Almost Optimal Distance Oracles for Planar Graphs

Panagiotis Charalampopoulos[1,3] Paweł Gawrychowski[2]
Shay Mozes[3] Oren Weimann[4]

[1]King's College London, UK

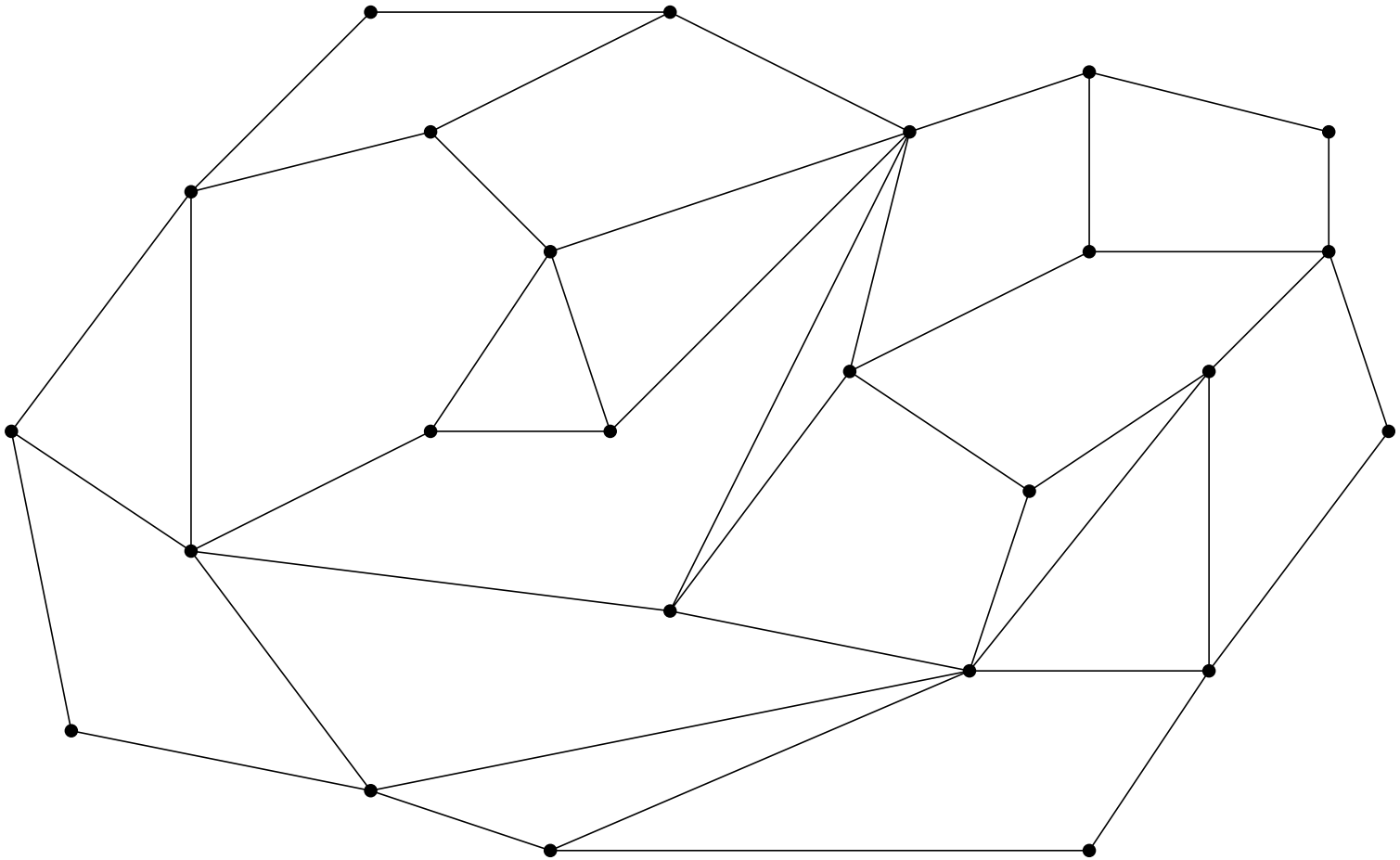
[2]University of Wrocław, Poland

[3]Interdisciplinary Center Herzliya, Israel

[4]University of Haifa, Israel

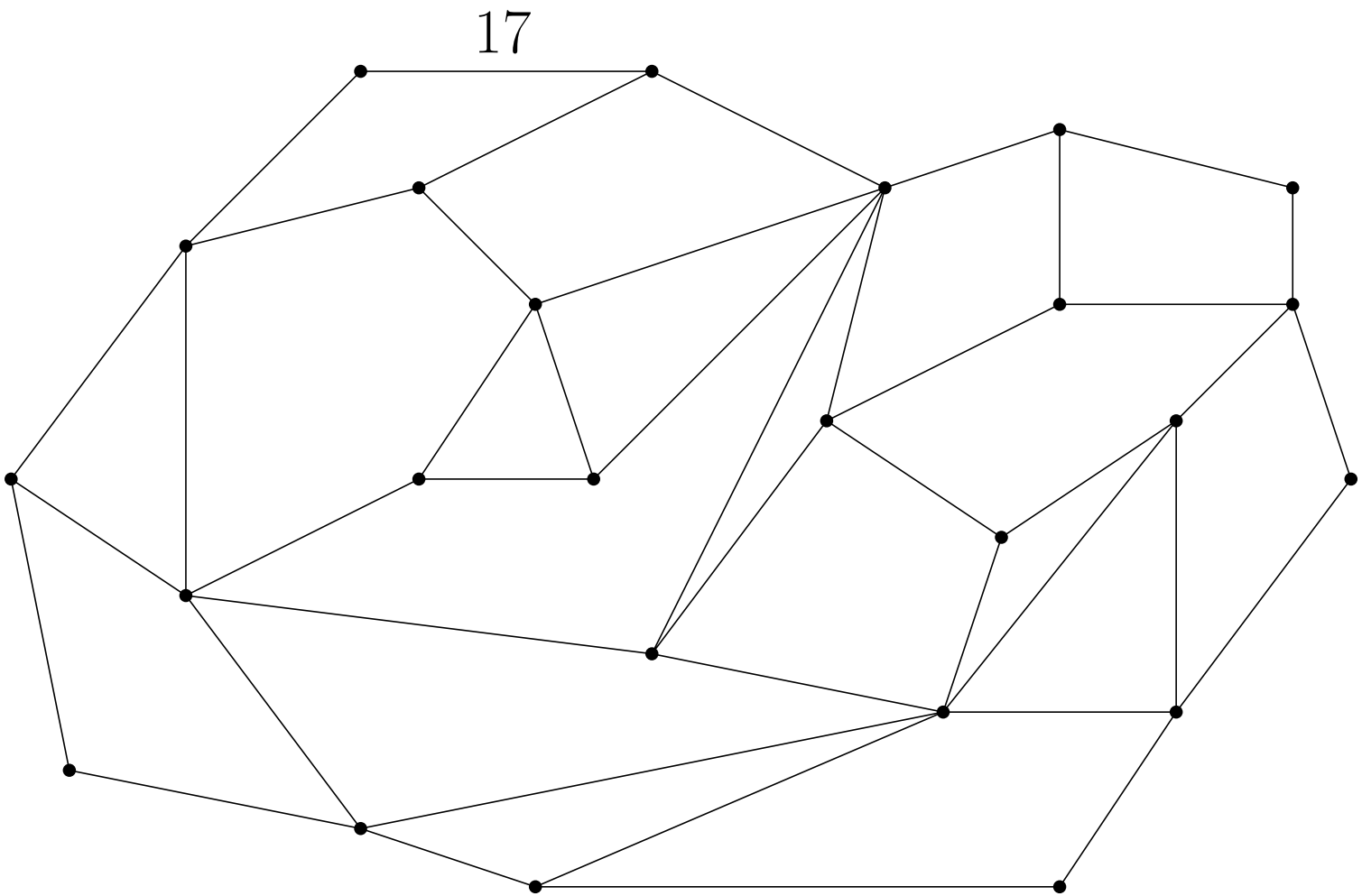
May 15, 2019

Problem definition



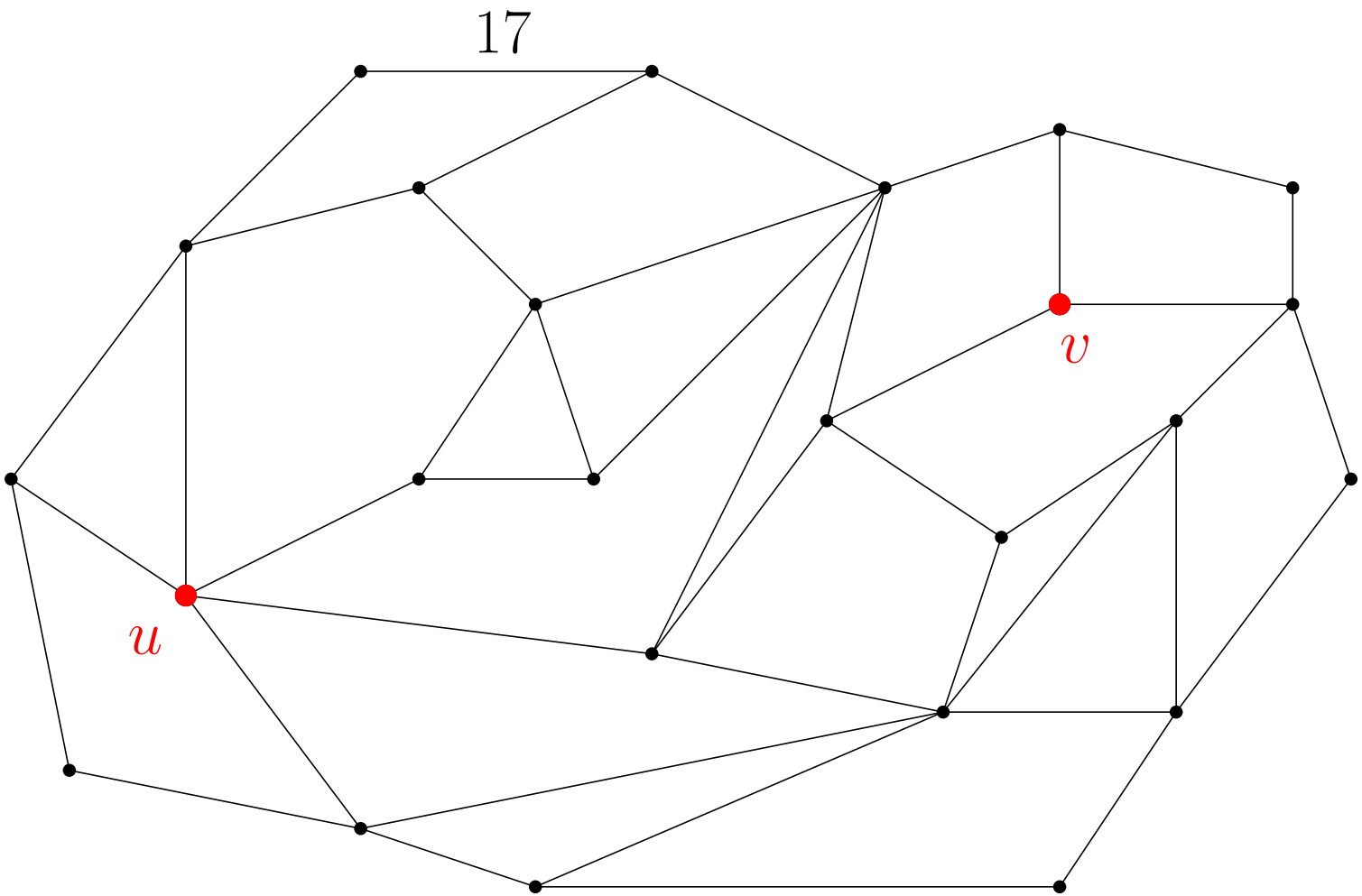
Preprocess an n-vertex planar graph $G = (V, E)$ with nonnegative arc lengths, so that given any $u, v \in V$ we can compute $d(u, v)$ efficiently.

Problem definition



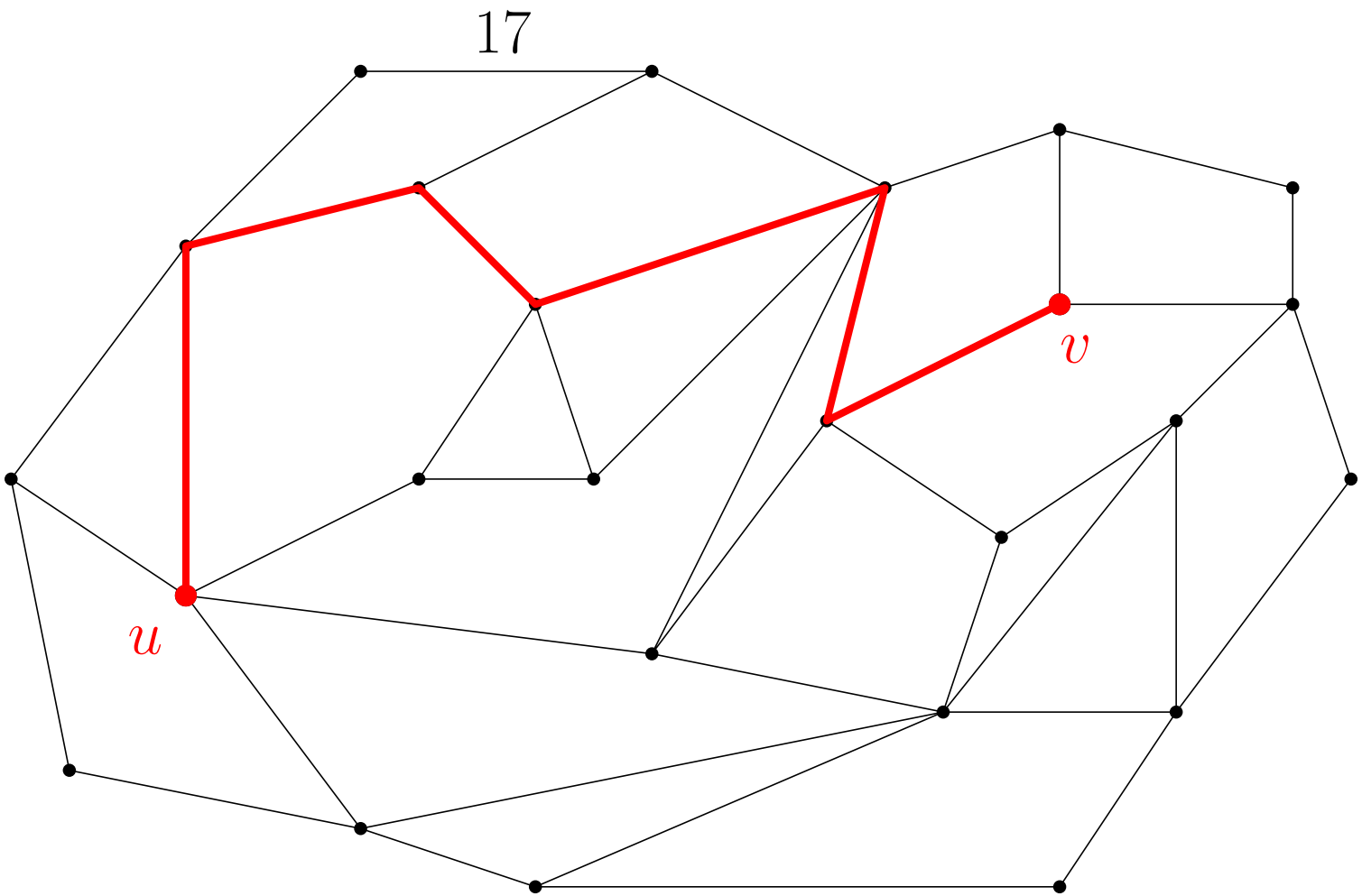
Preprocess an n-vertex planar graph $G = (V, E)$ with nonnegative arc lengths, so that given any $u, v \in V$ we can compute $d(u, v)$ efficiently.

Problem definition



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.

Problem definition



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.

Goals

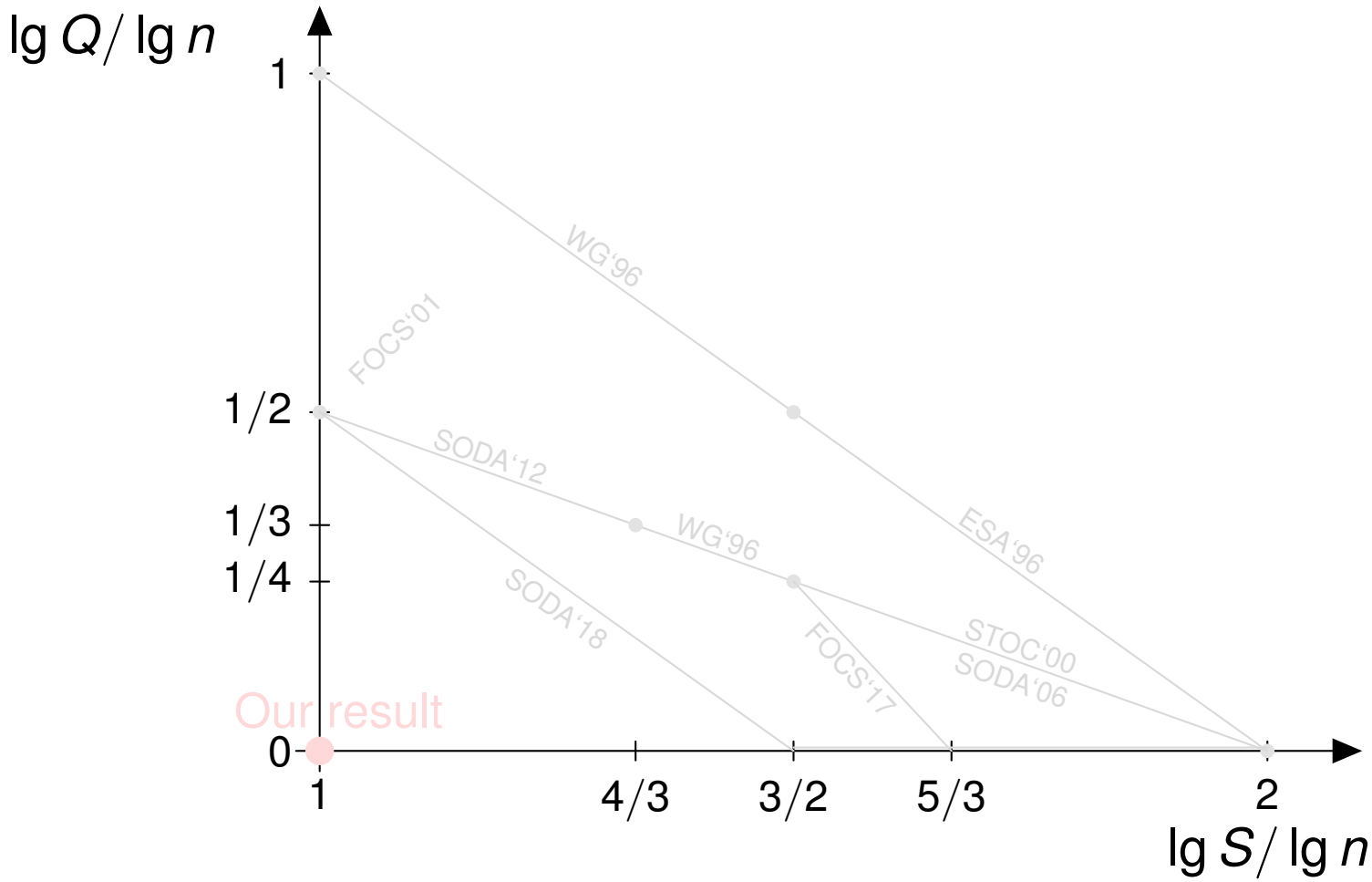
Ideally:

- Fast queries, ideally $Q = O(1)$.
- Small size, ideally $S = O(n)$.
- Fast construction, ideally $T = O(n)$.

The most important tradeoff is between query-time Q and size S.

