Almost Optimal Distance Oracles for Planar Graphs

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

[1]King's College London, UK

[2]Interdisciplinary Center Herzliya, Israel

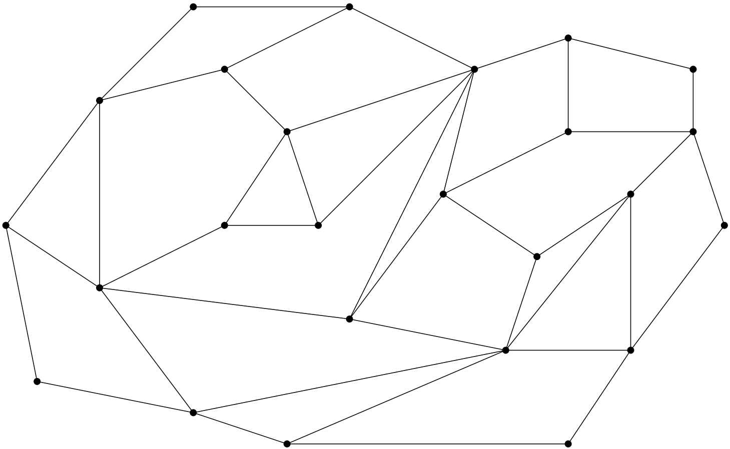
[3]University of Wrocław, Poland

[4]University of Haifa, Israel

STOC 2019

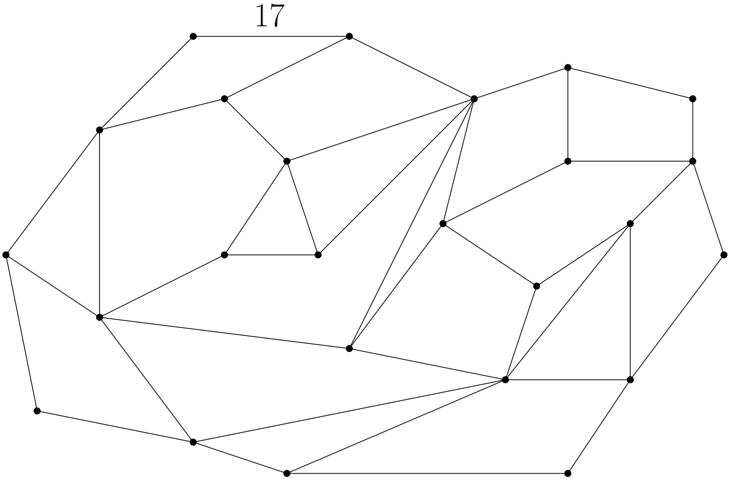
Phoenix, Arizona.

Problem definition



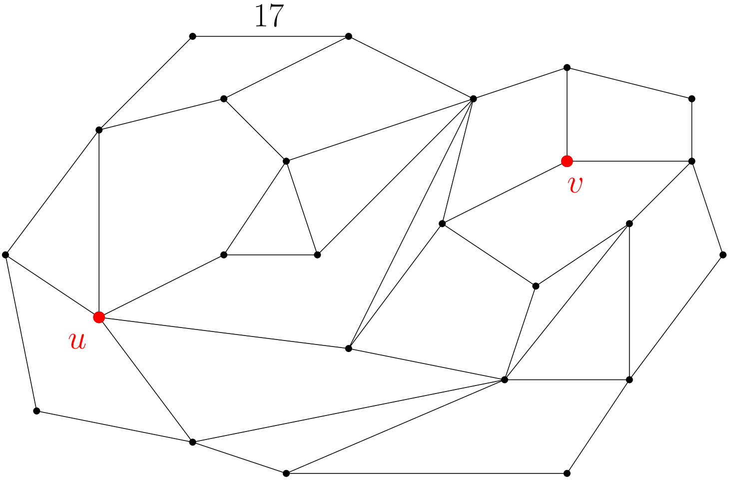
Preprocess an n-vertex planar graph $G = (V, E)$ with nonnegative arc lengths, so that given any $u, v \in V$ we can compute $d(u, v)$ efficiently.

