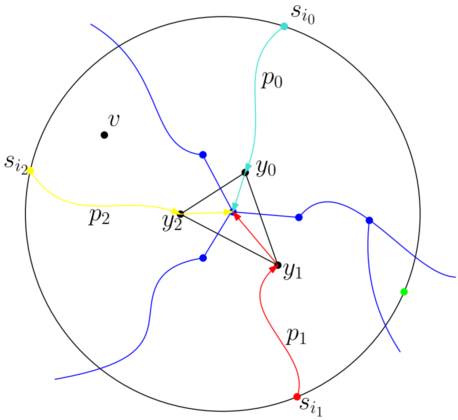


Point location

Any tree on k nodes contains a centroid node u such that every component of $T \setminus \{u\}$ is of size $\frac{2}{3}k$.

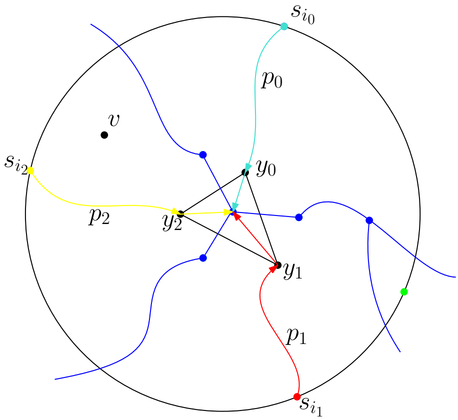


Point location



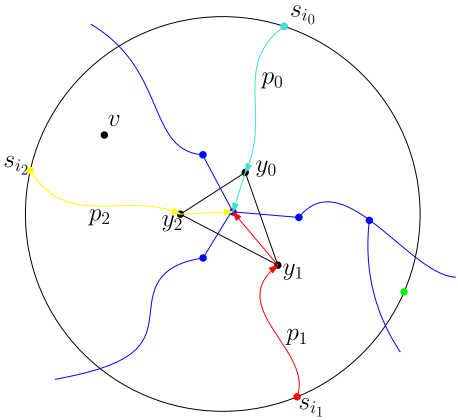
- check that v is on the left of p_2,
- check that v is on the right of p_0,

Point location



- check that v is on the left of p_2,
- check that v is on the right of p_0,

Point location



p_j plus the extra edge is a path in the shortest paths tree rooted at s_{i_j} that doesn't depend on ω.

Point location

- By storing preorder numbers for the shortest paths tree rooted at s_{i_j} we can check if v is on the left/right of p_{i_j} in constant time.
- This allows us to detect in constant time the relevant smaller component of the tree representing the Voronoi diagram that needs to be recursively searched for the cell of v (if it is not the cell of any s_{i_j}).

Time: $O(\log n)$

Space:
- $|Sep| \cdot O(n)$ to store the shortest path tree for every site,
- $n \cdot O(\sqrt{n})$ to store the Voronoi diagram and its centroid decomposition for every u,
- $O(n^{1.5})$ in total.

Point location

- By storing preorder numbers for the shortest paths tree rooted at s_{i_j} we can check if v is on the left/right of p_{i_j} in constant time.
- This allows us to detect in constant time the relevant smaller component of the tree representing the Voronoi diagram that needs to be recursively searched for the cell of v (if it is not the cell of any s_{i_j}).

Time: $O(\log n)$

Space:
- $|Sep| \cdot O(n)$ to store the shortest path tree for every site,
- $n \cdot O(\sqrt{n})$ to store the Voronoi diagram and its centroid decomposition for every u,
- $O(n^{1.5})$ in total.

Point location

- By storing preorder numbers for the shortest paths tree rooted at s_{i_j} we can check if v is on the left/right of p_{i_j} in constant time.
- This allows us to detect in constant time the relevant smaller component of the tree representing the Voronoi diagram that needs to be recursively searched for the cell of v (if it is not the cell of any s_{i_j}).