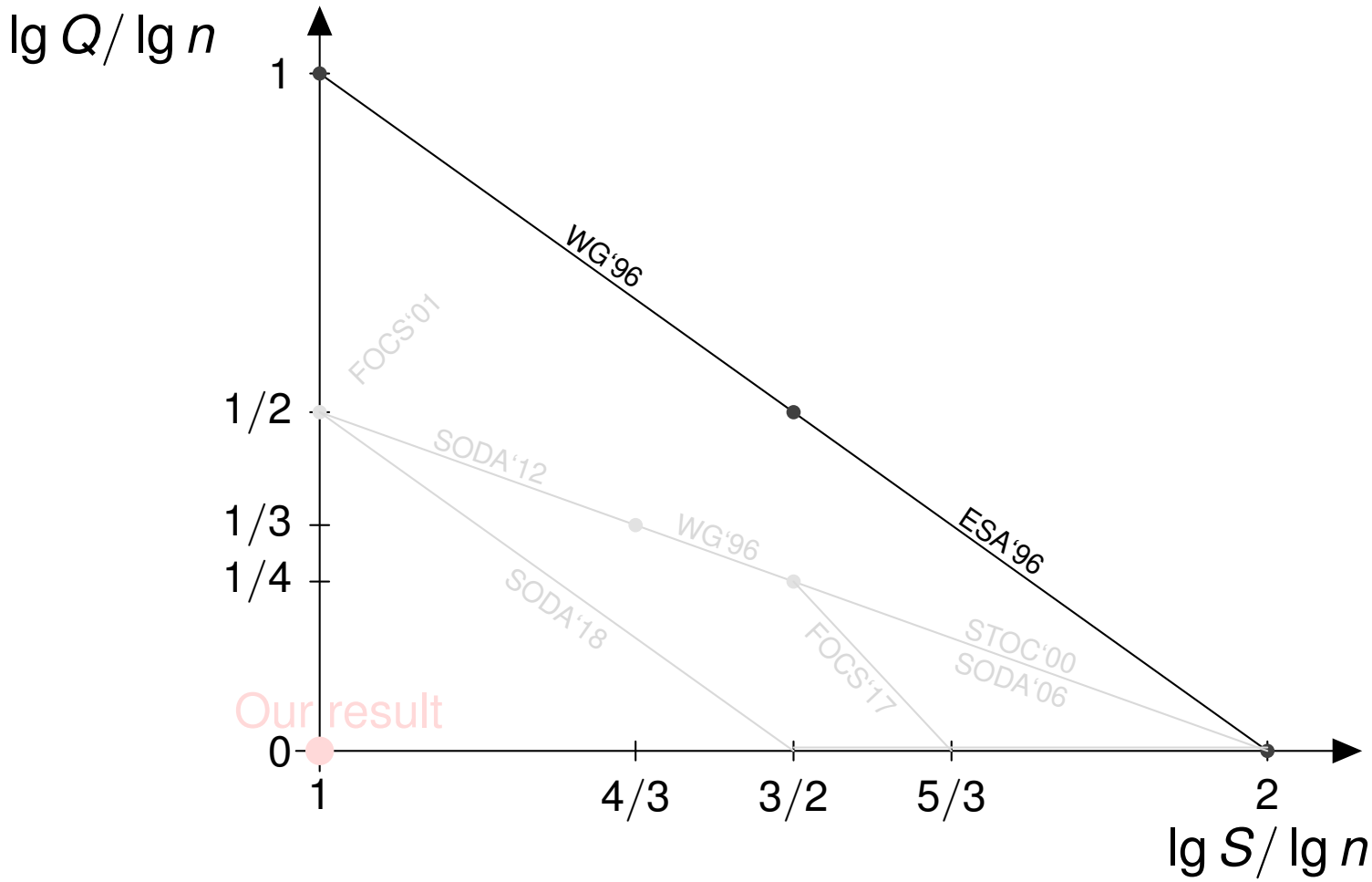
Previous work

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



Previous work

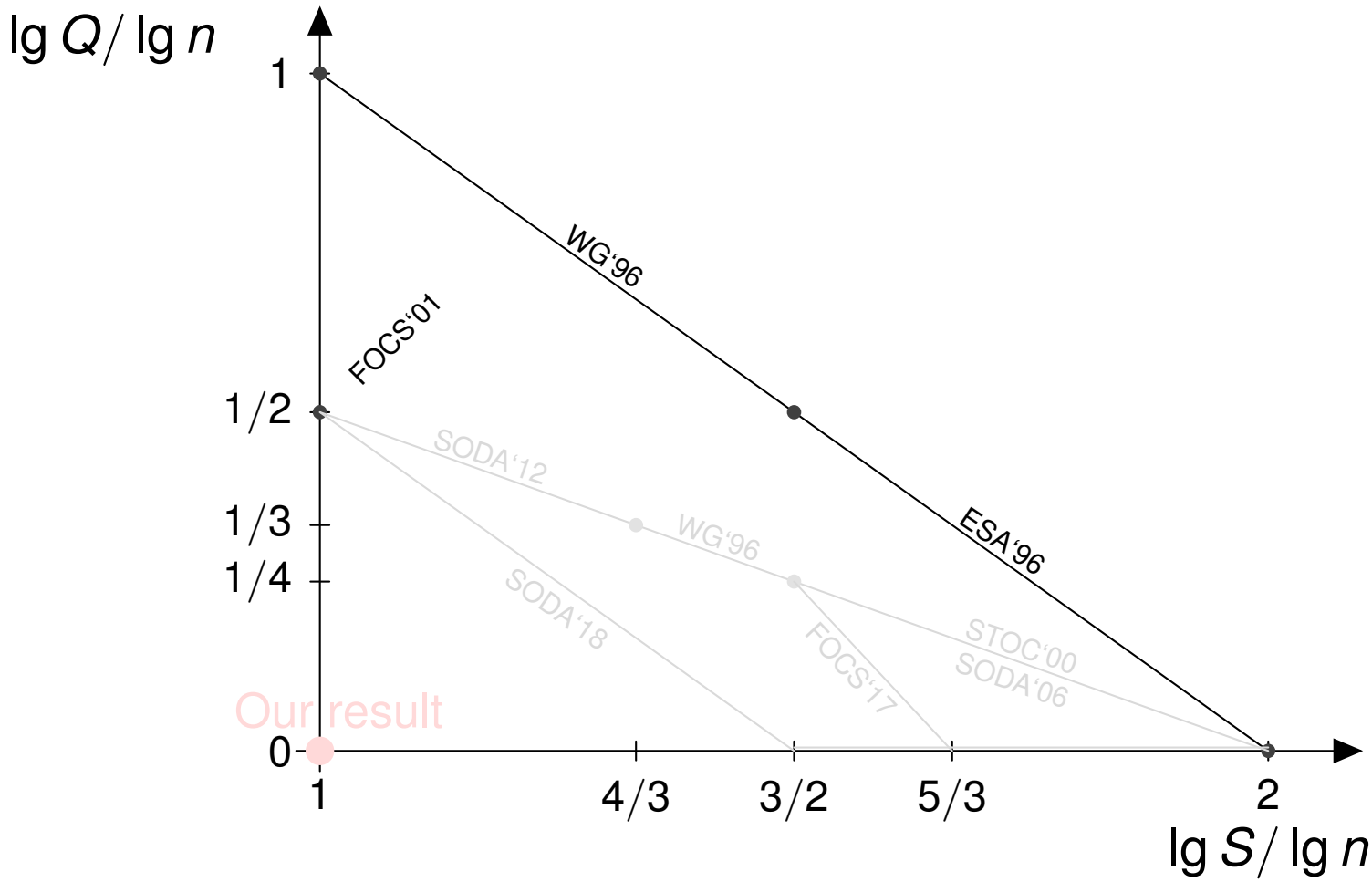
The tradeoff between the query-time Q and the size S of the structure:



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

Previous work

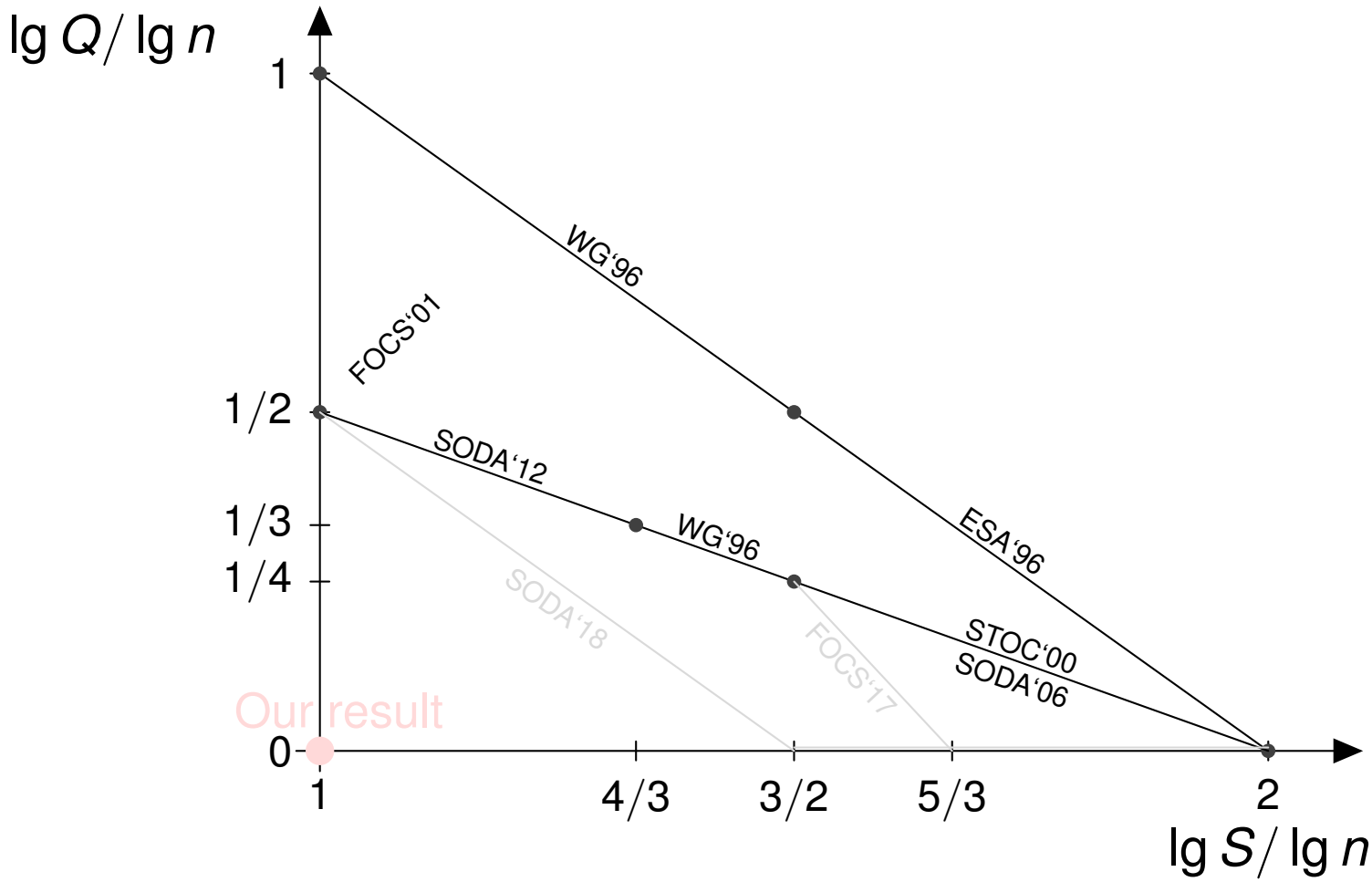
The tradeoff between the query-time Q and the size S of the structure:



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

Previous work

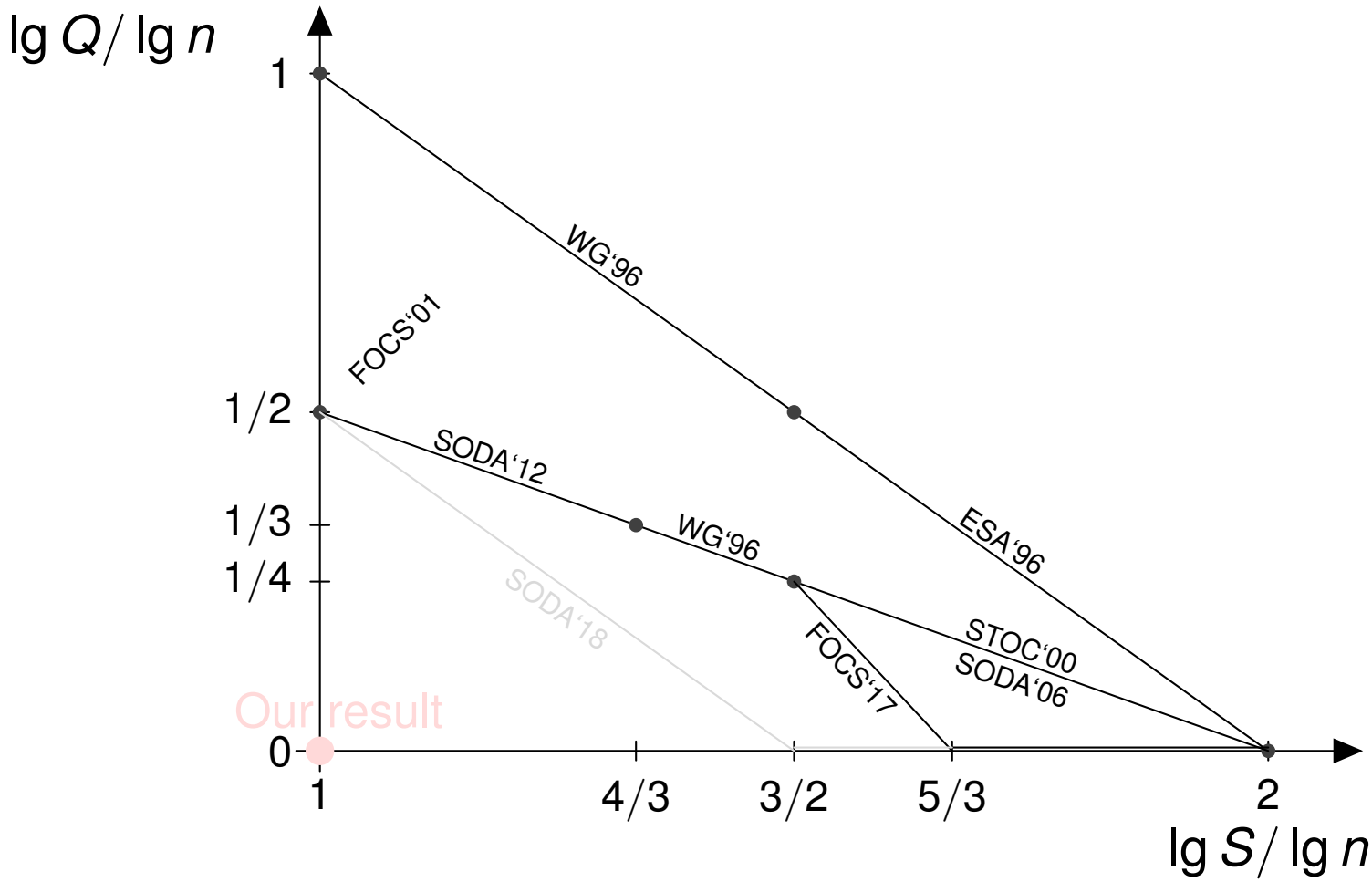
The tradeoff between the query-time Q and the size S of the structure:



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

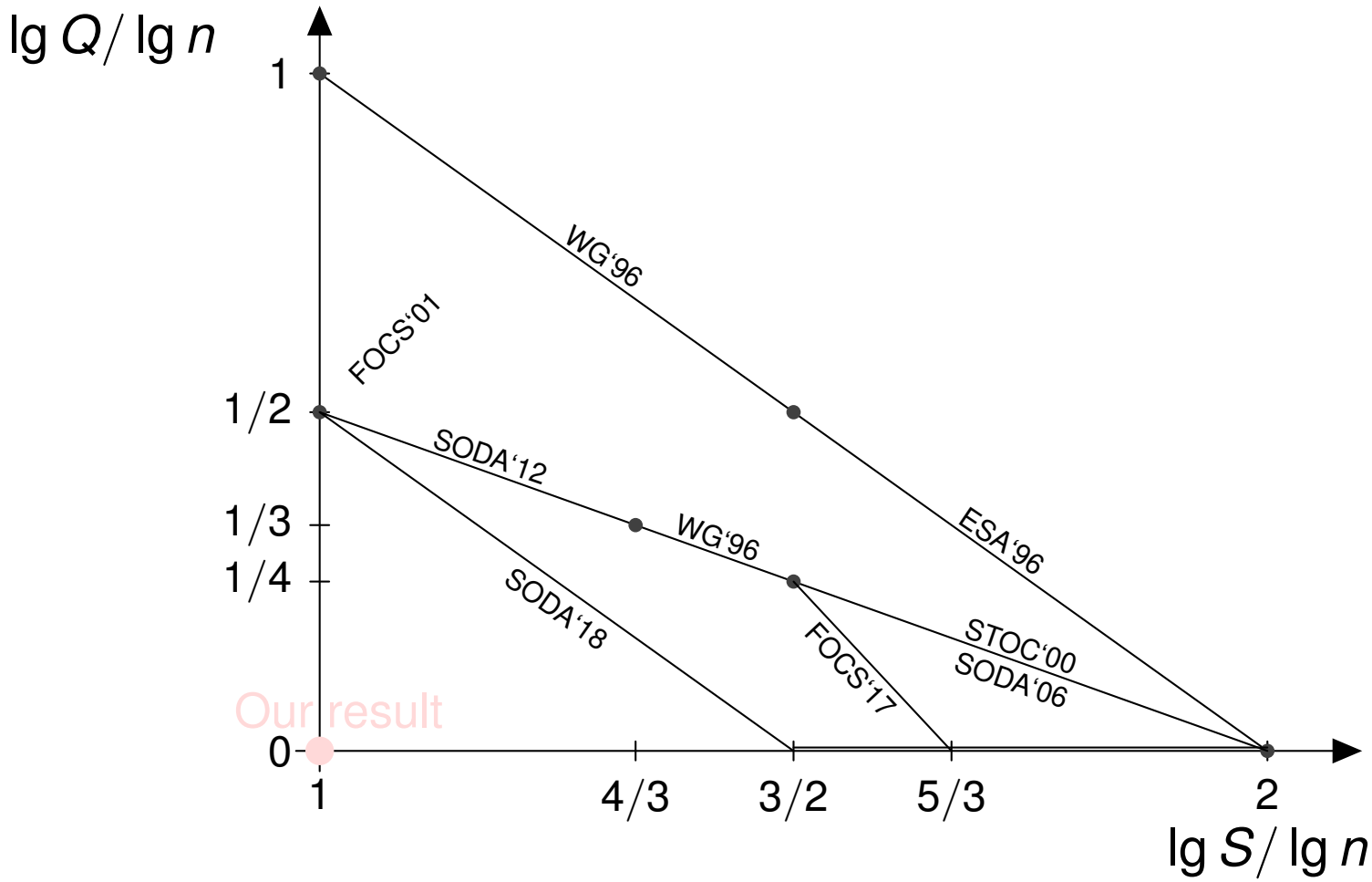
The tradeoff between the query-time Q and the size S of the structure:



In 2017, Cohen-Addad, Dahlgaard, and Wulff-Nilsen showed that this is not optimal, and $S = O(n^{5/3})$ with $Q = O(\log n)$ is possible.

Previous work

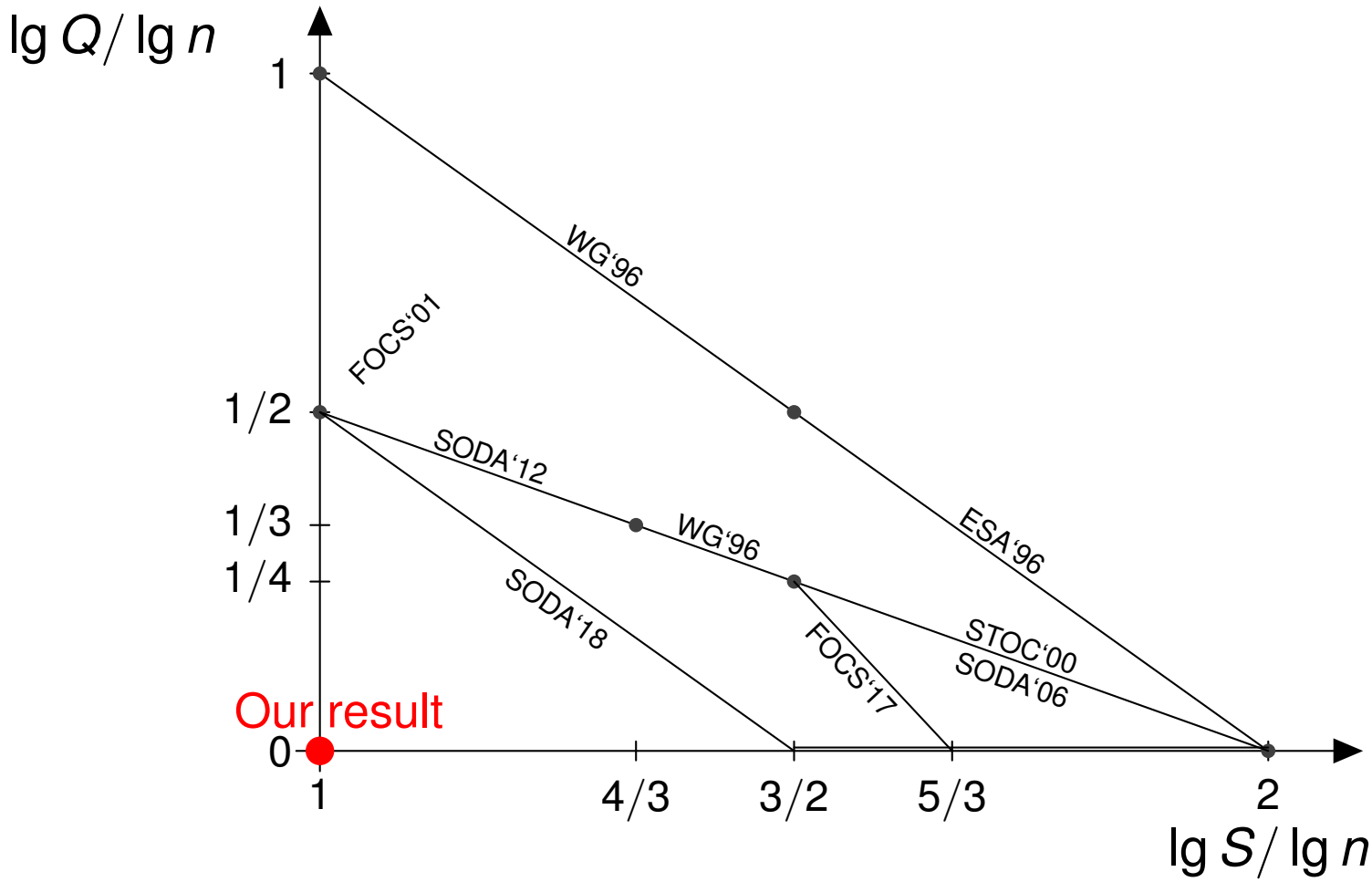
The tradeoff between the query-time Q and the size S of the structure:



In 2018, Gawrychowski et al. improved this to $S = O(n^{1.5})$ and $Q = O(\log n)$.

Previous work

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



We improve this to $S = O(n^{1+\epsilon})$ and $Q = \tilde{O}(1)$ for any $\epsilon > 0$.

Main result

We show the following tradeoffs for $\langle S, Q \rangle$:

1. $\langle \tilde{O}(n^{1+\epsilon}), O(\log^{1/\epsilon} n) \rangle$, for any constant $1/2 \geq \epsilon > 0$;
2. $\langle O(n \log^{2+1/\epsilon} n), \tilde{O}(n^{\epsilon}) \rangle$, for any constant $\epsilon > 0$;
3. $\langle n^{1+o(1)}, n^{o(1)} \rangle$.

The oracle is based on a recursive view of the point location mechanism for Voronoi diagrams of Gawrychowski et al. [SODA'18].

Main result

We show the following tradeoffs for $\langle S, Q \rangle$:

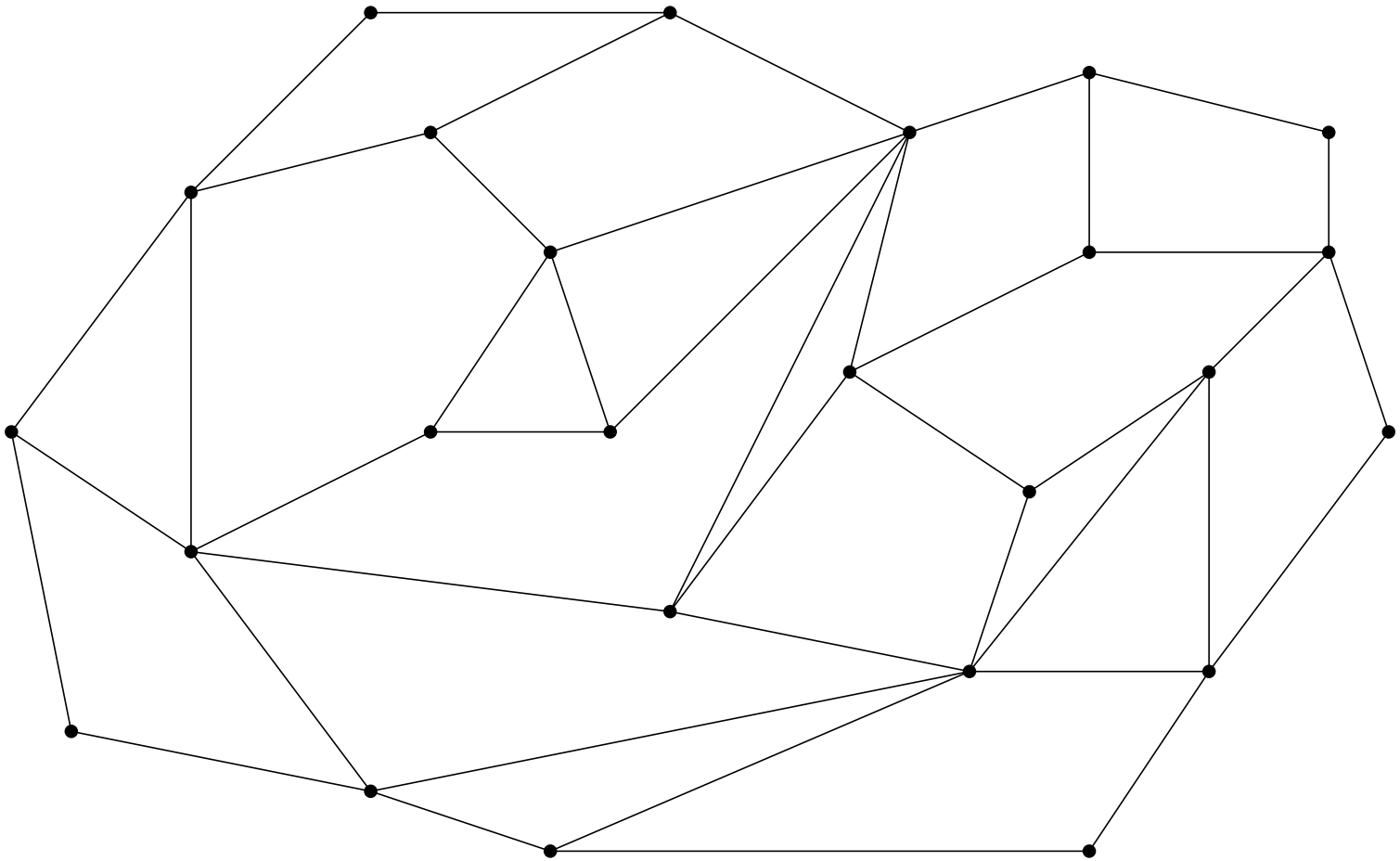
1. $\langle \tilde{O}(n^{1+\epsilon}), O(\log^{1/\epsilon} n) \rangle$, for any constant $1/2 \geq \epsilon > 0$;
2. $\langle O(n \log^{2+1/\epsilon} n), \tilde{O}(n^{\epsilon}) \rangle$, for any constant $\epsilon > 0$;
3. $\langle n^{1+o(1)}, n^{o(1)} \rangle$.