Problem definition



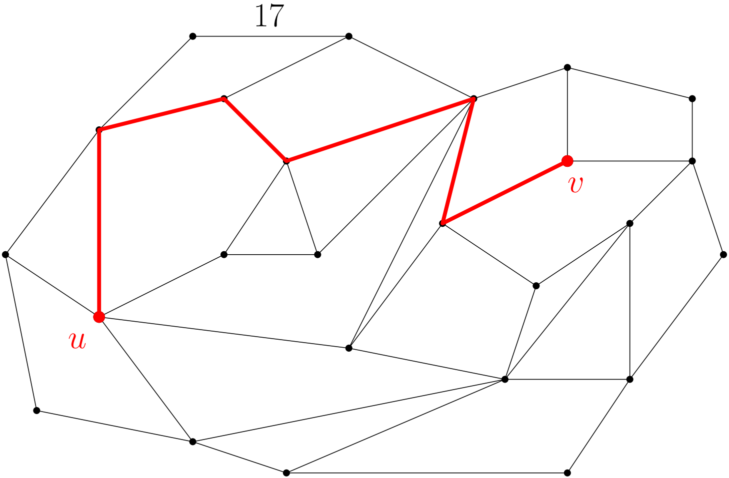
Preprocess an n-vertex planar graph $G = (V, E)$ with nonnegative arc lengths, so that given any $u, v \in V$ we can compute $d(u, v)$ efficiently.

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

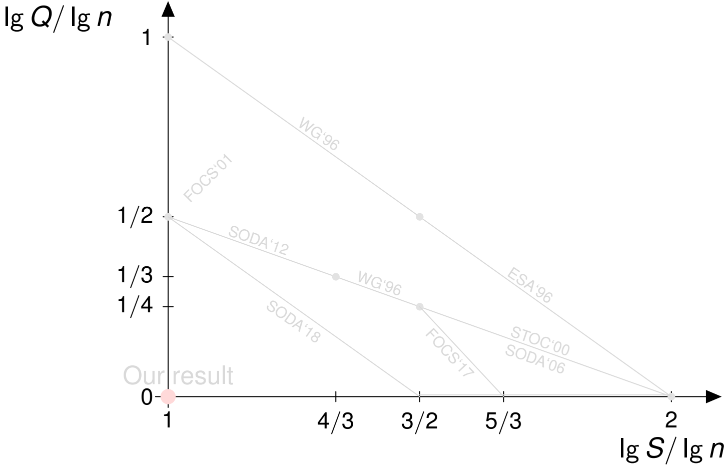
Ideally:

- 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.