Time: $O(\log n)$

Space:
- $|Sep| \cdot O(n)$ to store the shortest path tree for every site,
- $n \cdot O(\sqrt{n})$ to store the Voronoi diagram and its centroid decomposition for every u,
- $O(n^{1.5})$ in total.

Point location

- By storing preorder numbers for the shortest paths tree rooted at s_{i_j} we can check if v is on the left/right of p_{i_j} in constant time.
- This allows us to detect in constant time the relevant smaller component of the tree representing the Voronoi diagram that needs to be recursively searched for the cell of v (if it is not the cell of any s_{i_j}).

Time: $O(\log n)$

Space:
- $|Sep| \cdot O(n)$ to store the shortest path tree for every site,
- $n \cdot O(\sqrt{n})$ to store the Voronoi diagram and its centroid decomposition for every u,
- $O(n^{1.5})$ in total.

Point location

- By storing preorder numbers for the shortest paths tree rooted at s_{i_j} we can check if v is on the left/right of p_{i_j} in constant time.
- This allows us to detect in constant time the relevant smaller component of the tree representing the Voronoi diagram that needs to be recursively searched for the cell of v (if it is not the cell of any s_{i_j}).

Time: $O(\log n)$

Space:
- $|Sep| \cdot O(n)$ to store the shortest path tree for every site,
- $n \cdot O(\sqrt{n})$ to store the Voronoi diagram and its centroid decomposition for every u,
- $O(n^{1.5})$ in total.

Point location

- By storing preorder numbers for the shortest paths tree rooted at s_{i_j} we can check if v is on the left/right of p_{i_j} in constant time.
- This allows us to detect in constant time the relevant smaller component of the tree representing the Voronoi diagram that needs to be recursively searched for the cell of v (if it is not the cell of any s_{i_j}).

Time: $O(\log n)$

Space:
- $|Sep| \cdot O(n)$ to store the shortest path tree for every site,
- $n \cdot O(\sqrt{n})$ to store the Voronoi diagram and its centroid decomposition for every u,
- $O(n^{1.5})$ in total.

Basic recursion, again

- Recursively build an exact distance oracle for the inside.
- Recursively build an exact distance oracle for the outside.
- Build a structure that that can be used to find the shortest path from u to v that visits at least one node of the separator.

The overall space is roughly $S(n) = O(n^{1.5}) + 2S(n/2)$, so $O(n^{1.5})$ overall. But the query would take $O(\log^2 n)$...

Can we decrease the number of subproblems where we use the point location structure?

It is enough to query only the structure corresponding to the topmost subproblem where u and v lie on different sides of the separator if we choose the separators more carefully. The query time becomes $O(\log n)$.

Basic recursion, again

- Recursively build an exact distance oracle for the inside.
- Recursively build an exact distance oracle for the outside.
- Build a structure that that can be used to find the shortest path from u to v that visits at least one node of the separator.

The overall space is roughly $S(n) = O(n^{1.5}) + 2S(n/2)$, so $O(n^{1.5})$ overall. But the query would take $O(\log^2 n)$...

Can we decrease the number of subproblems where we use the point location structure?

It is enough to query only the structure corresponding to the topmost subproblem where u and v lie on different sides of the separator if we choose the separators more carefully. The query time becomes $O(\log n)$.

Basic recursion, again

- Recursively build an exact distance oracle for the inside.
- Recursively build an exact distance oracle for the outside.
- Build a structure that that can be used to find the shortest path from u to v that visits at least one node of the separator.

The overall space is roughly $S(n) = O(n^{1.5}) + 2S(n/2)$, so $O(n^{1.5})$ overall. But the query would take $O(\log^2 n)$...

Can we decrease the number of subproblems where we use the point location structure?

It is enough to query only the structure corresponding to the topmost subproblem where u and v lie on different sides of the separator if we choose the separators more carefully. The query time becomes $O(\log n)$.