The oracle is based on a recursive view of the point location mechanism for Voronoi diagrams of Gawrychowski et al. [SODA'18].

Cycle Separator

Miller [JCSS'86]

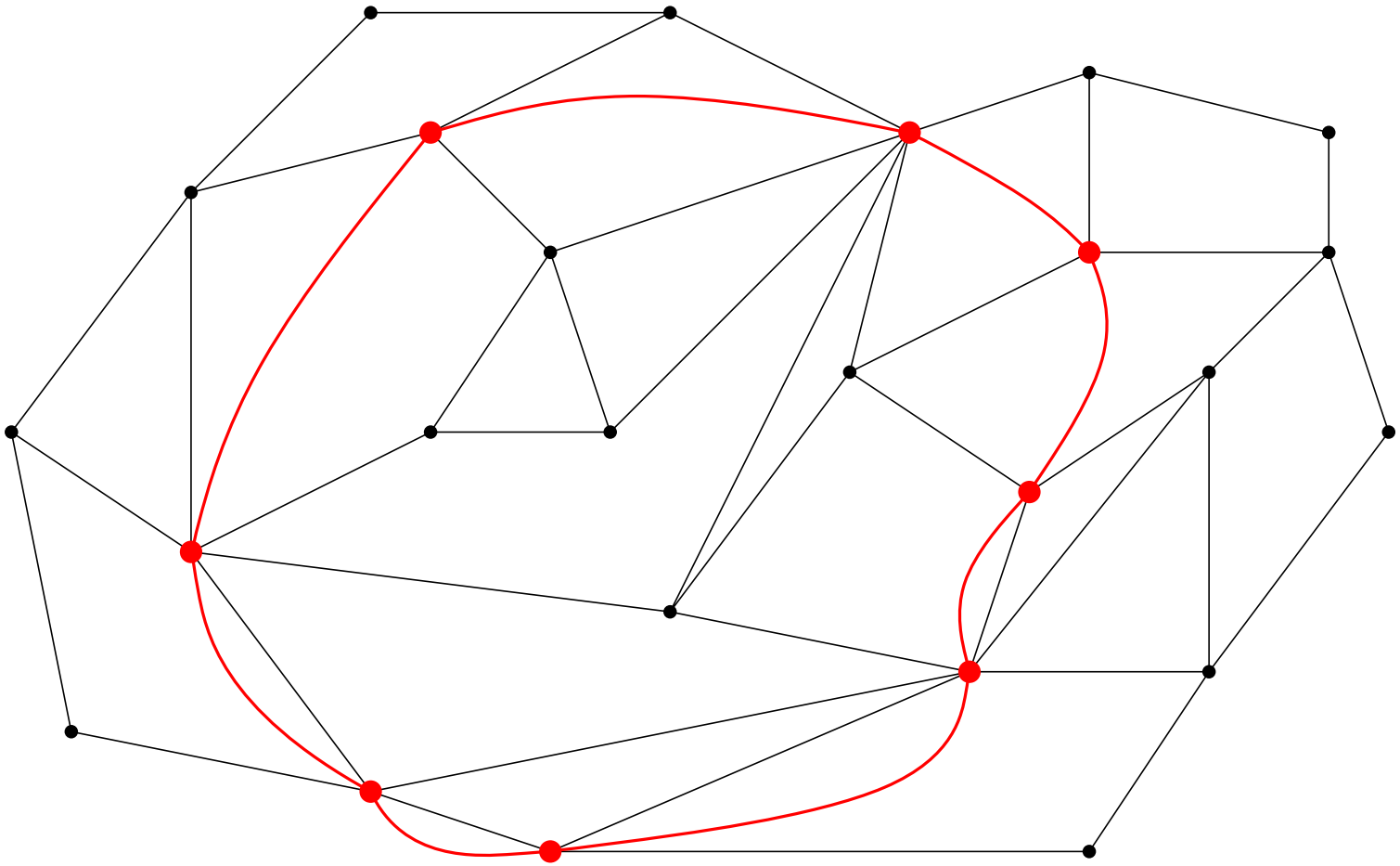
There always exists a Jordan curve separator of size $O(\sqrt{n})$ such that there are at most $\frac{2}{3}n$ nodes on its inside/outside.



Cycle Separator

Miller [JCSS'86]

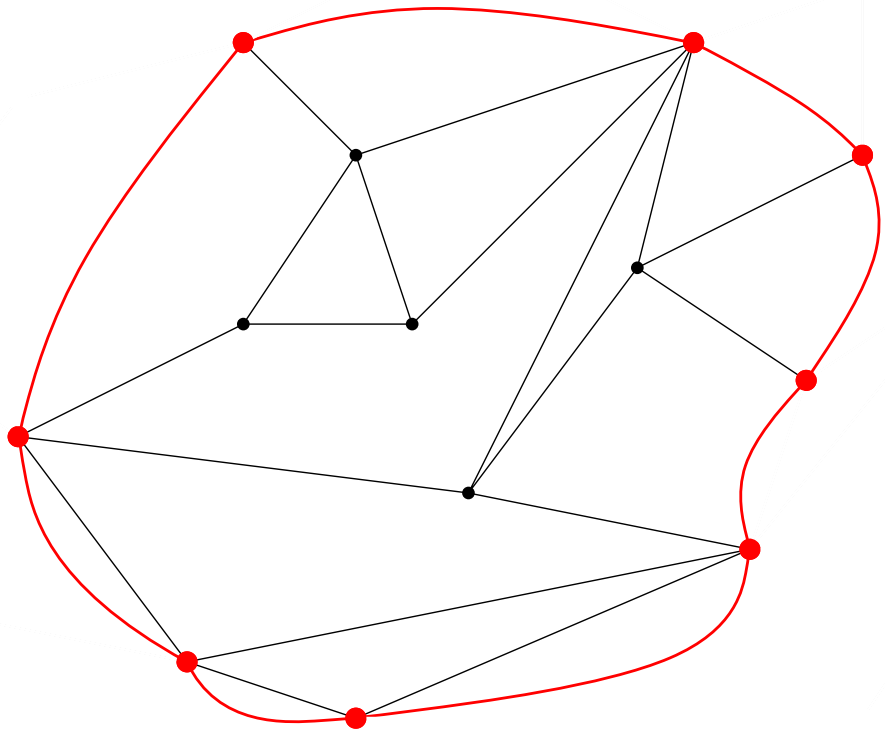
There always exists a Jordan curve separator of size $O(\sqrt{n})$ such that there are at most $\frac{2}{3}n$ nodes on its inside/outside.



Multiple Source Shortest Paths (MSSP)

Klein [SODA'05]

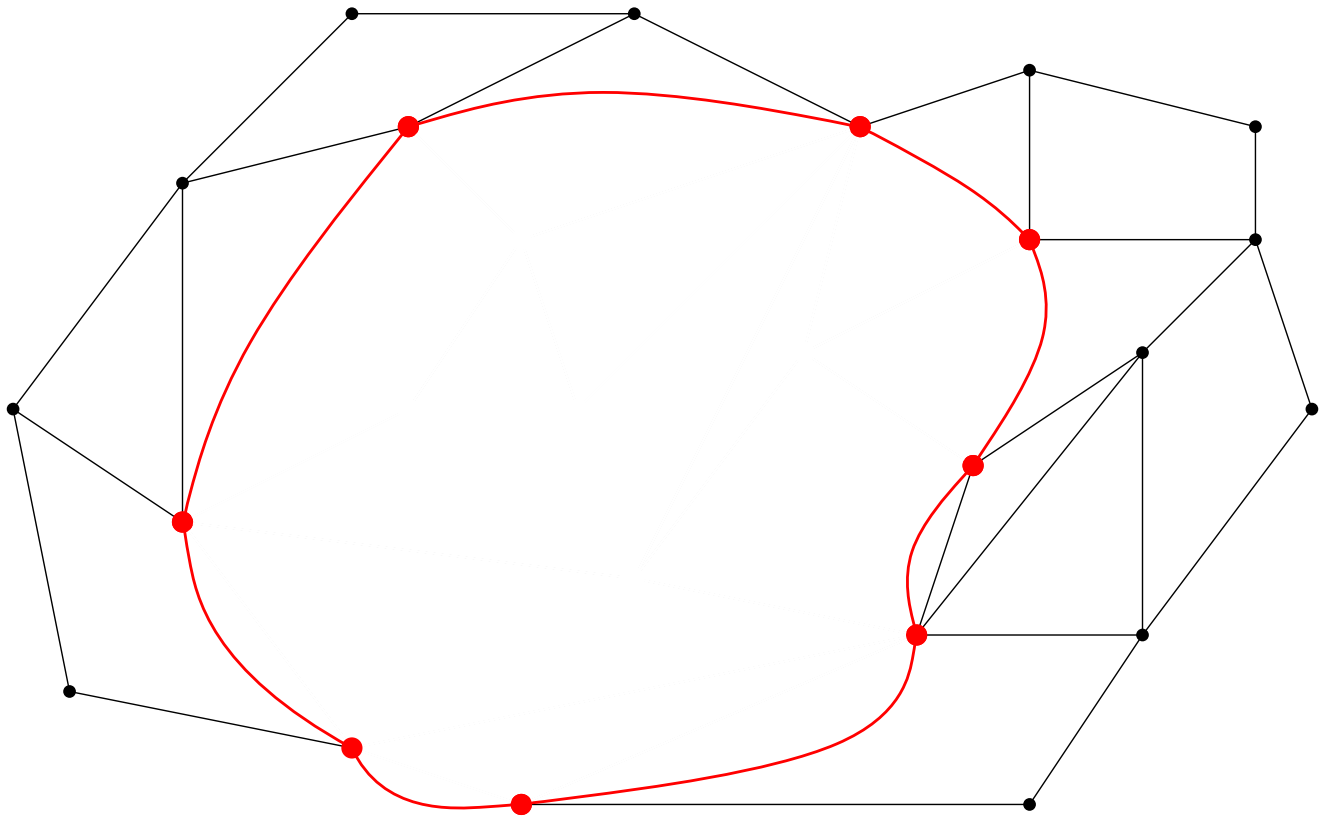
There exists a data structure requiring $O(n \log n)$ space that can report in $O(\log n)$ time the distance between any node on the infinite face (boundary node) and any node in the graph.



Multiple Source Shortest Paths (MSSP)

Klein [SODA'05]

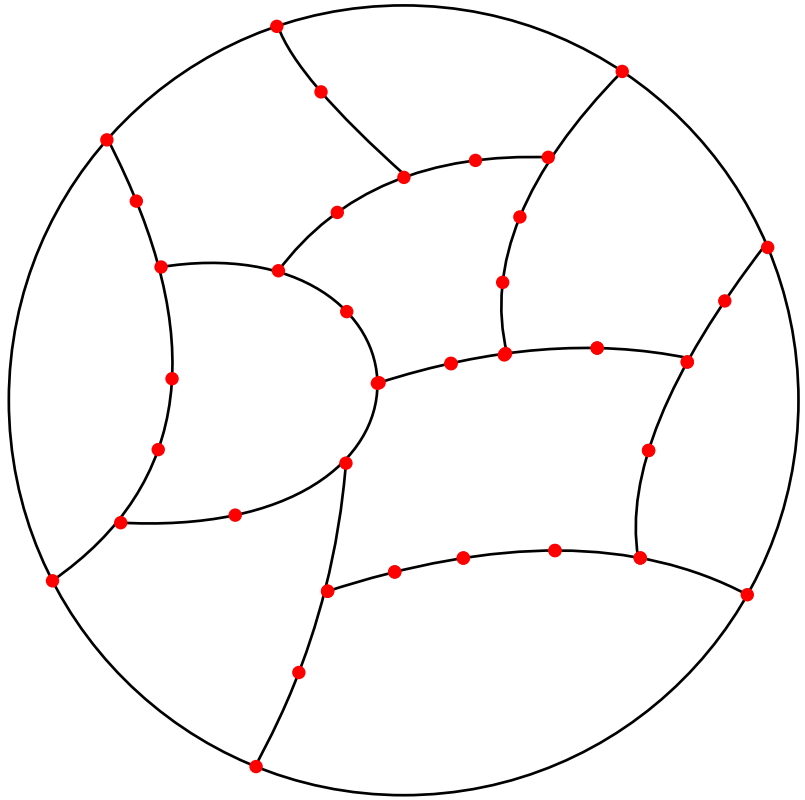
There exists a data structure requiring $O(n \log n)$ space that can report in $O(\log n)$ time the distance between any node on the infinite face (boundary node) and any node in the graph.



r-divisions

For $r \in [1, n]$, a decomposition of the graph into:

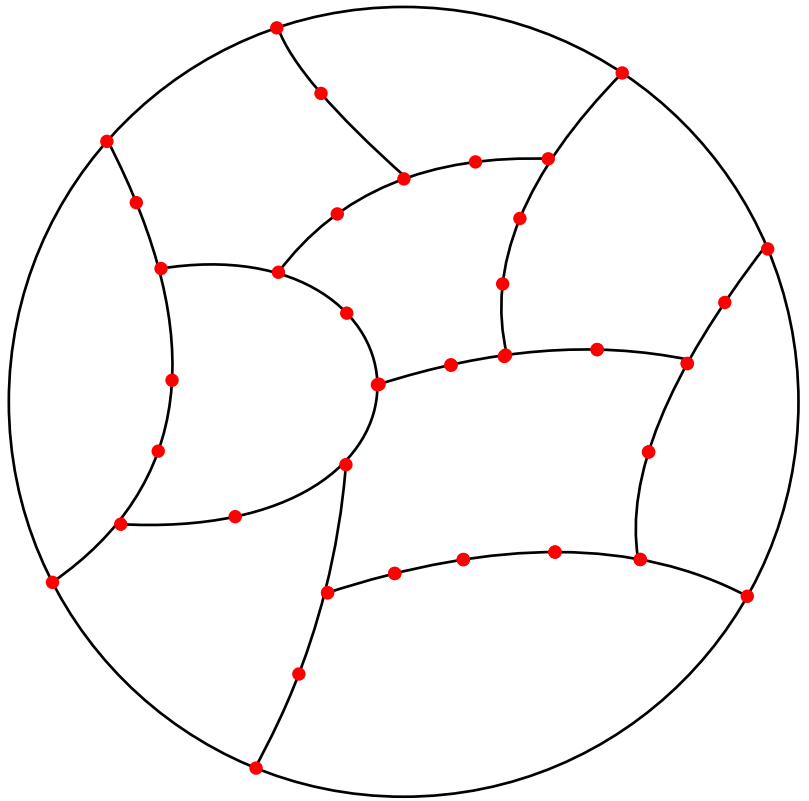
- $O(n/r)$ pieces
- each piece has $O(r)$ vertices.
- each piece has $O(\sqrt{r})$ boundary vertices, (vertices incident to edges in other pieces)



We denote the boundary of a piece P by ∂P and assume that all such nodes lie on a single face of P.

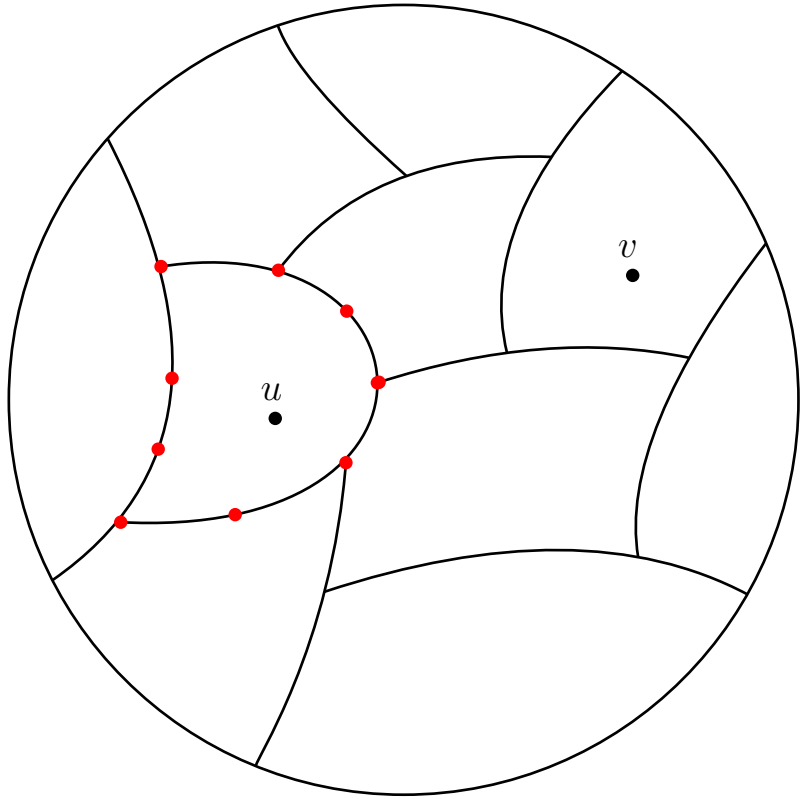
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

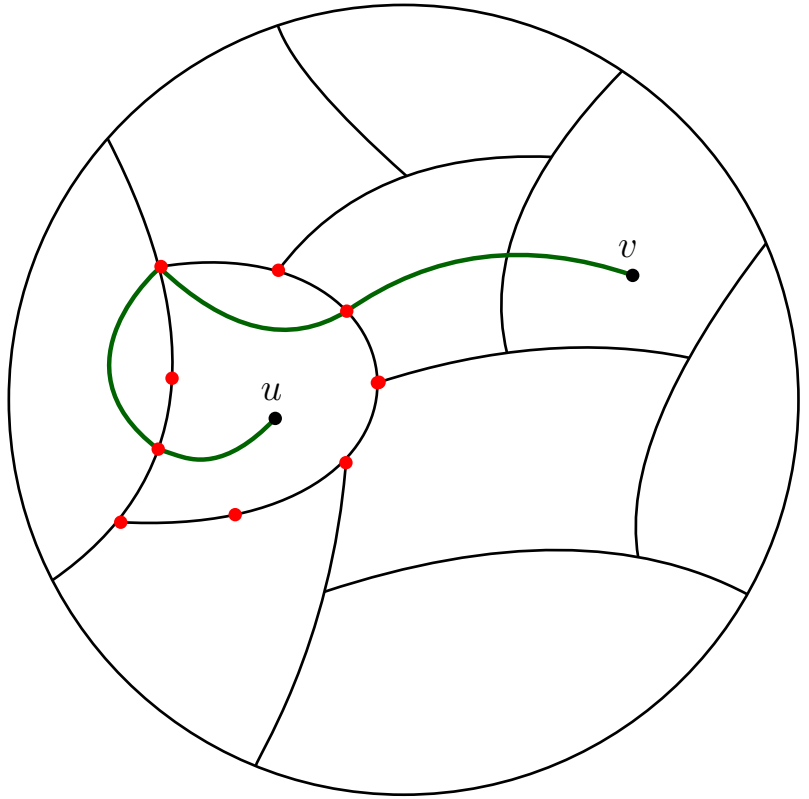
- Compute an r-division.
- For each piece P, for each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$.
  Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P.
  Space $\tilde{O}(n/r \cdot n)$.



Interesting case.

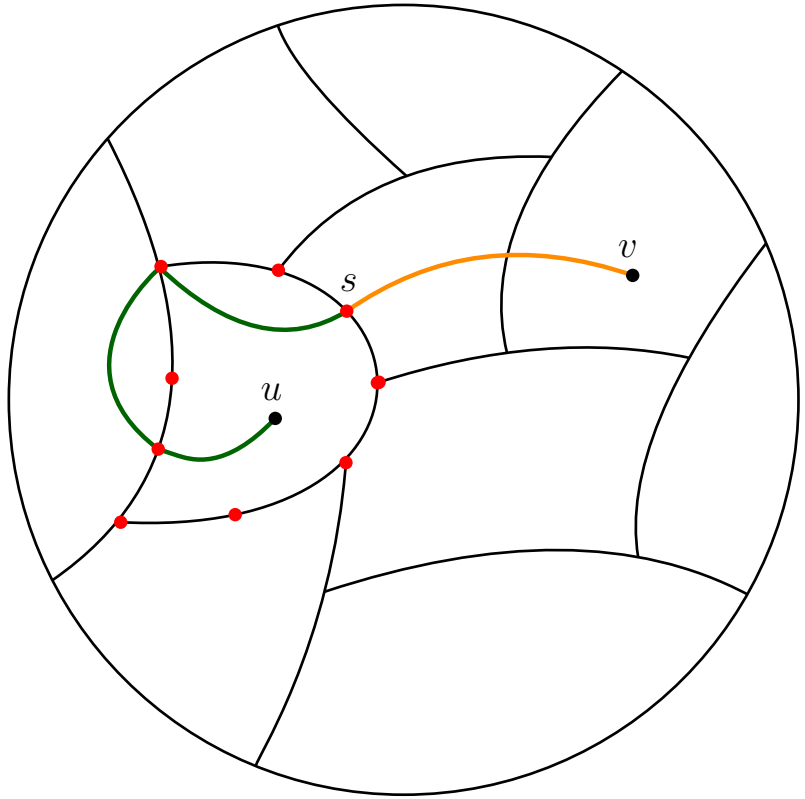
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

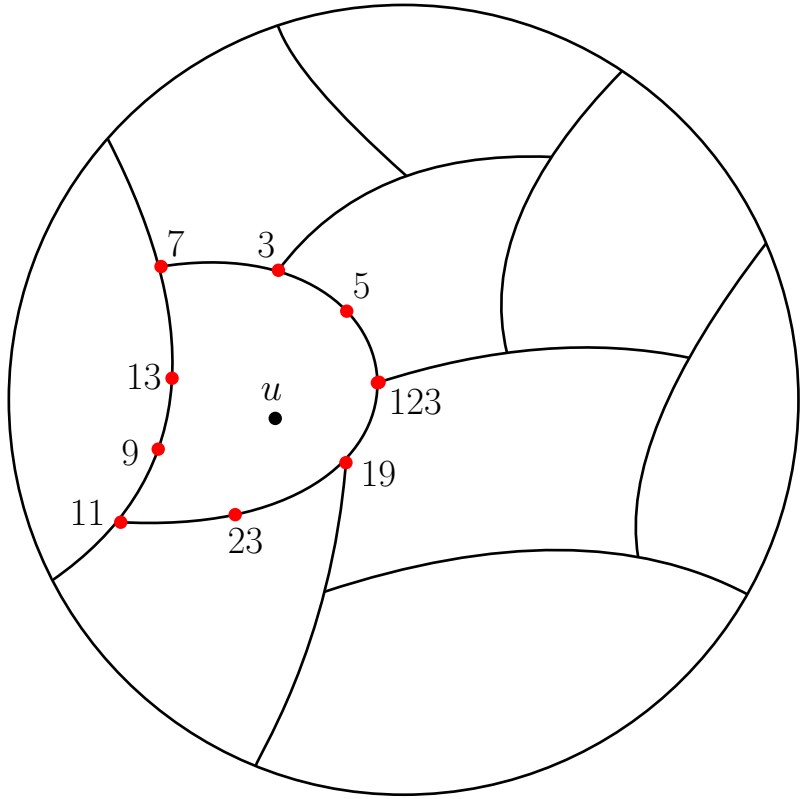
- Compute an r-division.
- For each piece P, for each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



We decompose the path on the last boundary node it visits.