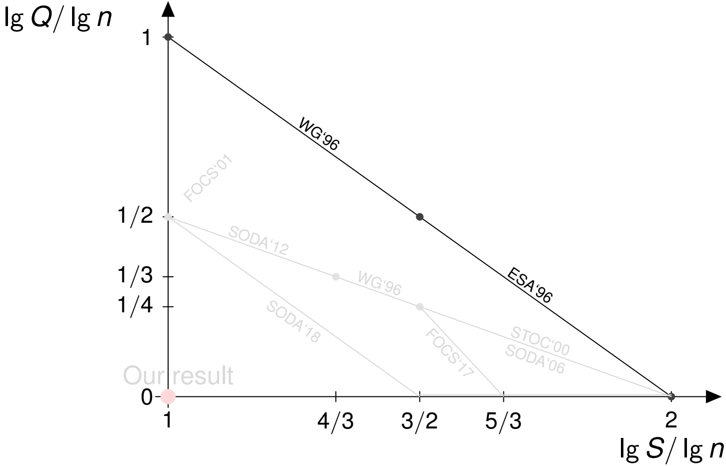
Previous work

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



Previous work

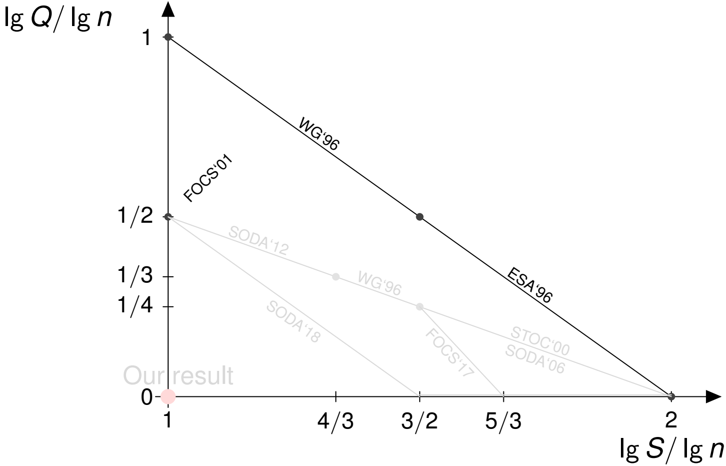
The tradeoff between the query-time Q and the size S of the structure:



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

Previous work

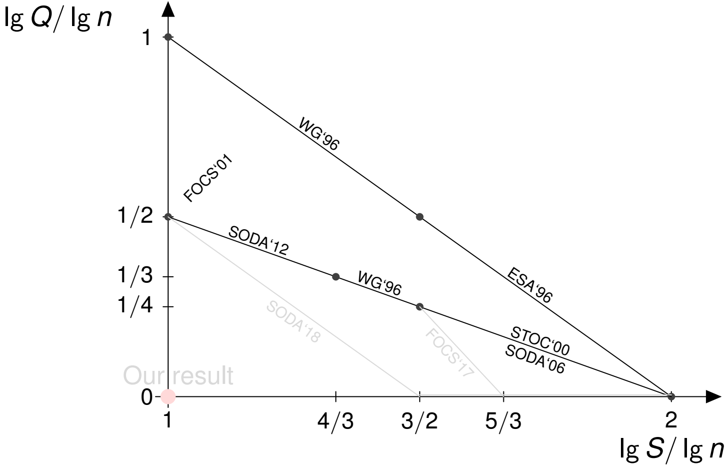
The tradeoff between the query-time Q and the size S of the structure:



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

Previous work

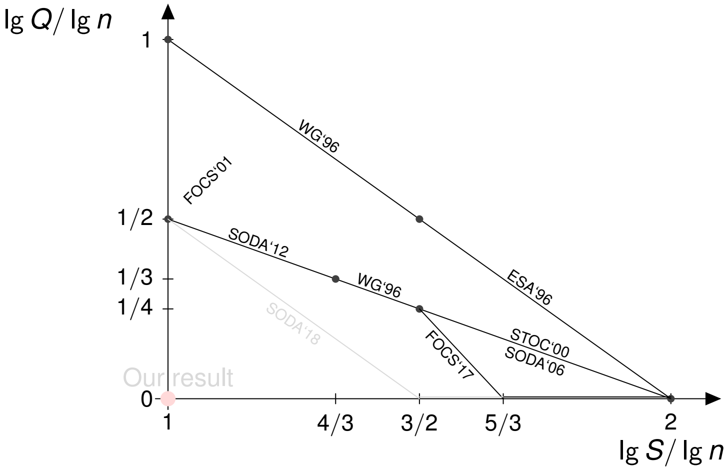
The tradeoff between the query-time Q and the size S of the structure:



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

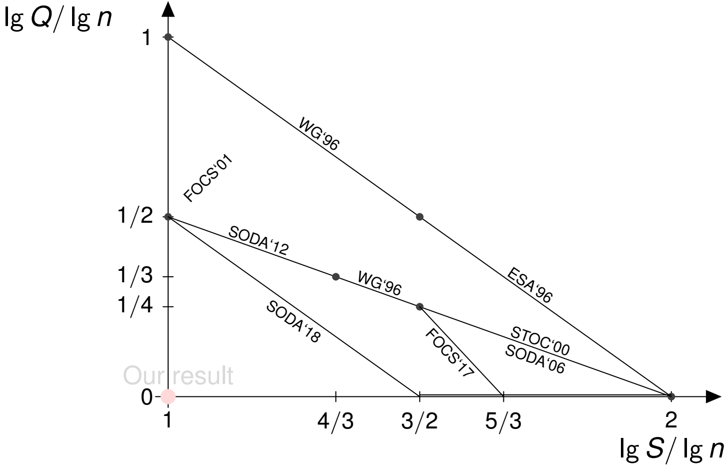
The tradeoff between the query-time Q and the size S of the structure:



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

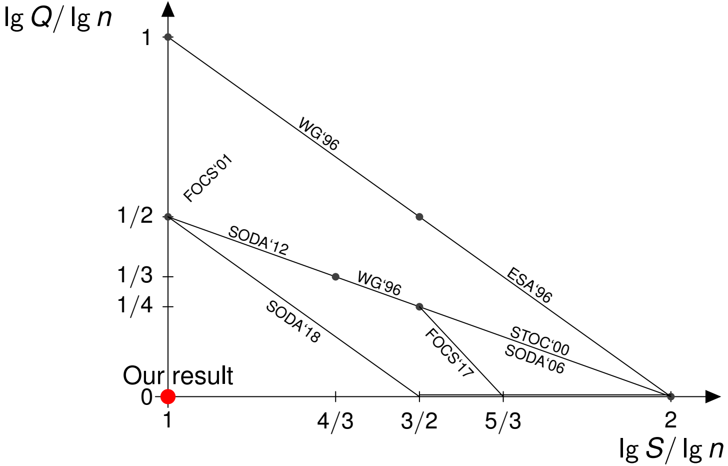
The tradeoff between the query-time Q and the size S of the structure:



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

Previous work

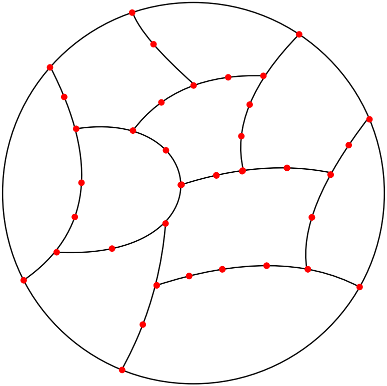
The tradeoff between the query-time Q and the size S of the structure:



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

r-divisions

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

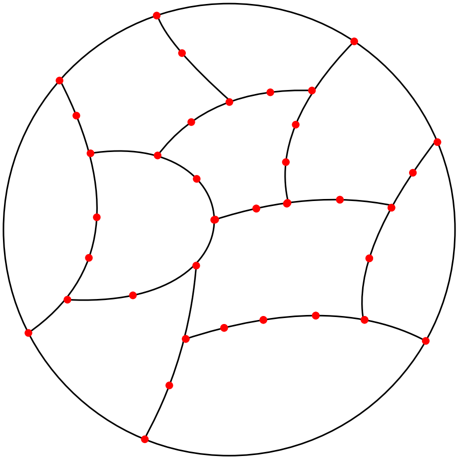


- $O(n/r)$ pieces;
- each piece has $O(r)$ vertices;
- each piece has $O(\sqrt{r})$ boundary vertices (vertices incident to edges in other pieces).

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

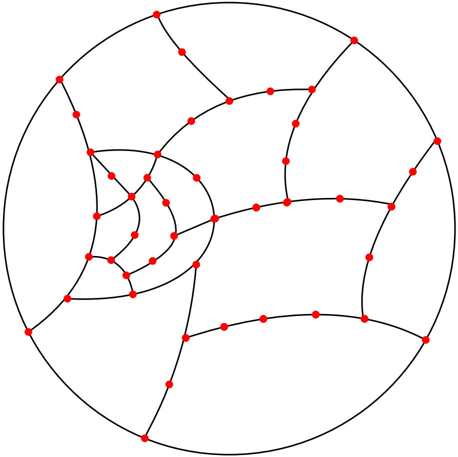
Recursive r-divisions

For $r_1 < r_2 < \cdots < r_m \in [1, n]$, we can efficiently compute r_i-divisions, such that each r_i-division respects the r_{i+1}-division.