Basic recursion, again

- Recursively build an exact distance oracle for the inside.
- Recursively build an exact distance oracle for the outside.
- Build a structure that that can be used to find the shortest path from u to v that visits at least one node of the separator.

The overall space is roughly $S(n) = O(n^{1.5}) + 2S(n/2)$, so $O(n^{1.5})$ overall. But the query would take $O(\log^2 n)$...

Can we decrease the number of subproblems where we use the point location structure?

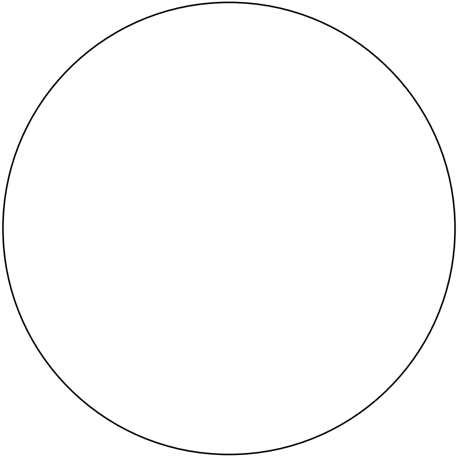
It is enough to query only the structure corresponding to the topmost subproblem where u and v lie on different sides of the separator if we choose the separators more carefully. The query time becomes $O(\log n)$.

Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.

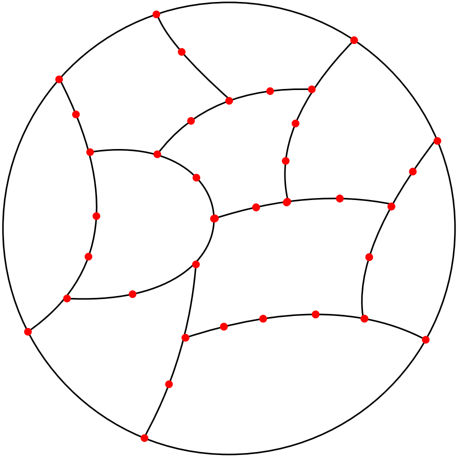
Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.



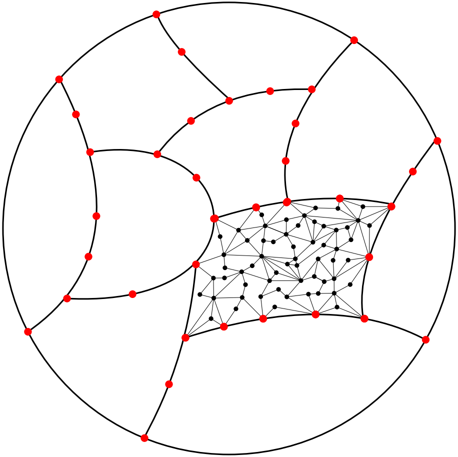
Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.



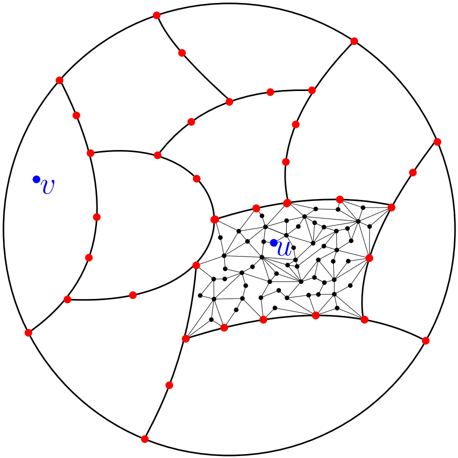
Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.