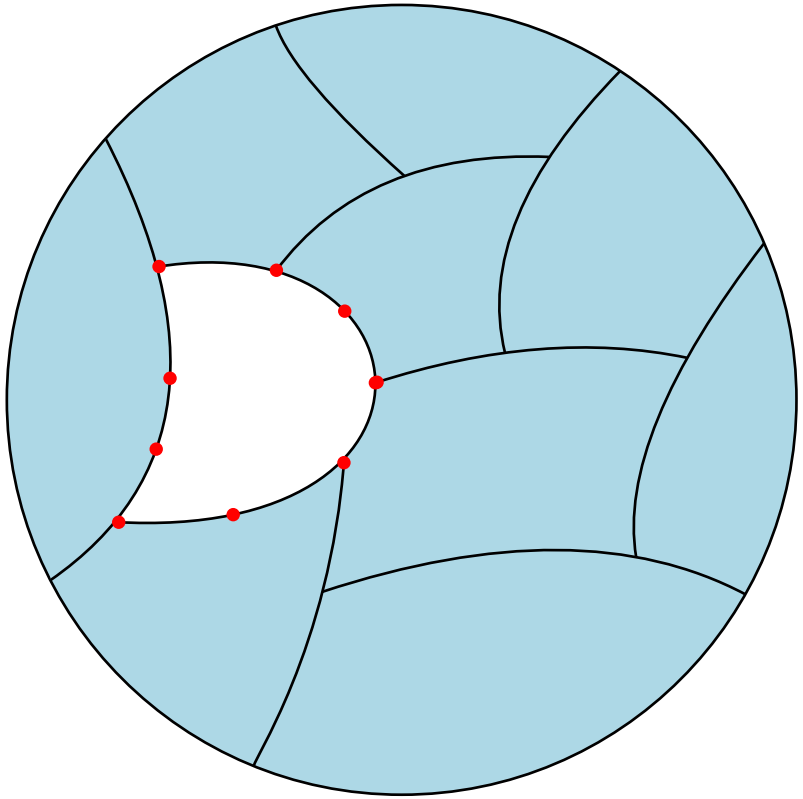
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



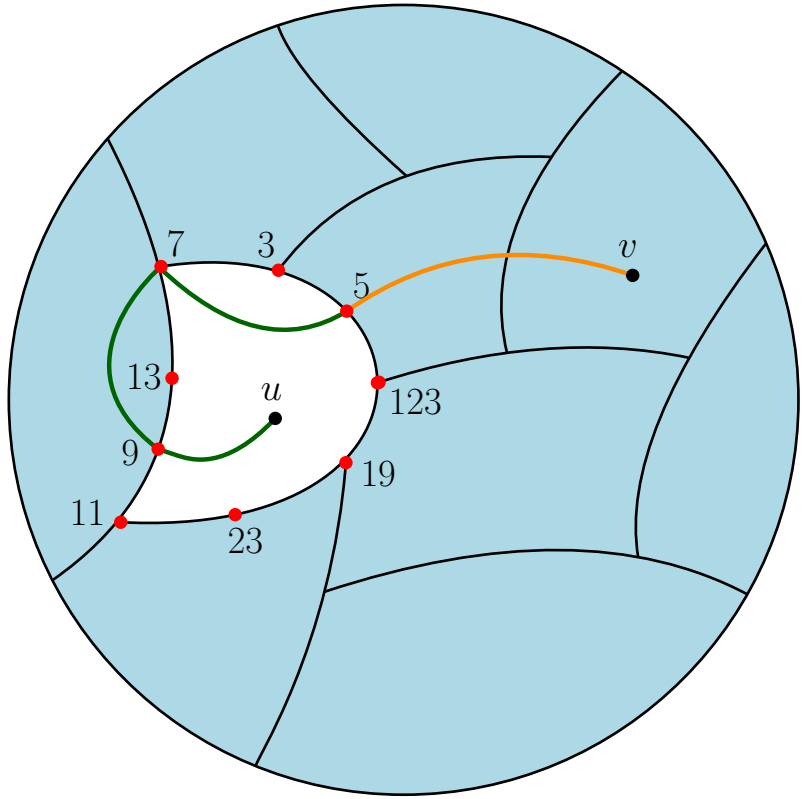
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

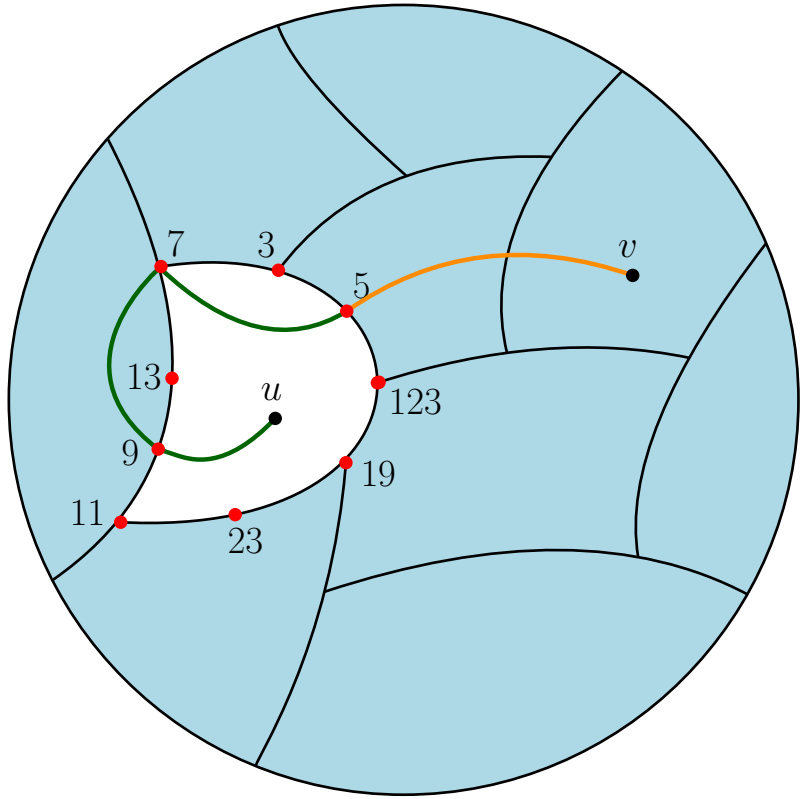
- Compute an r-division.
- For each piece P, for each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



At query, try all possible $O(\sqrt{r})$ candidate boundary nodes.

Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

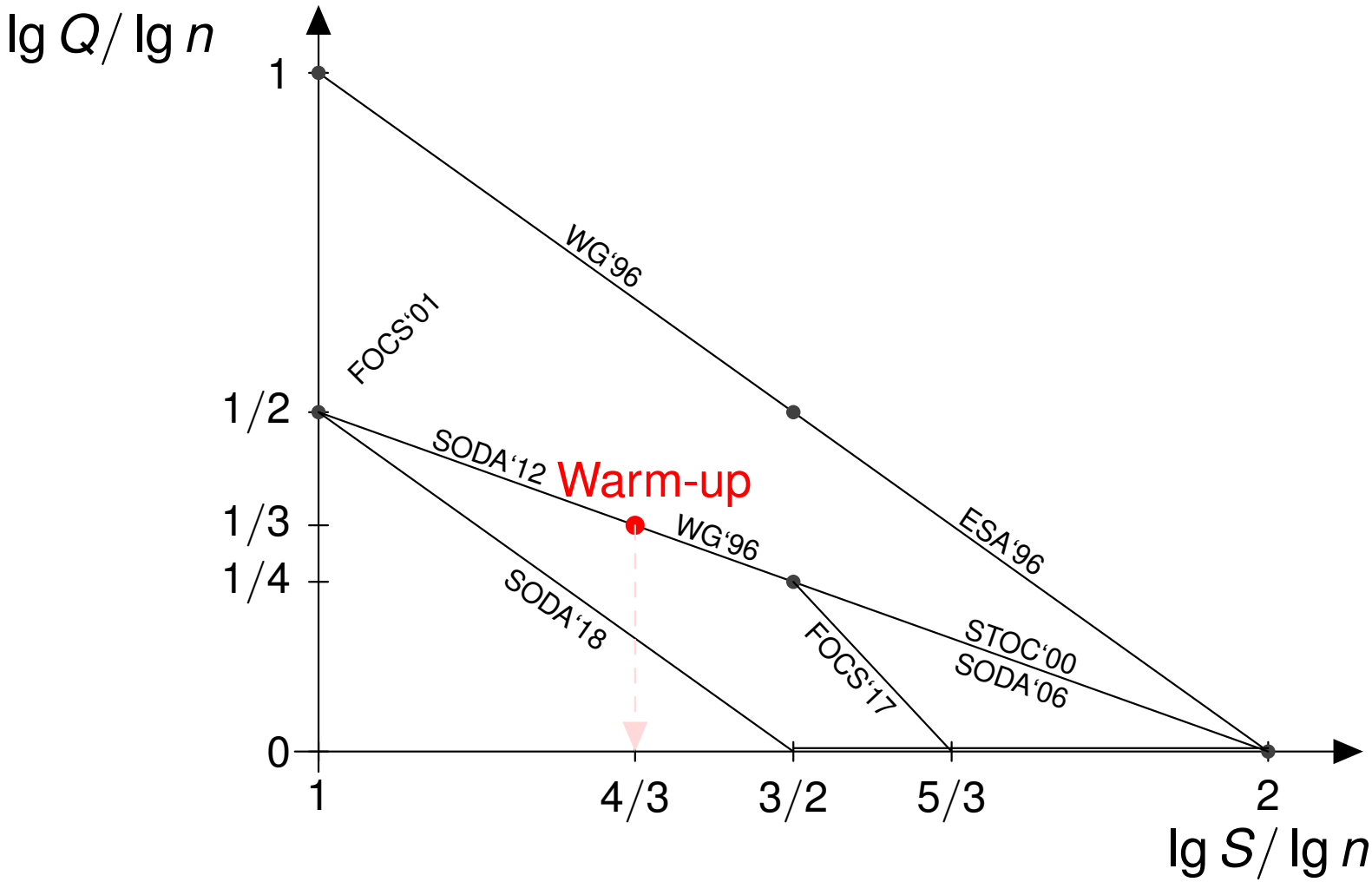
- Compute an r-division.
- For each piece P, for each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



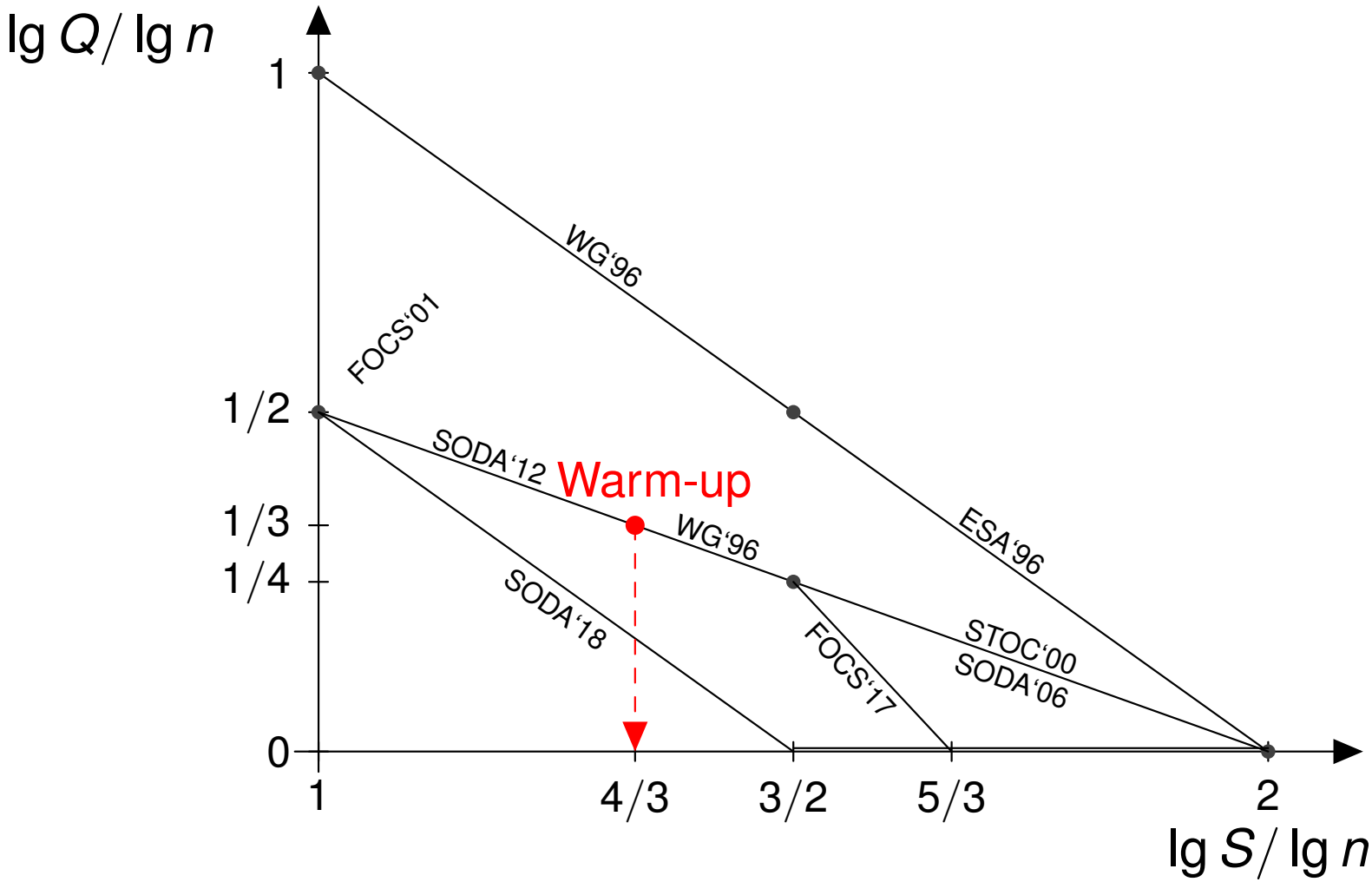
At query, try all possible $O(\sqrt{r})$ candidate boundary nodes.

The balance is at $r = n^{2/3}$.

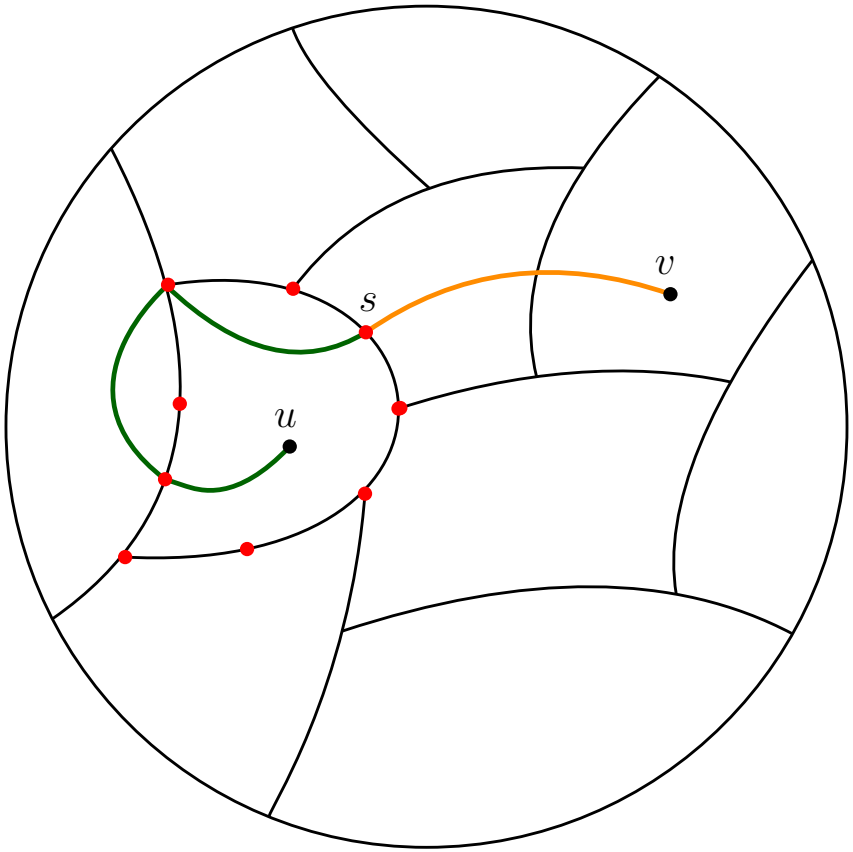
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$



Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$



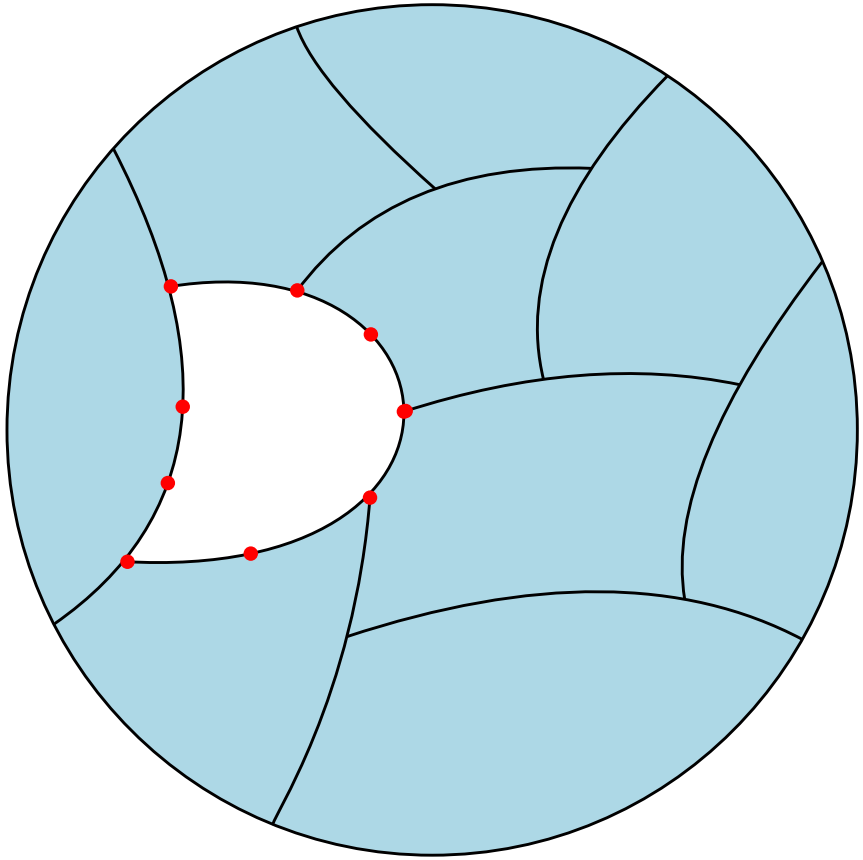
First goal



Instead of trying all possible $O(\sqrt{r}) = O(n^{1/3})$ candidate boundary nodes, we want to compute the last boundary node s visited by the shortest path in $\tilde{O}(1)$ time.

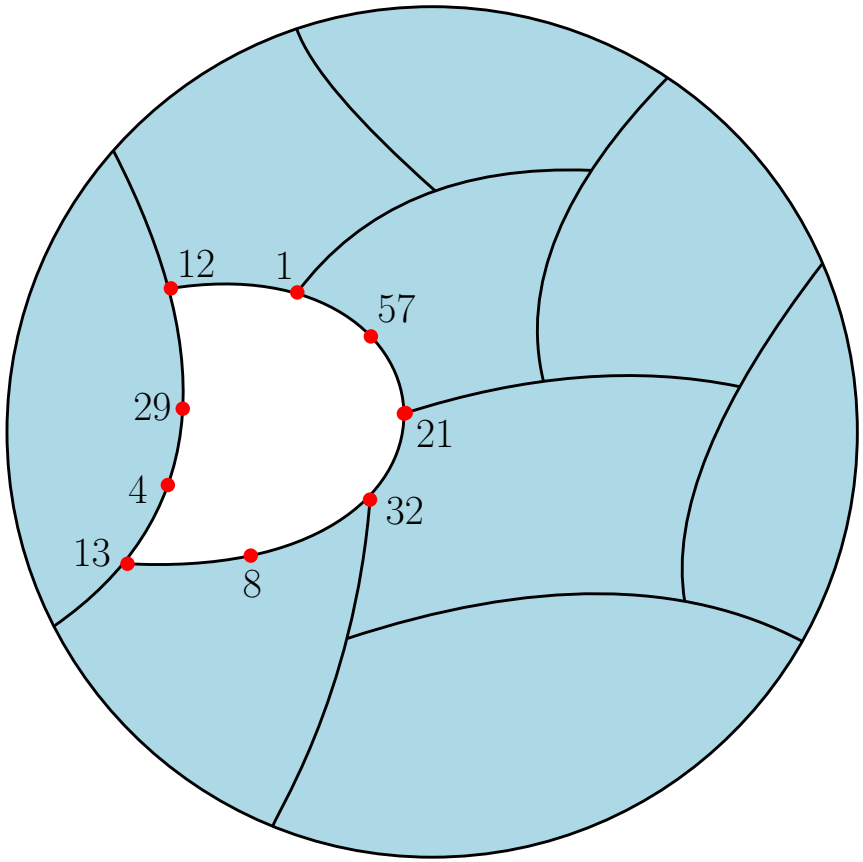
Point location

- Piece P.
- Weights $w_u(s) \geq 0$ for $s \in \partial P$.



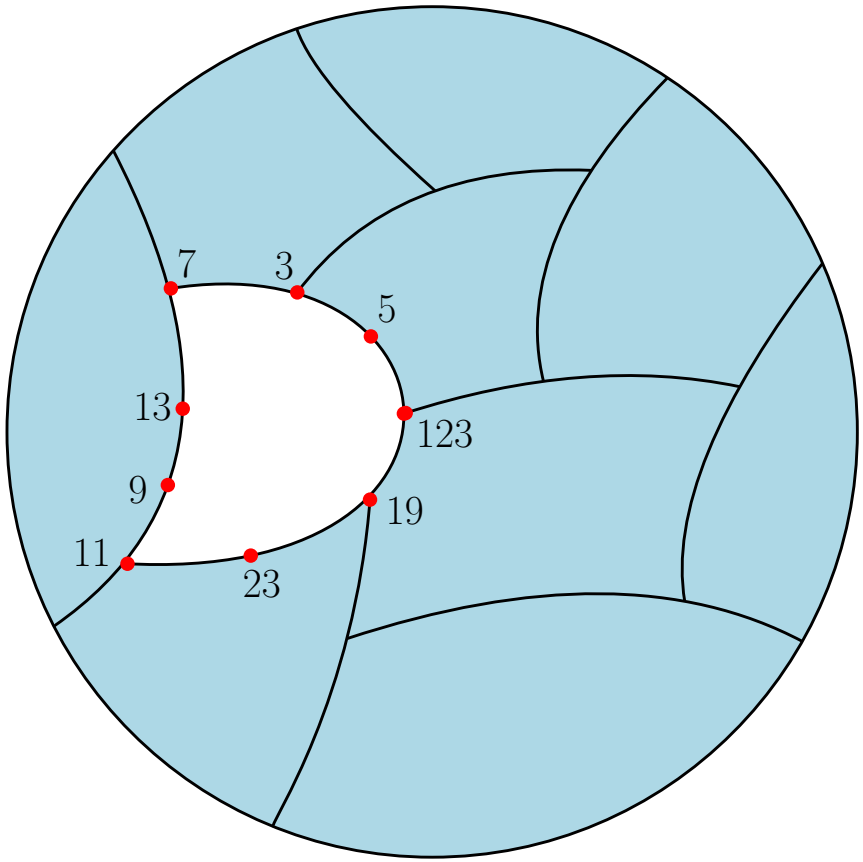
Point location

- Piece P.
- Weights $w_u(s) \geq 0$ for $s \in \partial P$.