Recursive r-divisions

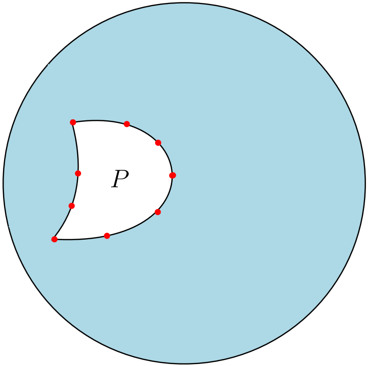
For $r_1 < r_2 < \cdots < r_m \in [1, n]$, we can efficiently compute r_i-divisions, such that each r_i-division respects the r_{i+1}-division.



Multiple Source Shortest Paths (MSSP)

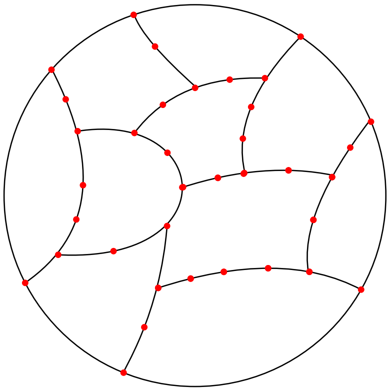
Klein [SODA'05]

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.



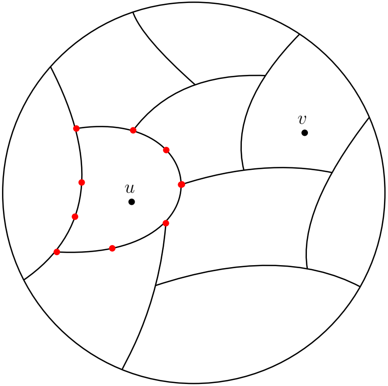
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ 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 $\tilde{O}(n^{1/3})$

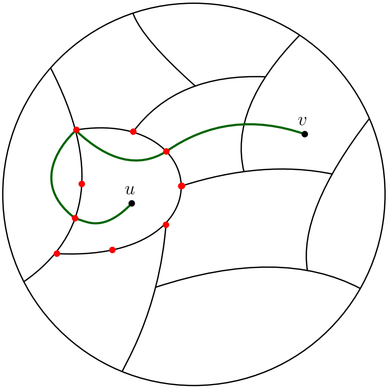
- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



Interesting case.

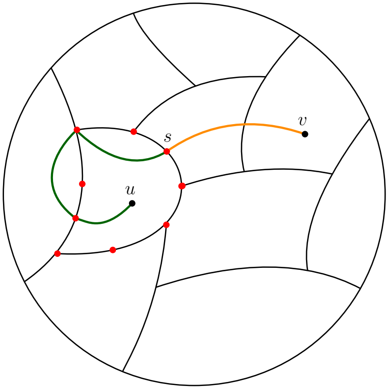
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ 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 $\tilde{O}(n^{1/3})$

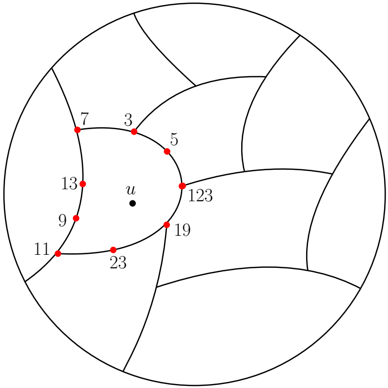
- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$. Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P. Space $\tilde{O}(n/r \cdot n)$.



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

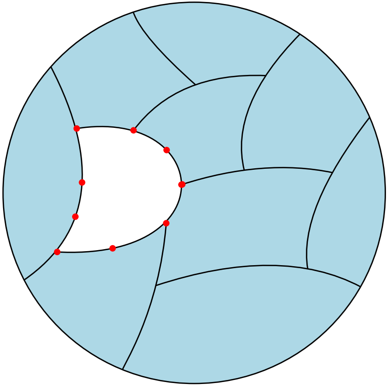
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ 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 $\tilde{O}(n/r \cdot n)$.