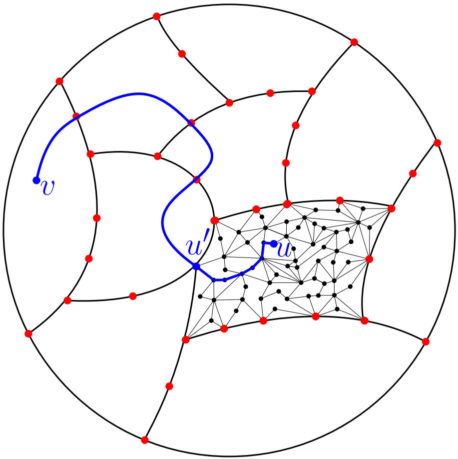
Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.



Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.



Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.
- Replace the explicitly stored shortest paths trees with the MSSP structure of Klein, except that we need to slightly extend the interface of the link-cut trees used to maintain the current tree.

After some calculations...

Time: $O(\sqrt{r}\log^2 n)$

Space: $O(n^{1.5}/\sqrt{r} + n\log n \log(n/r))$

Time can be decreased to $O(\sqrt{r}\log n \log r)$ with an additional trick.

Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.
- Replace the explicitly stored shortest paths trees with the MSSP structure of Klein, except that we need to slightly extend the interface of the link-cut trees used to maintain the current tree.

After some calculations...

Time: $O(\sqrt{r}\log^2 n)$

Space: $O(n^{1.5}/\sqrt{r} + n\log n\log(n/r))$

Time can be decreased to $O(\sqrt{r}\log n\log r)$ with an additional trick.

Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.
- Replace the explicitly stored shortest paths trees with the MSSP structure of Klein, except that we need to slightly extend the interface of the link-cut trees used to maintain the current tree.

After some calculations...

Time: $O(\sqrt{r}\log^2 n)$

Space: $O(n^{1.5}/\sqrt{r} + n\log n\log(n/r))$

Time can be decreased to $O(\sqrt{r}\log n\log r)$ with an additional trick.

Tradeoff

- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.
- Replace the explicitly stored shortest paths trees with the MSSP structure of Klein, except that we need to slightly extend the interface of the link-cut trees used to maintain the current tree.

After some calculations...

Time: $O(\sqrt{r}\log^2 n)$

Space: $O(n^{1.5}/\sqrt{r} + n\log n\log(n/r))$

Time can be decreased to $O(\sqrt{r}\log n\log r)$ with an additional trick.

Tradeoff

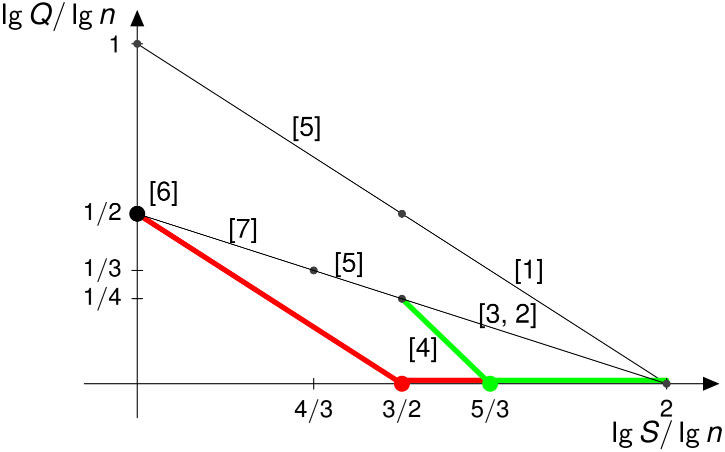
- Use r-divisions to decrease the number of stored Voronoi diagrams by a factor of $\sqrt{r}$. Then, we need to guess the boundary node u' in the region of u, there are $\sqrt{r}$ possibilities.
- Replace the explicitly stored shortest paths trees with the MSSP structure of Klein, except that we need to slightly extend the interface of the link-cut trees used to maintain the current tree.

After some calculations...

Time: $O(\sqrt{r}\log^2 n)$

Space: $O(n^{1.5}/\sqrt{r} + n\log n\log(n/r))$

Time can be decreased to $O(\sqrt{r}\log n\log r)$ with an additional trick.



Questions?