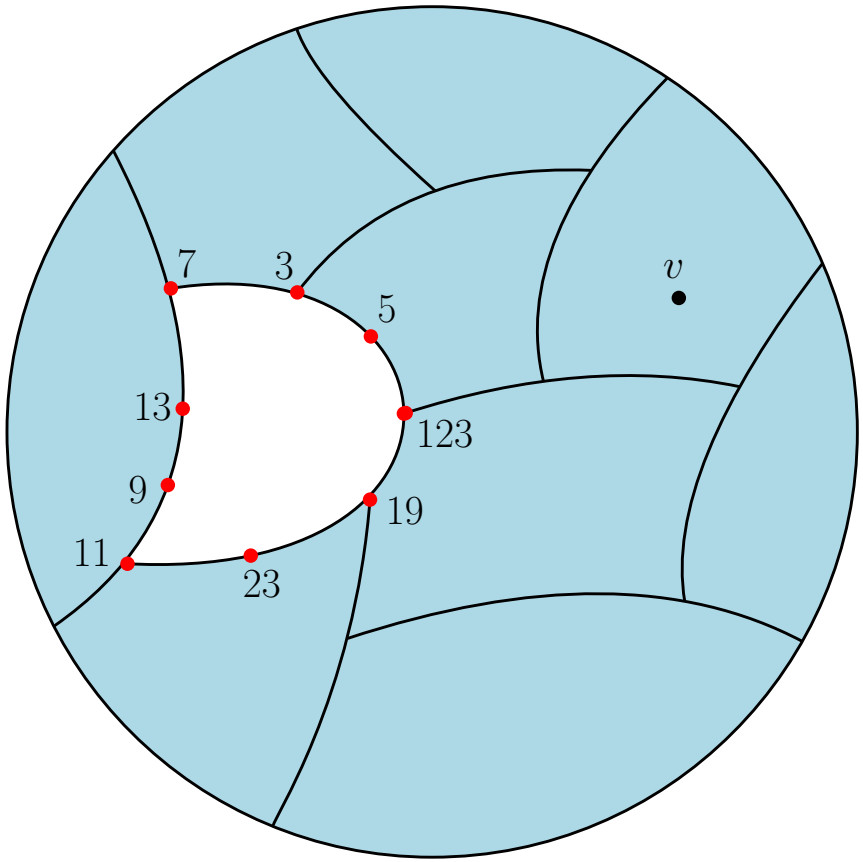
Point location

- Piece P.
- Weights $w_u(s) \geq 0$ for $s \in \partial P$.



Point location

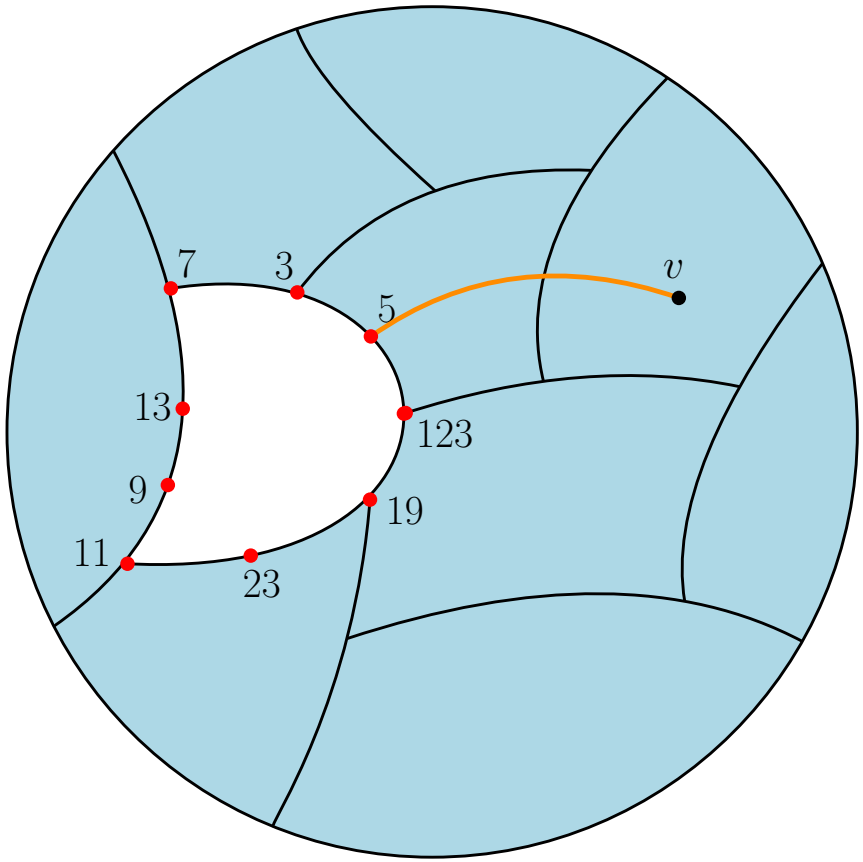
- Piece P.
- Weights $w_u(s) \geq 0$ for $s \in \partial P$.



PointLocation(u, v) query: given a vertex $v \notin P$, find site $s \in \partial P$ minimizing $w_u(s) + d(s, v)$.

Point location

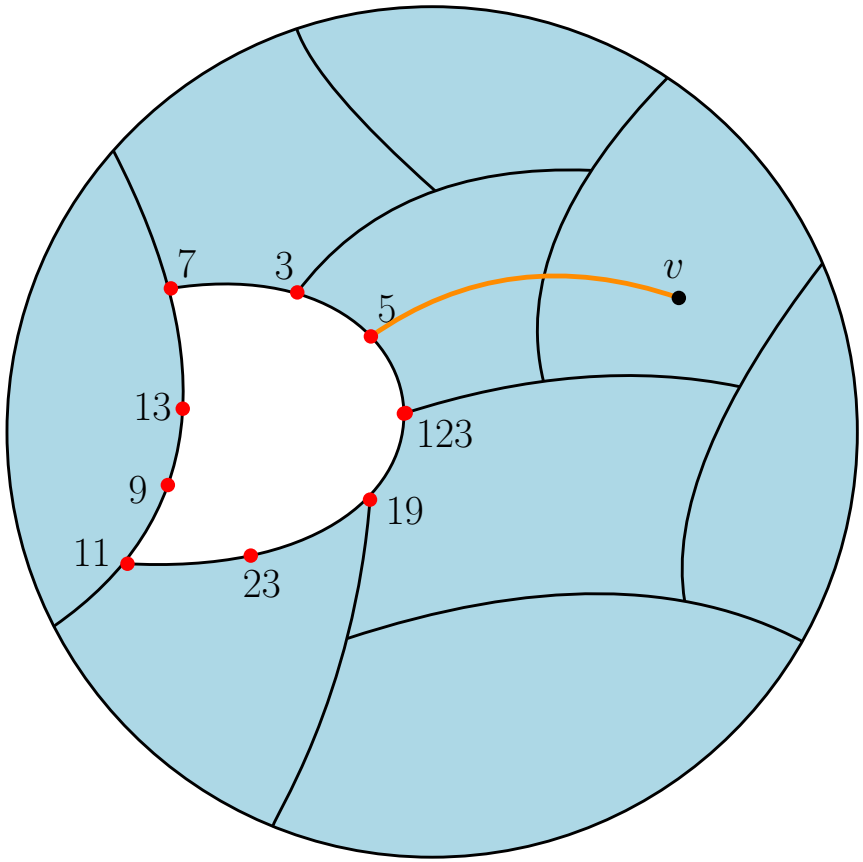
- Piece P.
- Weights $w_u(s) \geq 0$ for $s \in \partial P$.



POINTLOCATION(u, v) query: given a vertex $v \notin P$, find site $s \in \partial P$ minimizing $w_u(s) + d(s, v)$.

Point location

- Piece P.
- Weights $w_u(s) \geq 0$ for $s \in \partial P$.

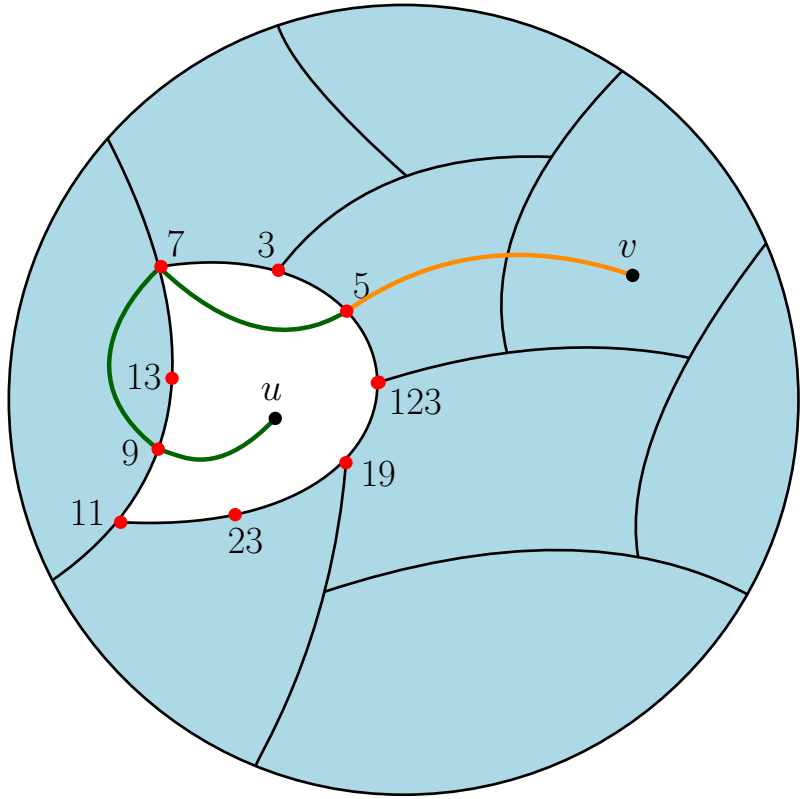


Gawrychowski et. al. [SODA'18]

Given an MSSP data structure for the outside of P, with sources ∂P, there exists an $\tilde{O}(|\partial P|)$-sized data structure for each set of additive weights for ∂P that answers point location queries in $\tilde{O}(1)$ time.

Warm-up 2: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(1)$

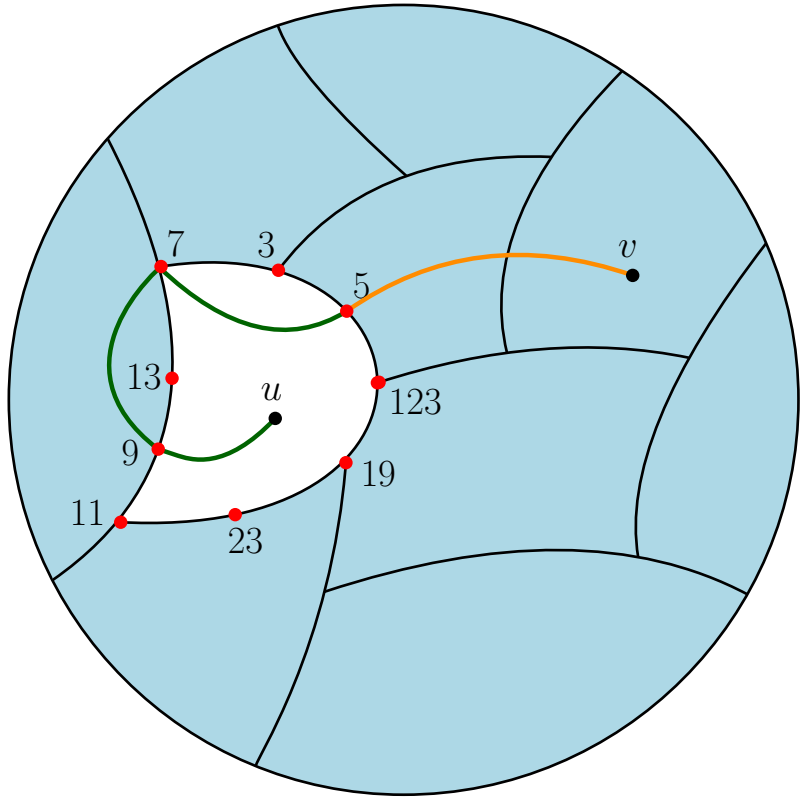
- Compute an r-division.
- For each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$.
  Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P.
  Space $\tilde{O}(n/r \cdot n)$.



At query, try all possible $O(\sqrt{r}) = O(n^{1/3})$ candidate boundary nodes.

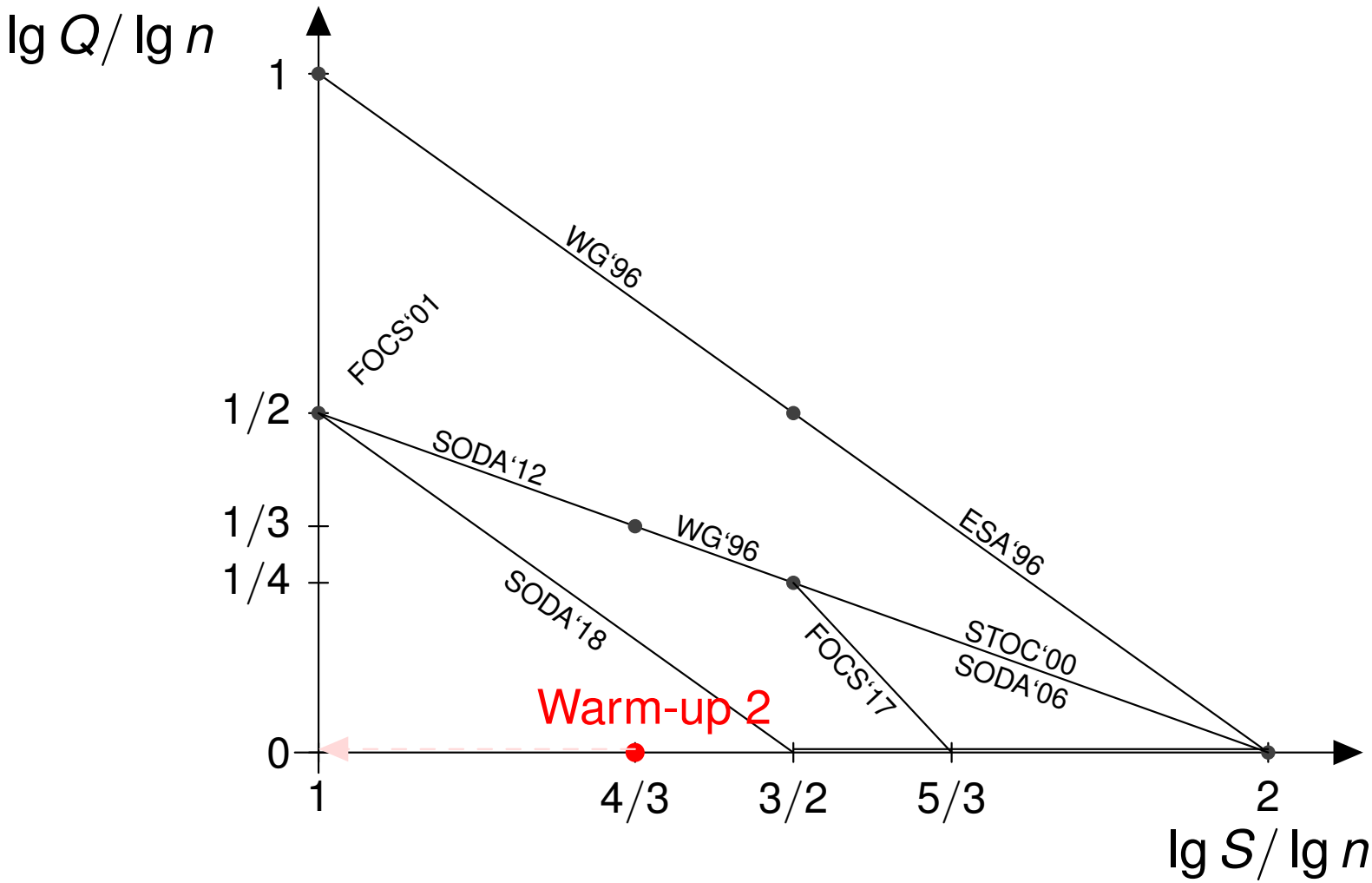
Warm-up 2: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(1)$

- Compute an r-division.
- For each node $u \in P$, store $d_G(u, p)$ for $p \in \partial P$. Preprocess these for point location.
  Space $\tilde{O}(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P.
  Space $\tilde{O}(n/r \cdot n)$.

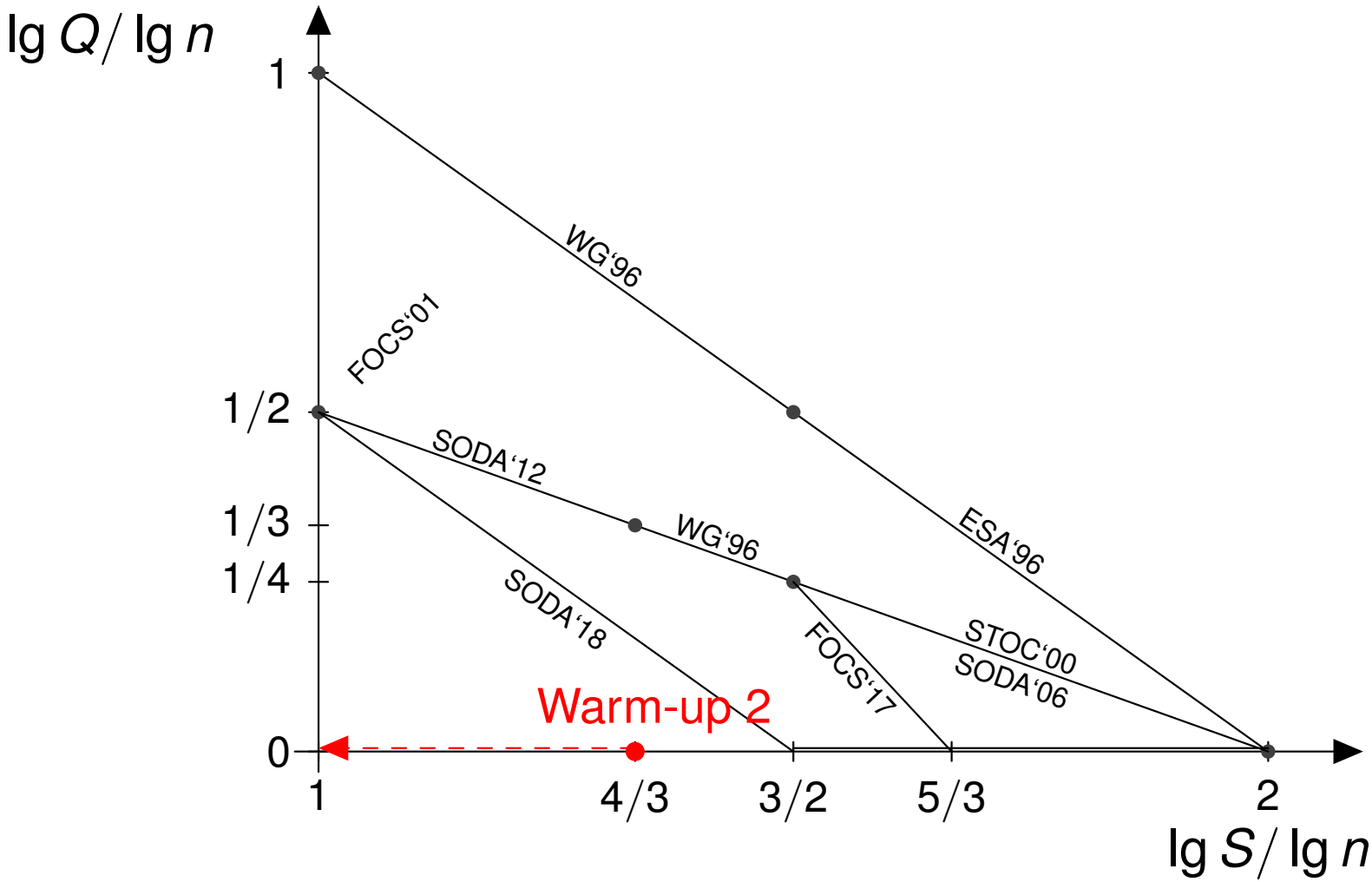


Perform point location!

Warm-up 2: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(1)$



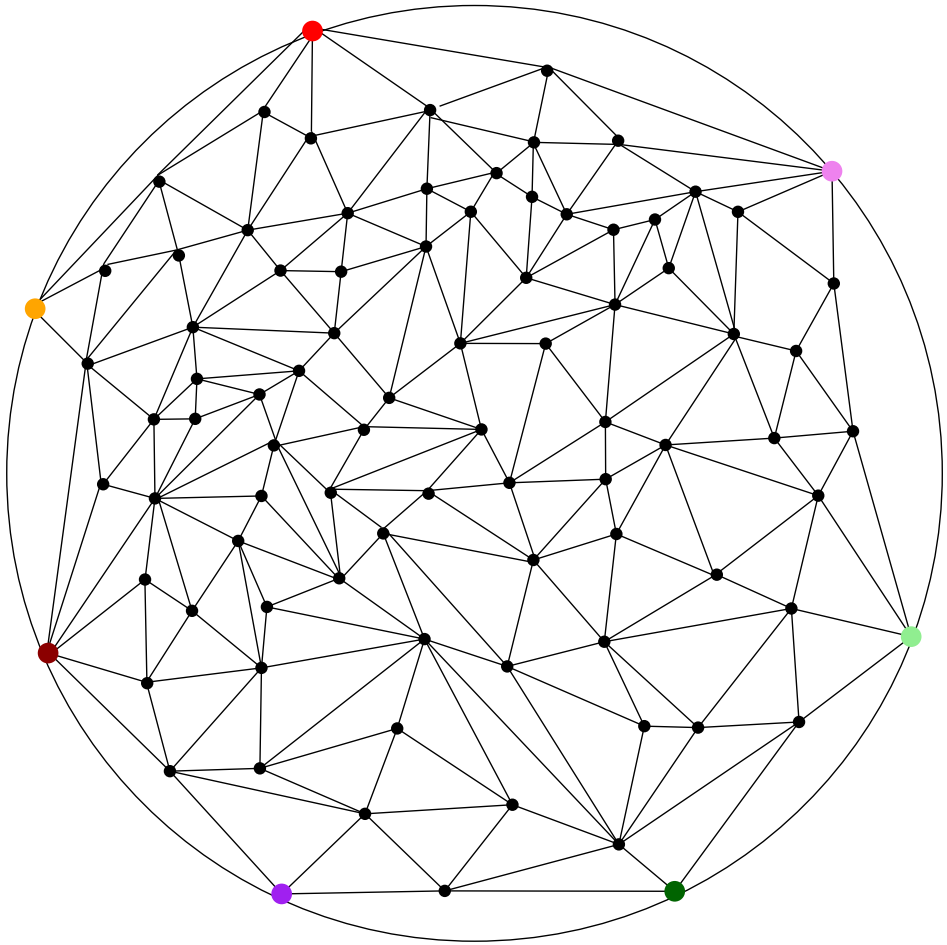
Warm-up 2: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(1)$



Additively weighted Voronoi diagrams

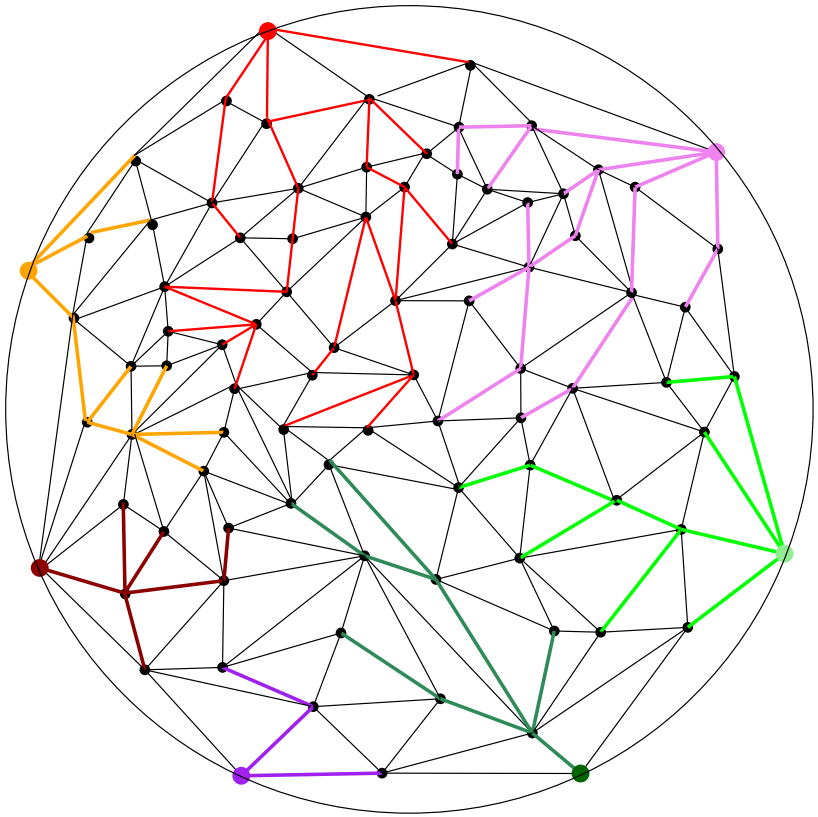
Internals of point location.