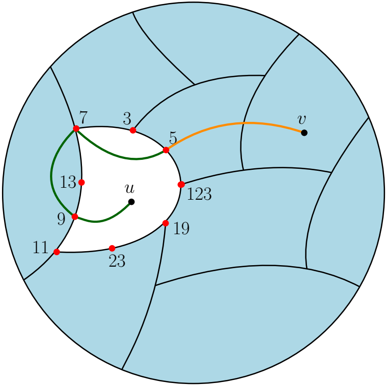
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ 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 $\tilde{O}(n^{1/3})$

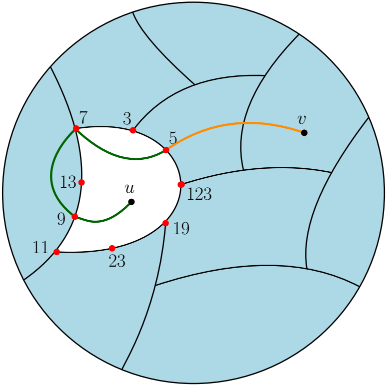
- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$.
  Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P.
  Space $\tilde{O}(n/r \cdot n)$.



At query, find node $p \in \partial P$, minimizing $d_G(u, p) + d_{G\setminus(P\setminus\partial P)}(p, v)$.
This is called *point location*.

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

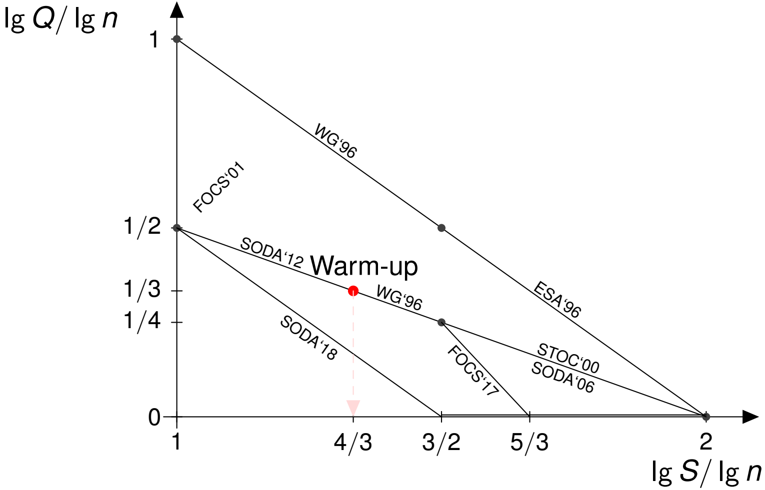
- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$.
  Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P.
  Space $\tilde{O}(n/r \cdot n)$.



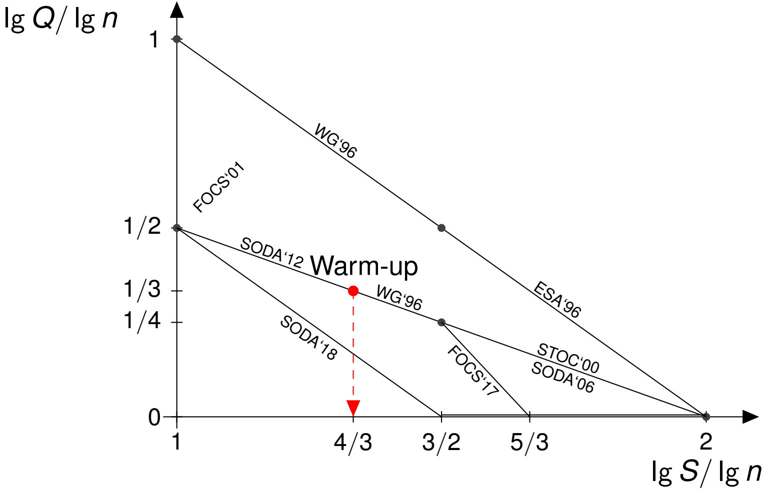
At query, find node $p \in \partial P$, minimizing $d_G(u, p) + d_{G \setminus (P \setminus \partial P)}(p, v)$.
This is called *point location*.

Perform point location by trying all $O(\sqrt{r})$ boundary nodes.