Additively weighted Voronoi diagrams



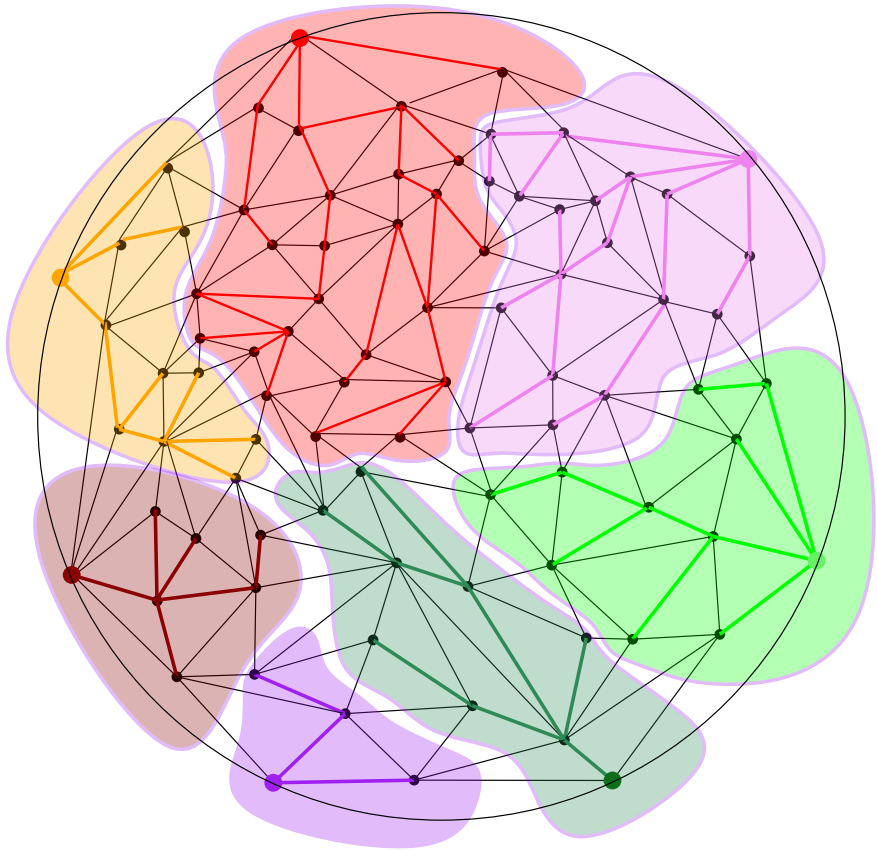
We are given weights for all nodes of ∂P, called sites.

Additively weighted Voronoi diagrams



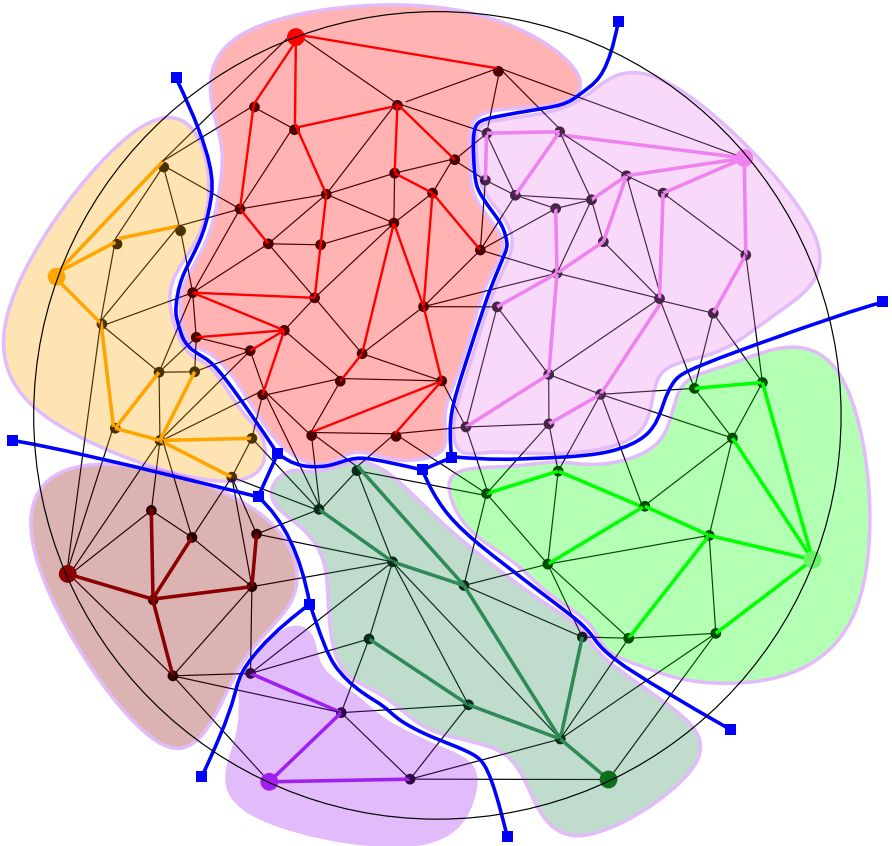
The Voronoi cell of each site consists of all nodes closer to it with respect to the additive distances.

Additively weighted Voronoi diagrams



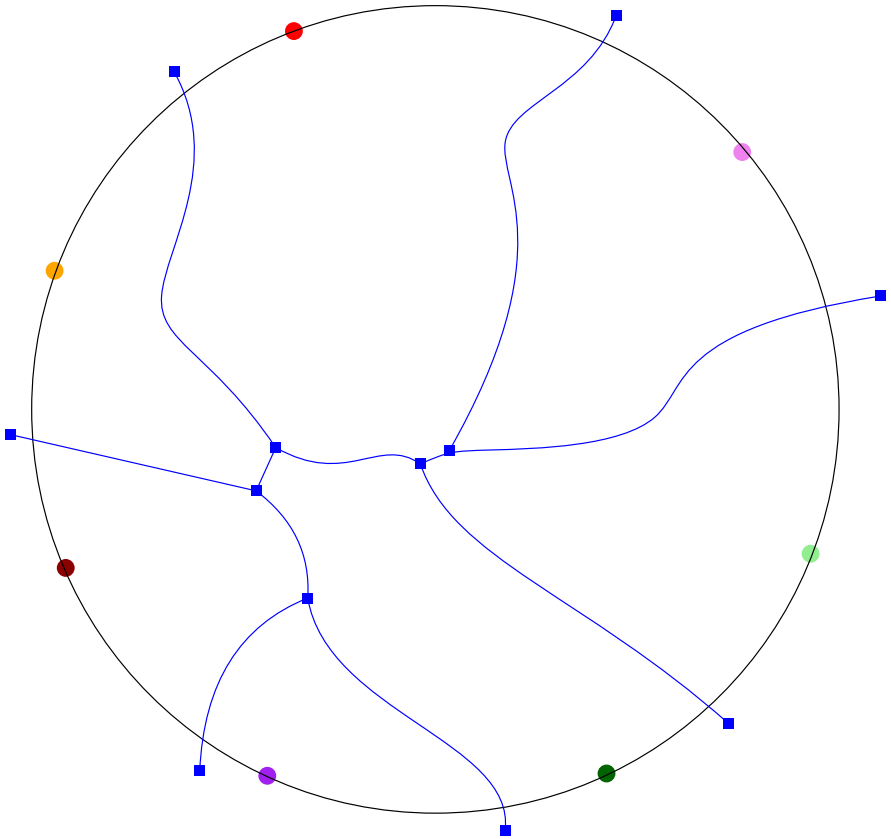
The Voronoi cell of each site consists of all nodes closer to it with respect to the additive distances.

Additively weighted Voronoi diagrams



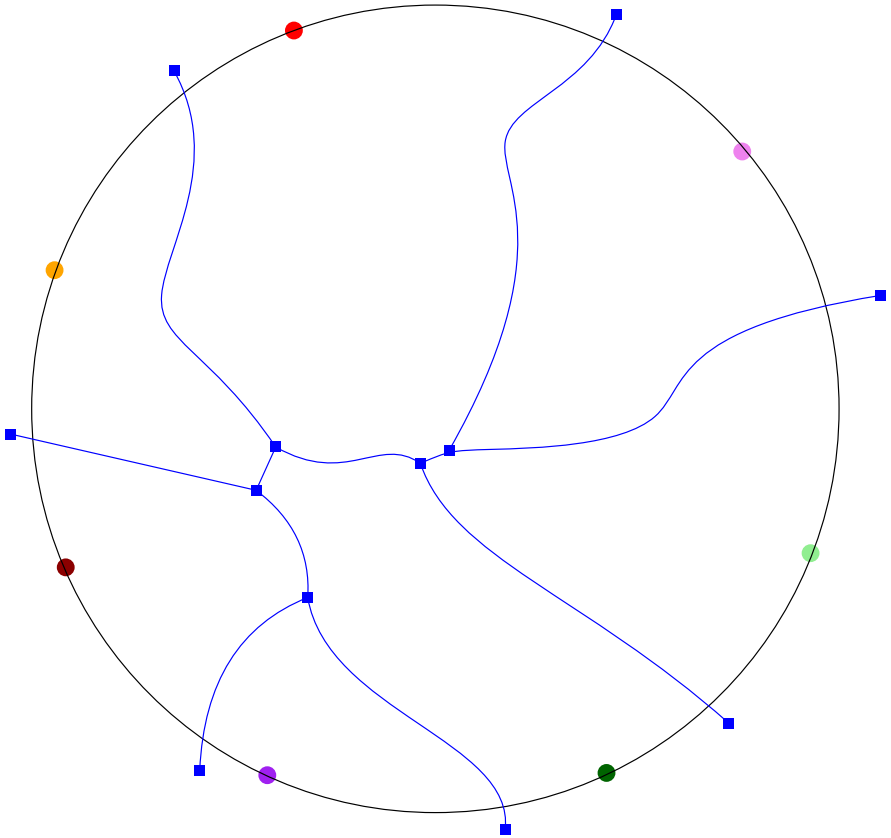
Because all sites are adjacent to one face, the diagram can be described by a tree on $O(|\partial P|) = O(\sqrt{r})$ nodes (independept of n!).

Additively weighted Voronoi diagrams



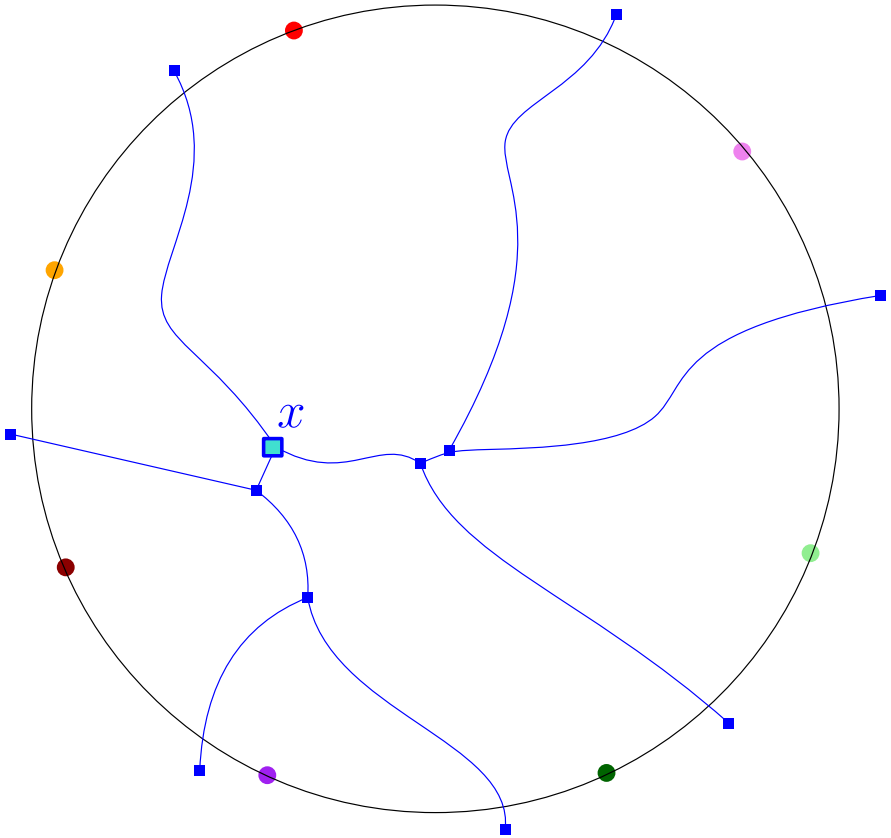
Because all sites are adjacent to one face, the diagram can be described by a tree on $O(|\partial P|) = O(\sqrt{r})$ nodes (independept of n!).

Point location



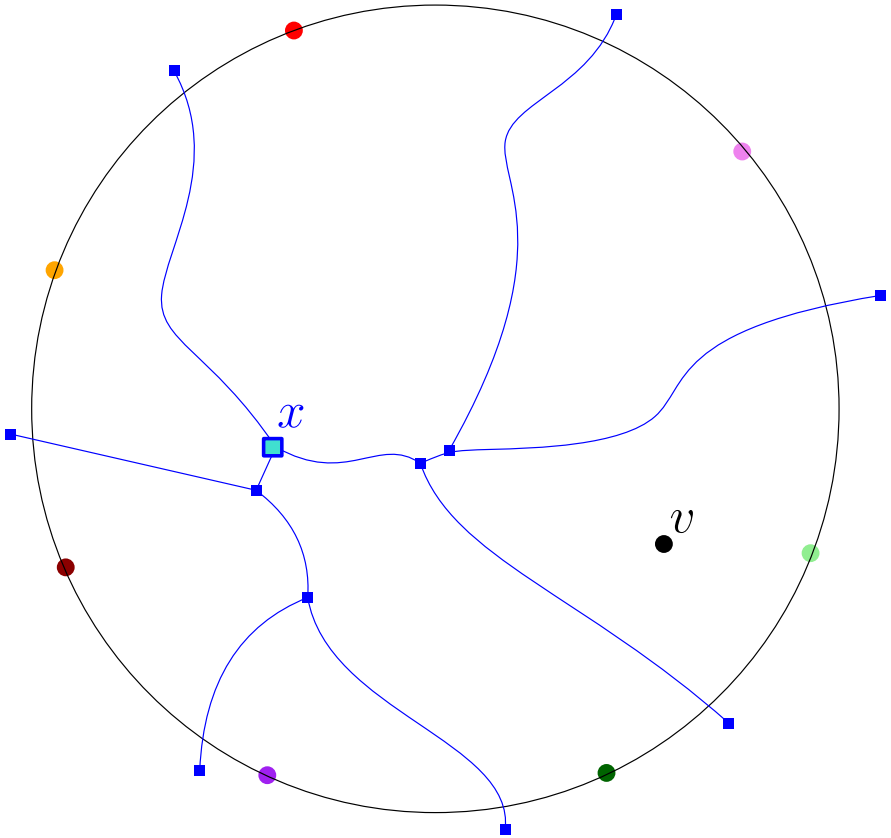
Point location

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



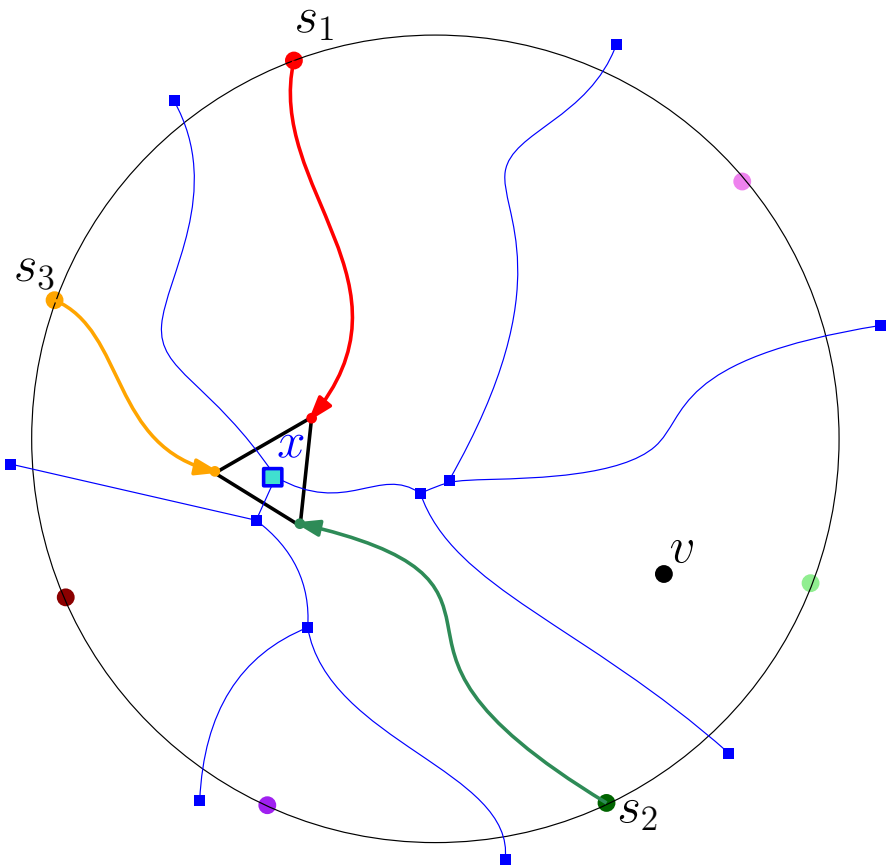
Point location

Main idea: consider the centroid. Find which subtree contains edges adjacent to the Voronoi cell containing v. Recurse.

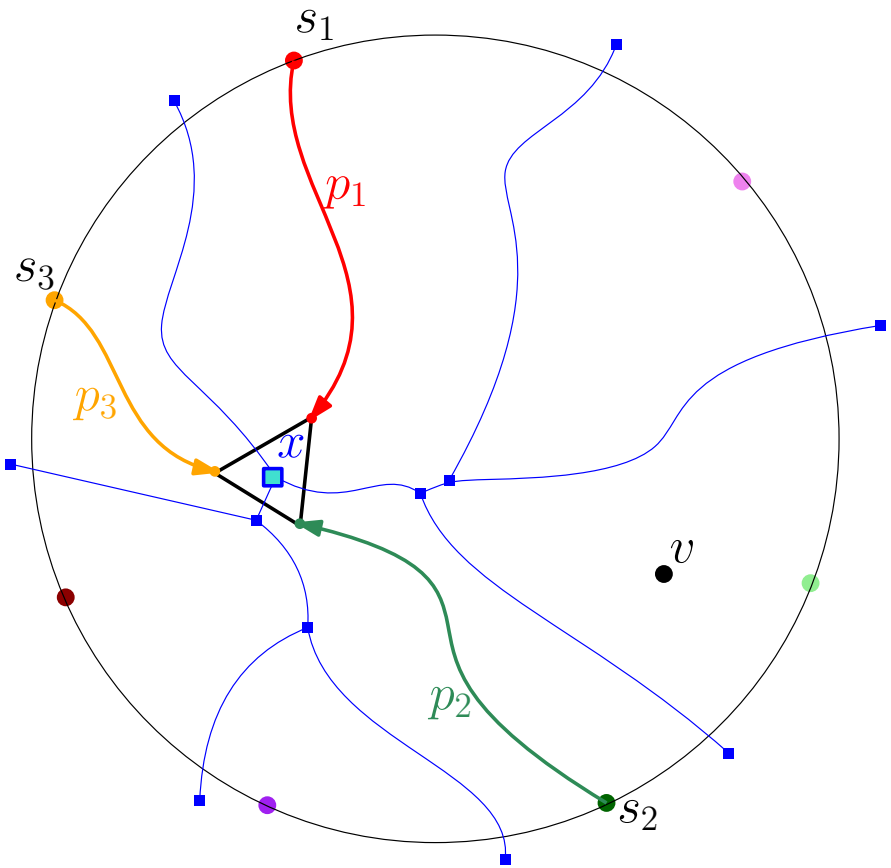


Point location

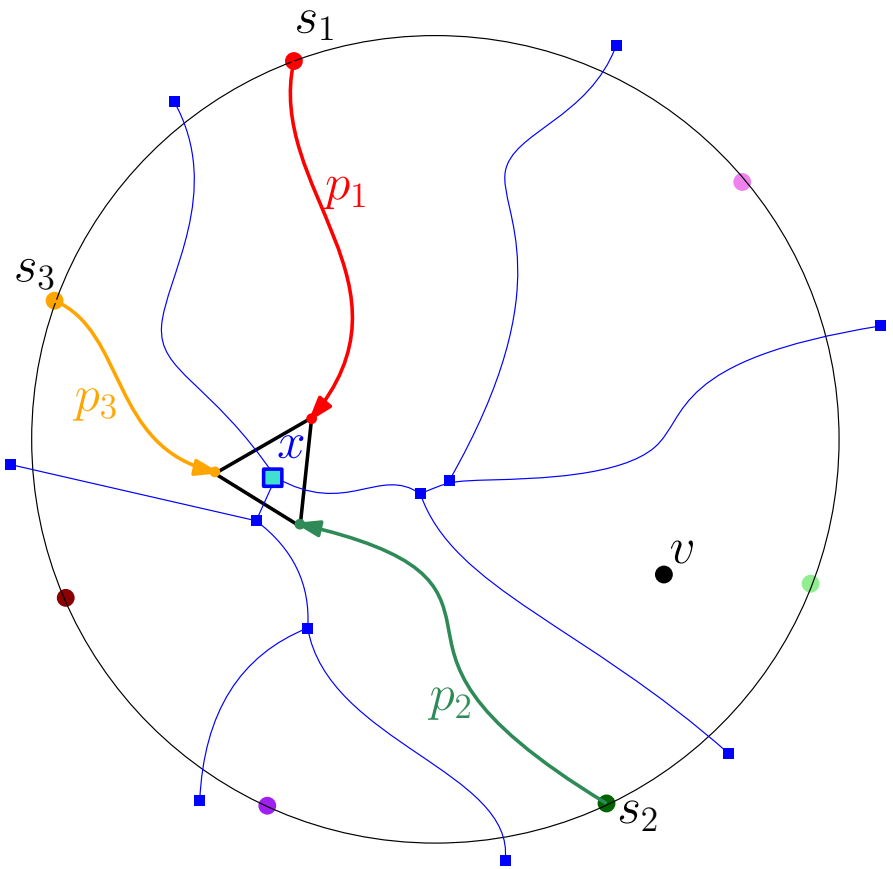
The centroid node corresponds to a trichromatic face of the original graph.



Point location

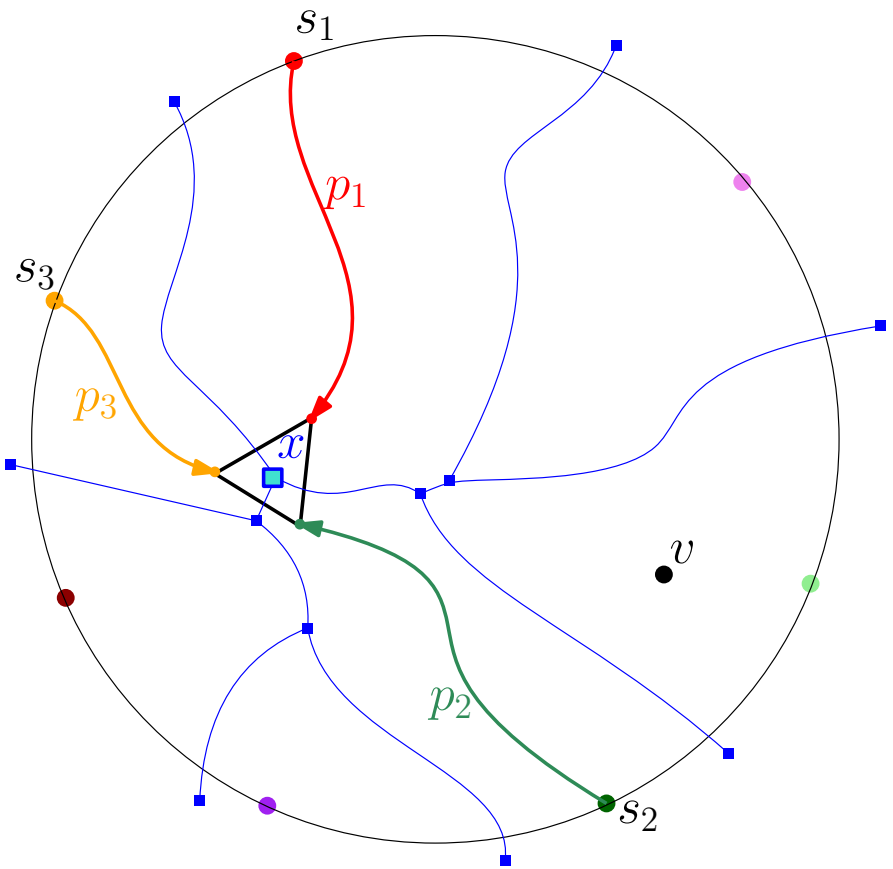


Point location



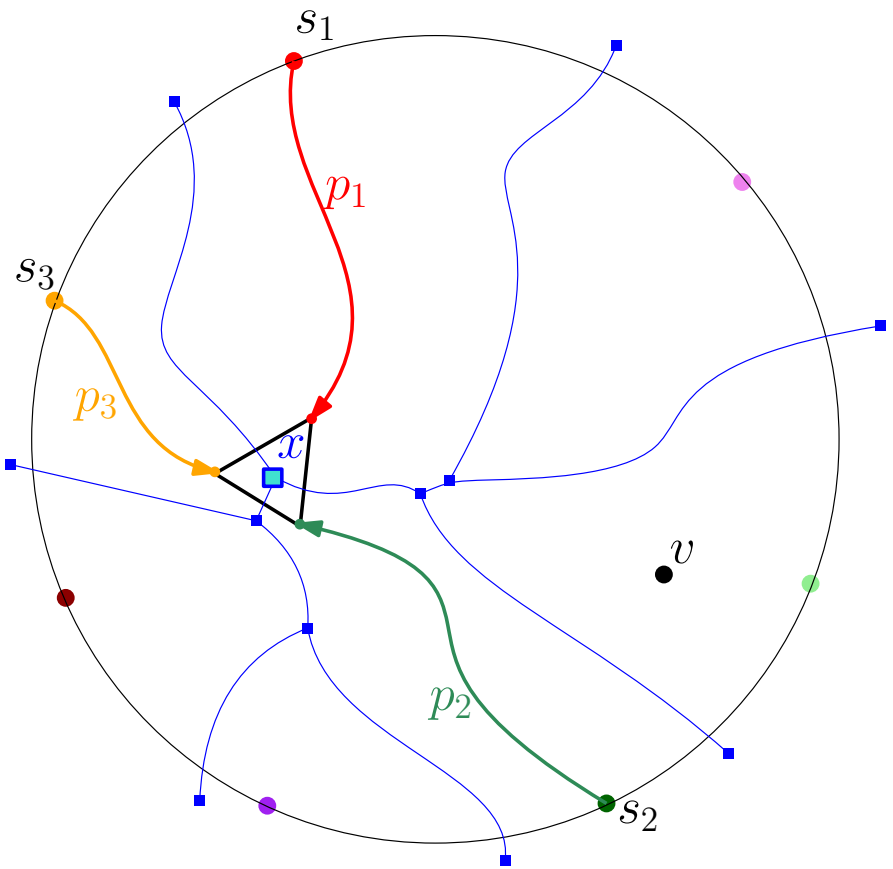
- Find which s_k among s_0, s_1, s_2 is closest to v. (distance query from boundary nodes)
- Check whether v is left or right of p_k. (left/right query)

Point location



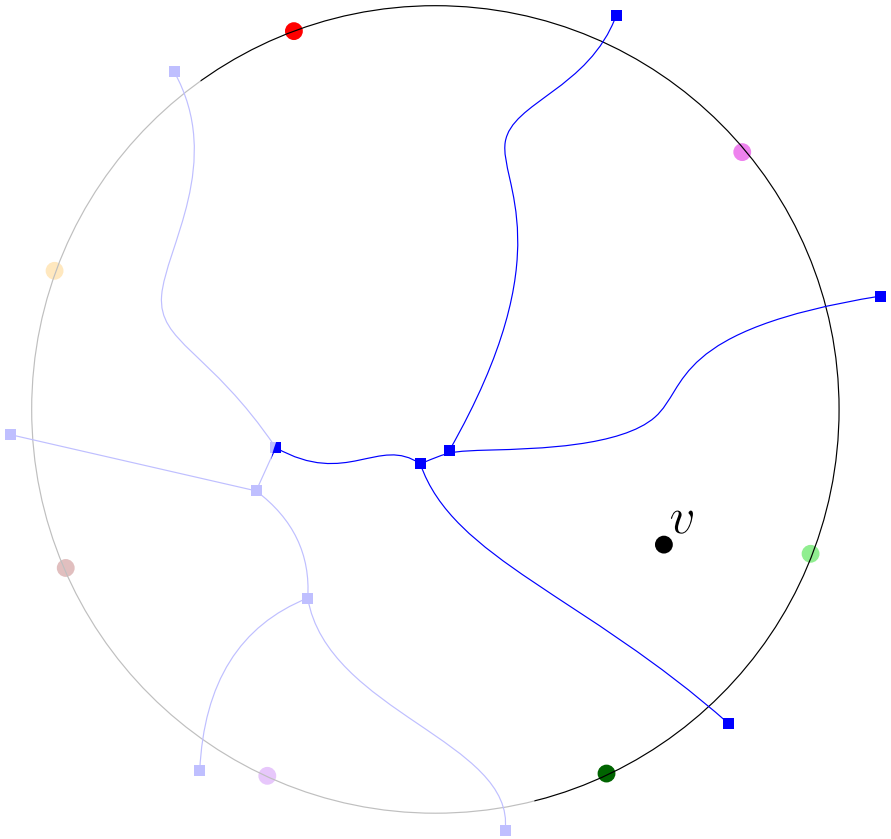
- Find which s_k among s_0, s_1, s_2 is closest to v. (distance query from boundary nodes)
- Check whether v is left or right of p_k. (left/right query)

Point location



- Find which s_k among s_0, s_1, s_2 is closest to v. (distance query from boundary nodes)
- Check whether v is left or right of p_k. (left/right query)

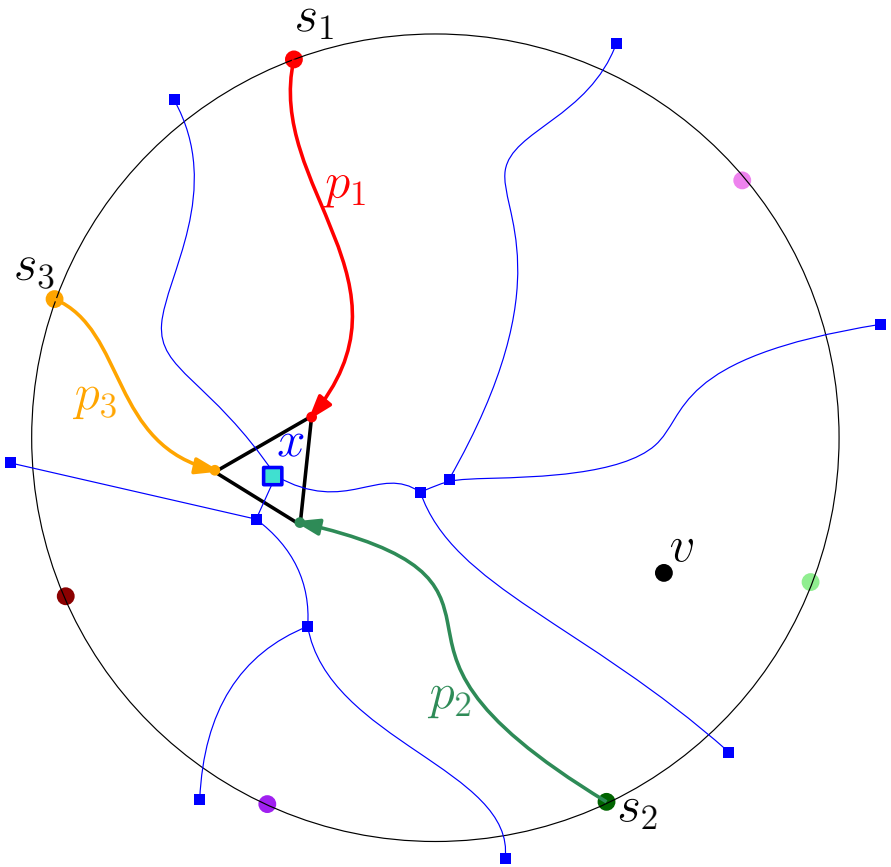
Point location



- Find which s_k among s_0, s_1, s_2 is closest to v. (distance query from boundary nodes)
- Check whether v is left or right of p_k. (left/right query)

Recurse on the remaining part of the tree.

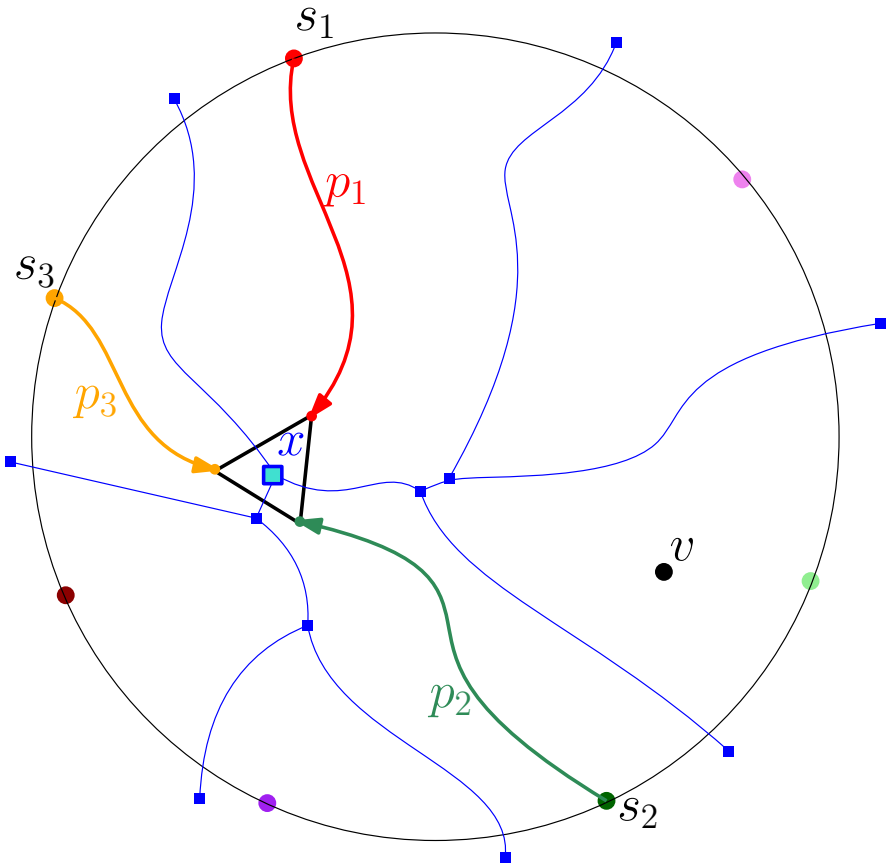
Point location



- Find which s_k among s_0, s_1, s_2 is closest to v. (distance query from boundary nodes)
- Check whether v is left or right of p_k. (left/right query)

We can answer both of these queries with an MSSP data structure.

Point location



- Find which s_k among s_0, s_1, s_2 is closest to v. (distance query from boundary nodes)
- Check whether v is left or right of p_k. (left/right query)

We can answer both of these queries with an MSSP data structure.

Second goal

Find which s_k among s_0, s_1, s_2 is closest to v. (distance query from boundary nodes)

Idea: Recursion, exploiting the structure of such queries.

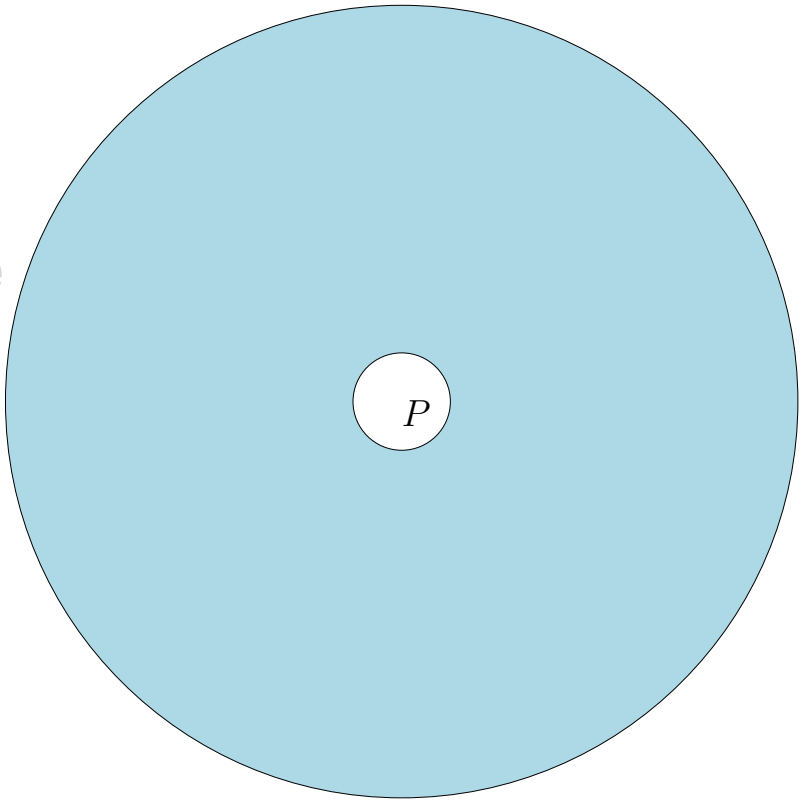
Second goal

Find which s_k among s_0, s_1, s_2 is closest to v. (distance query from boundary nodes)

Idea: Recursion, exploiting the structure of such queries.

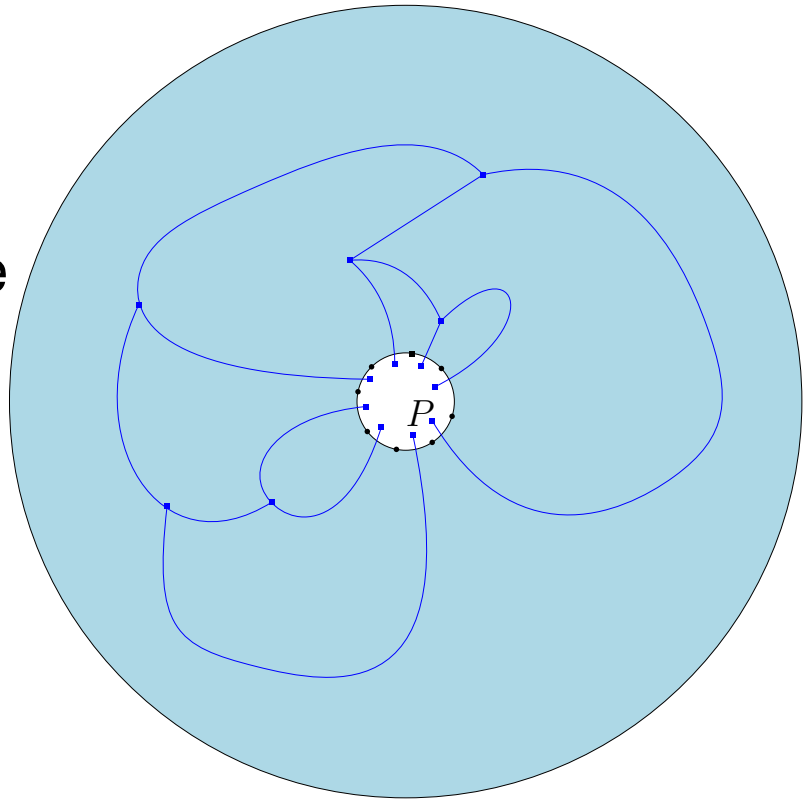
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

- Compute an r-division for $r = n^{\epsilon}$.
- For each piece P, for each node $u \in P$, store a Voronoi diagram for the outside of P with sites ∂P and additive weight $d_G(u, p)$ for $p \in \partial P$. Space $\tilde{O}(n \cdot \sqrt{r})$.
- For each piece P, store the required information to answer distance queries from ∂P to nodes outside P.



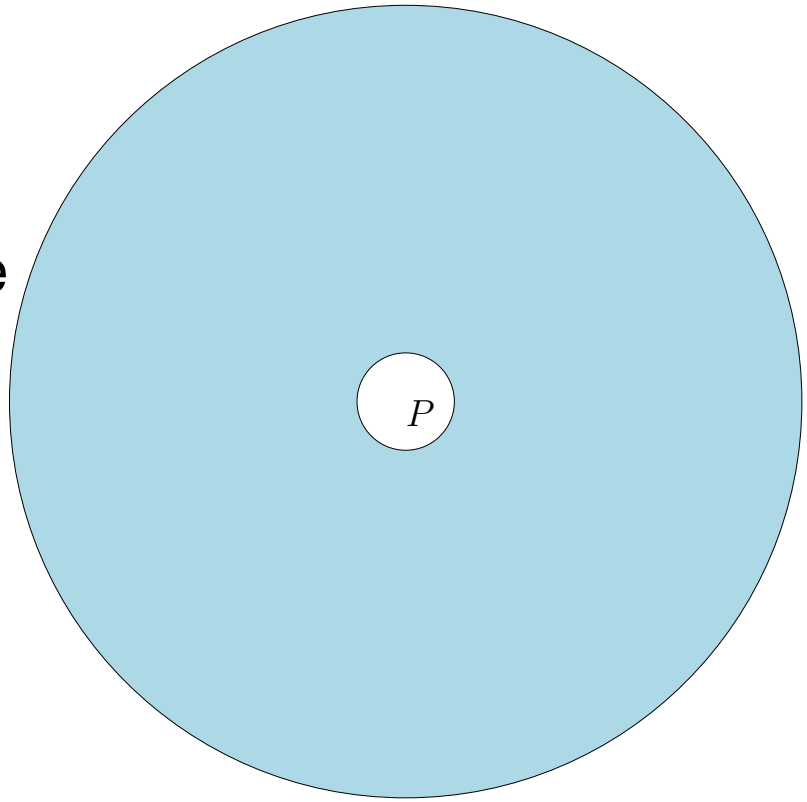
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

- Compute an r-division for $r = n^{\epsilon}$.
- For each piece P, for each node $u \in P$, store a Voronoi diagram for the outside of P with sites ∂P and additive weight $d_G(u, p)$ for $p \in \partial P$. Space $\tilde{O}(n \cdot \sqrt{r})$.
- For each piece P, store the required information to answer distance queries from ∂P to nodes outside P.



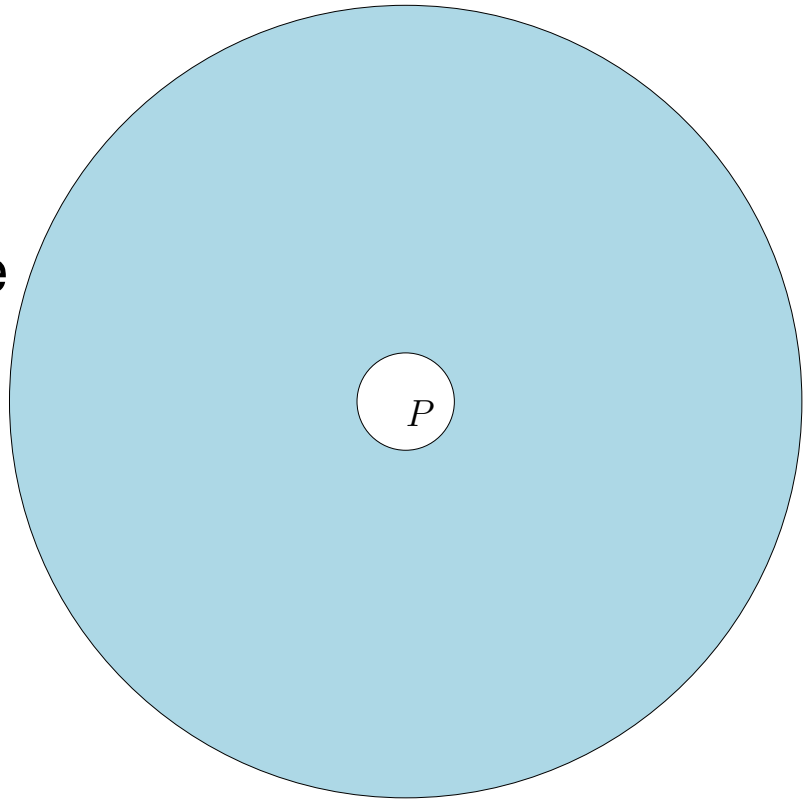
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

- Compute an r-division for $r = n^{\epsilon}$.
- For each piece P, for each node $u \in P$, store a Voronoi diagram for the outside of P with sites ∂P and additive weight $d_G(u, p)$ for $p \in \partial P$. Space $\tilde{O}(n \cdot \sqrt{r})$.
- For each piece P, store the required information to answer distance queries from ∂P to nodes outside P.



Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

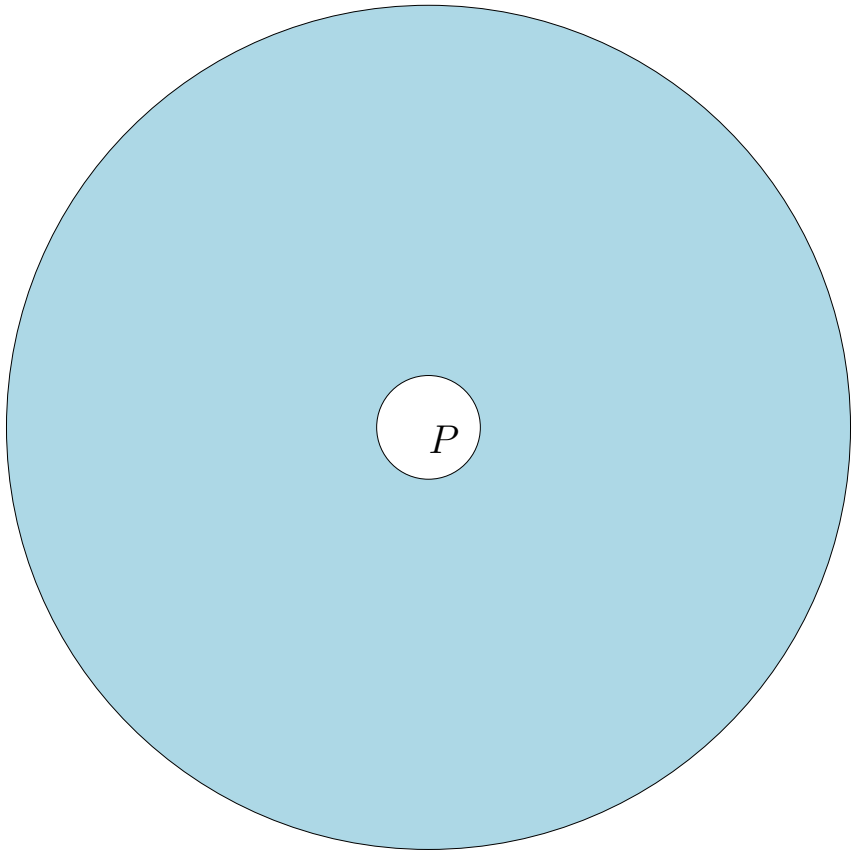
- Compute an r-division for $r = n^{\epsilon}$.
- For each piece P, for each node $u \in P$, store a Voronoi diagram for the outside of P with sites ∂P and additive weight $d_G(u, p)$ for $p \in \partial P$. Space $\tilde{O}(n \cdot \sqrt{r})$.
- For each piece P, store the required information to answer distance queries from ∂P to nodes outside P.



Perform point location.

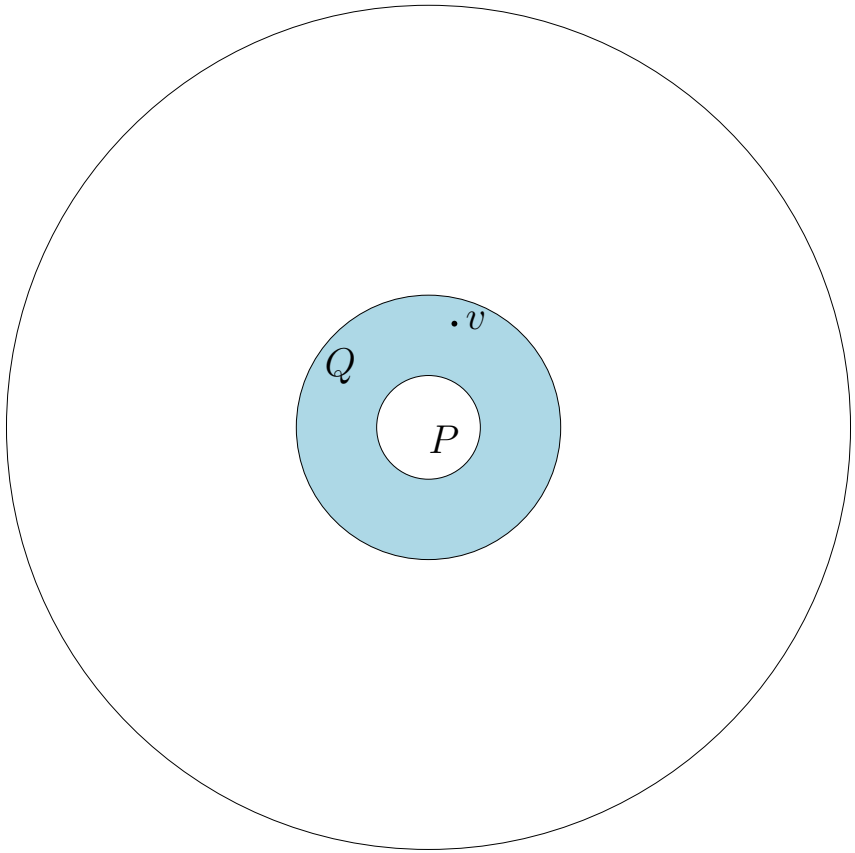
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

We have $n^{1-\epsilon}$ pieces and we can thus not afford to store an $\tilde{O}(n)$-sized MSSP for each of them.



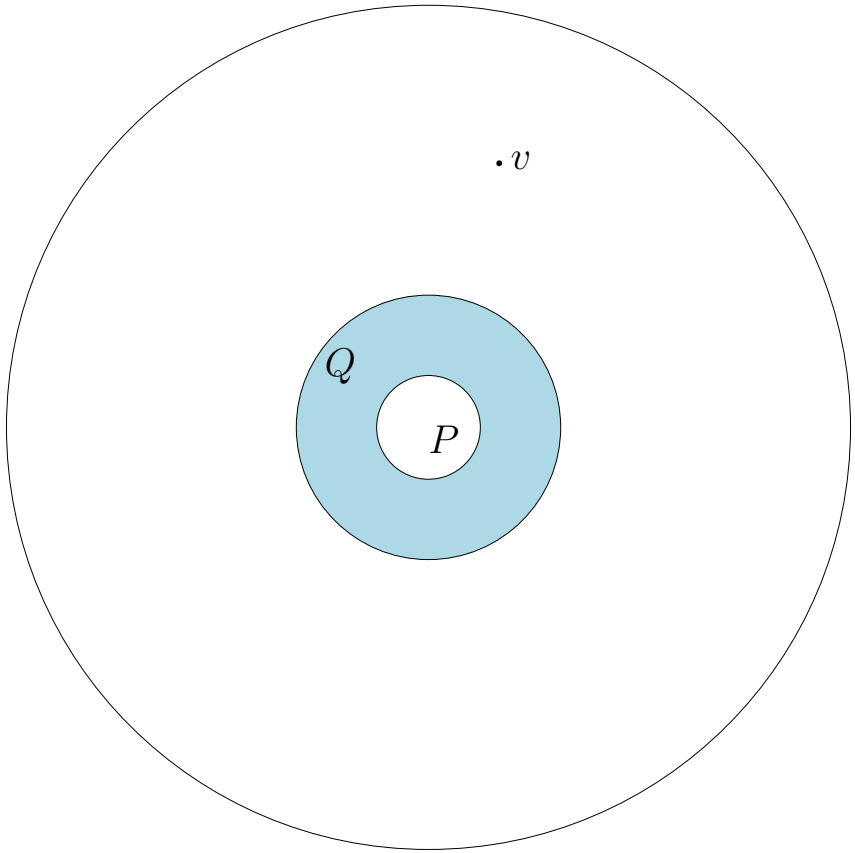
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

However, we can store an MSSP for a piece Q that contains P and belongs to a coarser r'-division, for $r' = n^{2\epsilon}$. This requires $\tilde{O}(n^{1-\epsilon} \cdot n^{2\epsilon}) = \tilde{O}(n^{1+\epsilon})$ space, and handles the case $v \in Q$.



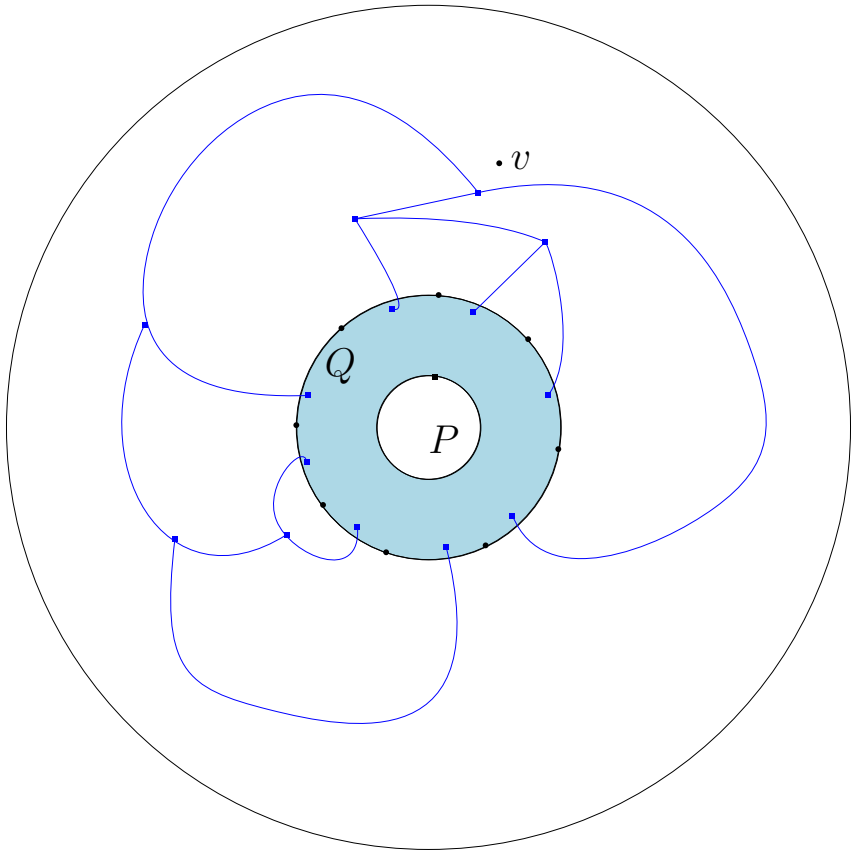
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

To handle the case $v \notin Q$, each $p \in \partial P$ stores a Voronoi diagram for the outside of Q with sites ∂Q. This requires $\tilde{O}(n^{1-\epsilon} \cdot n^{\epsilon/2} \cdot n^{2\epsilon/2}) = = \tilde{O}(n^{1+\epsilon/2})$ space. Recursively support point location outside Q.



Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

To handle the case $v \notin Q$, each $p \in \partial P$ stores a Voronoi diagram for the outside of Q with sites ∂Q. This requires $\tilde{O}(n^{1-\epsilon} \cdot n^{\epsilon/2} \cdot n^{2\epsilon/2}) = = \tilde{O}(n^{1+\epsilon/2})$ space. Recursively support point location outside Q.

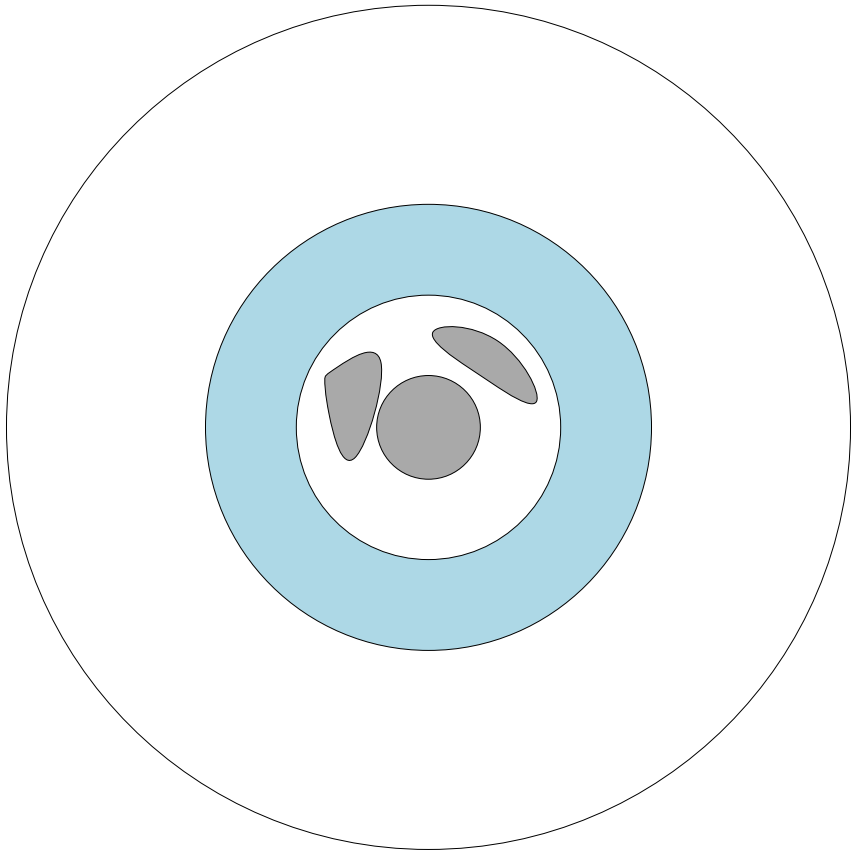


Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

Repeat the same reasoning for increasingly larger pieces of sizes $n^{i\cdot\epsilon}$, for $i = 1, \ldots, 1/\epsilon$. There are $n^{1-i\epsilon}$ pieces at level i, each stores MSSP and Voronoi diagrams of size $\tilde{O}(n^{(i+1)\epsilon})$. Total space is $\tilde{O}(\epsilon^{-1} n^{1+\epsilon})$.

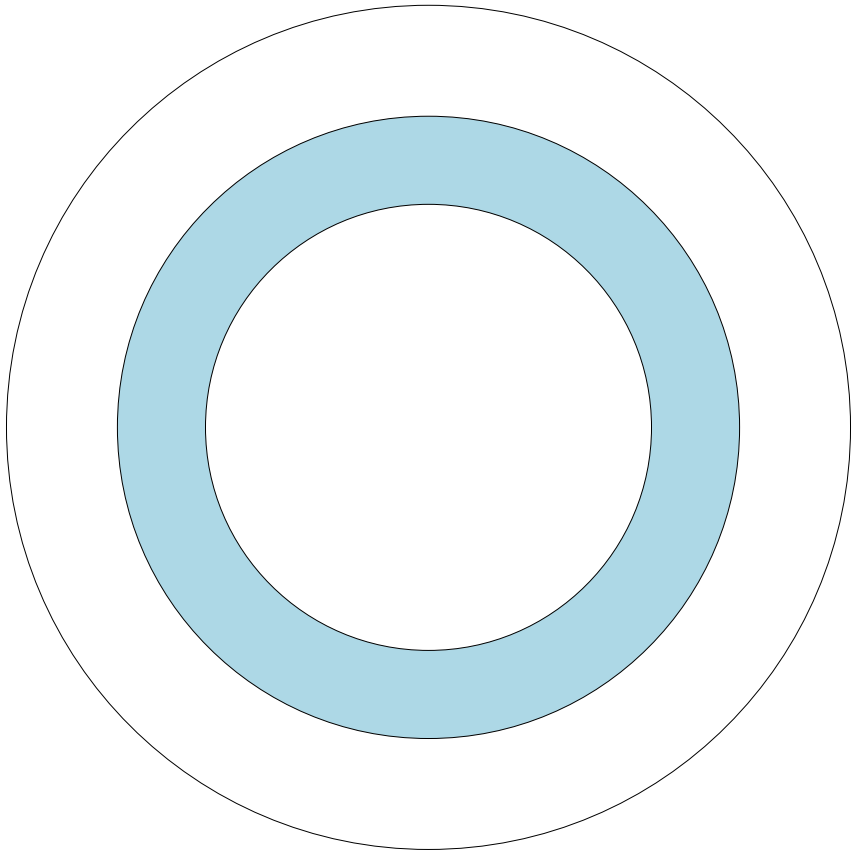
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

The idea is that smaller pieces share the cost of MSSP data structures at higher levels.



Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

The idea is that smaller pieces share the cost of MSSP data structures at higher levels.

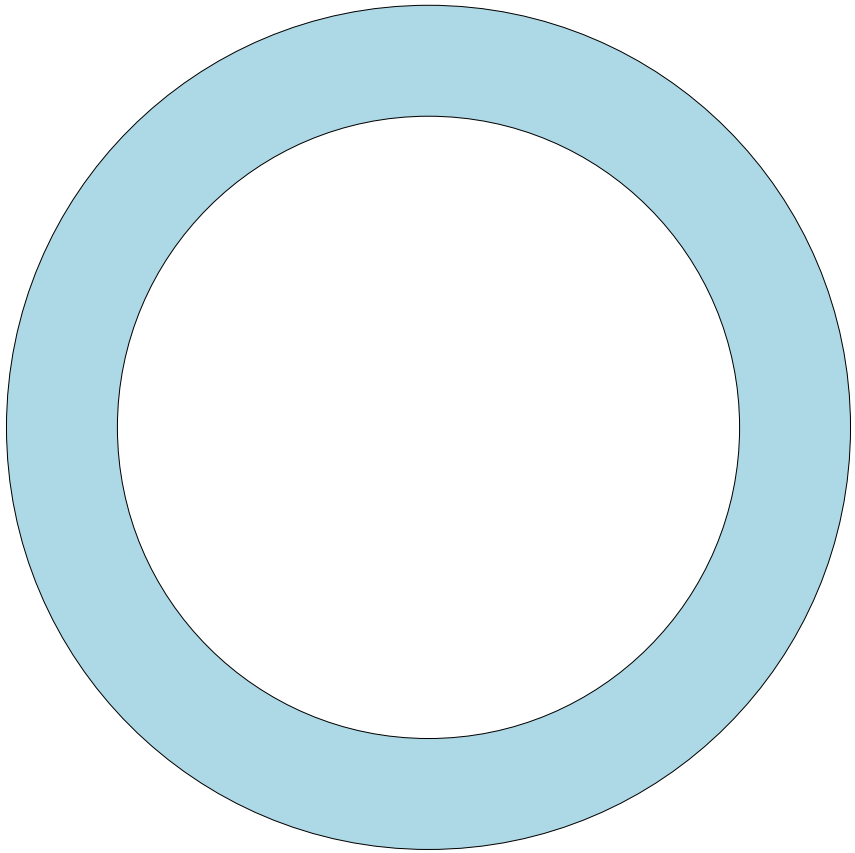


Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

Each point location query, either gets answered at the current level, or reduces to $O(\log n)$ point location queries at a higher level.

Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

If not earlier, then in the top level we can answer the query in $O(\log^2 n)$ time. So the overall query time is $O(\log^{1/\epsilon} n)$.



Tradeoffs and construction time

We show the following tradeoffs for $\langle S, Q \rangle$:

1. $\langle \tilde{O}(n^{1+\epsilon}), O(\log^{1/\epsilon} n) \rangle$, for any constant $1/2 \geq \epsilon > 0$;
2. $\langle O(n \log^{2+1/\epsilon} n), \tilde{O}(n^{\epsilon}) \rangle$, for any constant $\epsilon > 0$;
3. $\langle n^{1+o(1)}, n^{o(1)} \rangle$.

Some of the issues I shoved under the rug:

- handling left/right queries.
- ∂P is not a single face of P (holes).
- constructing these oracles in $O(n^{3/2+\epsilon})$ time.

Tradeoffs and construction time

We show the following tradeoffs for $\langle S, Q \rangle$:

1. $\langle \tilde{O}(n^{1+\epsilon}), O(\log^{1/\epsilon} n) \rangle$, for any constant $1/2 \geq \epsilon > 0$;
2. $\langle O(n \log^{2+1/\epsilon} n), \tilde{O}(n^{\epsilon}) \rangle$, for any constant $\epsilon > 0$;
3. $\langle n^{1+o(1)}, n^{o(1)} \rangle$.

Some of the issues I shoved under the rug:

- handling left/right queries.
- ∂P is not a single face of P (holes).
- constructing these oracles in $O(n^{3/2+\epsilon})$ time.

Tradeoffs and construction time

We show the following tradeoffs for $\langle S, Q \rangle$:

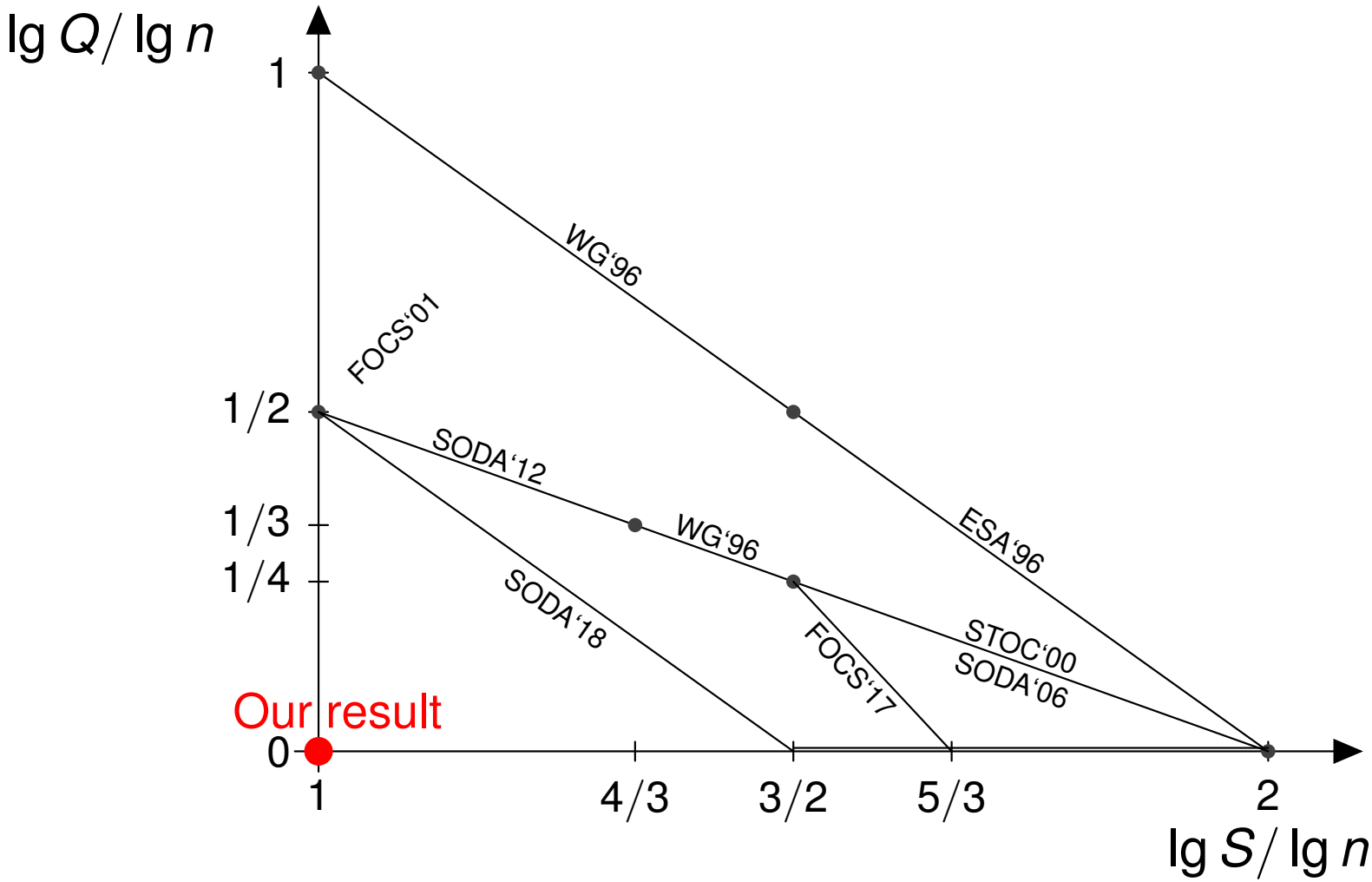
1. $\langle \tilde{O}(n^{1+\epsilon}), O(\log^{1/\epsilon} n) \rangle$, for any constant $1/2 \geq \epsilon > 0$;
2. $\langle O(n \log^{2+1/\epsilon} n), \tilde{O}(n^{\epsilon}) \rangle$, for any constant $\epsilon > 0$;
3. $\langle n^{1+o(1)}, n^{o(1)} \rangle$.

Some of the issues I shoved under the rug:

- handling left/right queries.
- ∂P is not a single face of P (holes).
- constructing these oracles in $O(n^{3/2+\epsilon})$ time.

Open problems

- Can we get $\tilde{O}(n)$ space and $\tilde{O}(1)$ query time?
- Can we get the construction time to be $\tilde{O}(n)$?
- Improvements on dynamic distance oracels?
  currently:
  1. exact: UB $\tilde{O}(n^{2/3})$; LB $\tilde{O}(n^{1/2})$ (conditioned on APSP)
  2. approx.: UB $\tilde{O}(n^{1/2})$ (undirected) ; no LB.


lg Q/ lg n
1
1/2
1/3
1/4
0
1
4/3
3/2
5/3
2
lg S/ lg n
WG'96
FOCS'01
SODA'12
WG'96
ESA'96
SODA'18
FOCS'17
STOC'00
SODA'06
Our result

Questions?