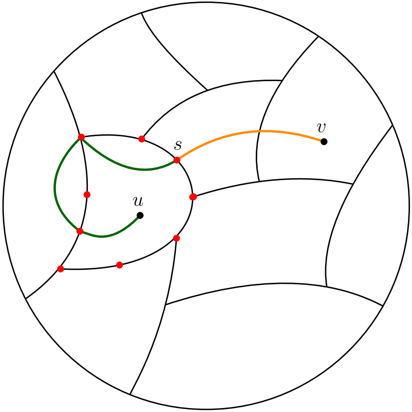
Warm-up: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(n^{1/3})$



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



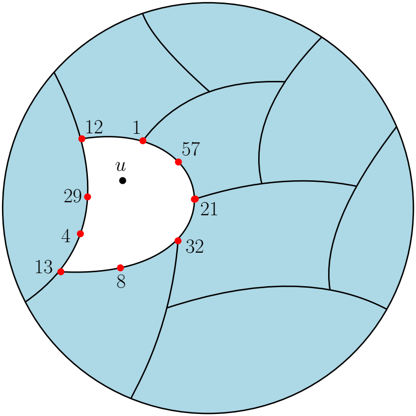
First goal



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.

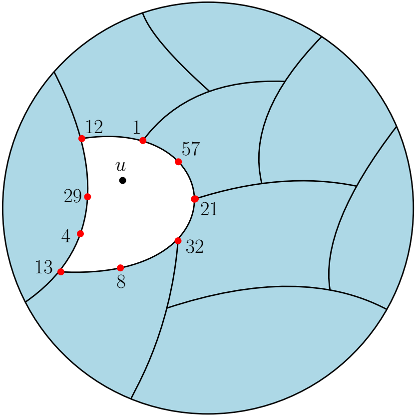
Point location

Each node u defines a set of additive weights $d_G(u, p)$ for $p \in \partial P$.



Point location

Each node u defines a set of additive weights $d_G(u, p)$ for $p \in \partial P$.

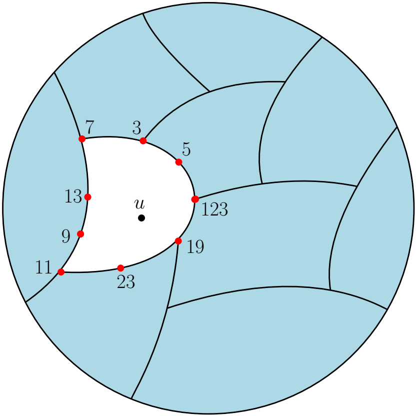


Gawrychowski et. al. [SODA'18]

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.

Point location

Each node u defines a set of additive weights $d_G(u,p)$ for $p \in \partial P$.

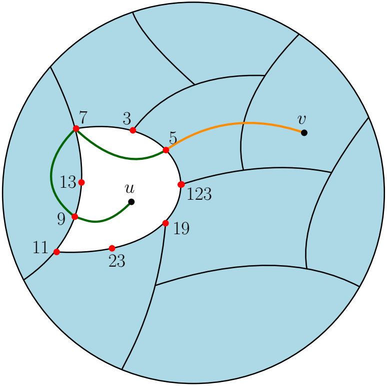


Gawrychowski et. al. [SODA'18]

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 $\tilde{O}(1)$

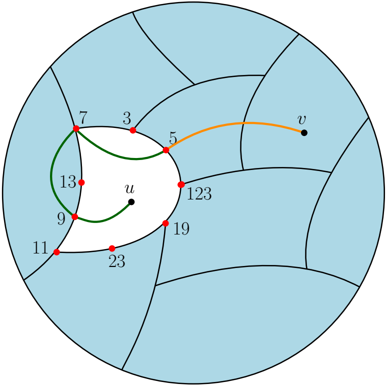
- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$.
  Space $O(n \cdot \sqrt{r})$.
- For each piece P, store an MSSP data structure for the outside of P with sources ∂P.
  Space $\tilde{O}(n/r \cdot n)$.



At query, perform point location by trying all possible $O(\sqrt{r})$ candidate boundary nodes.

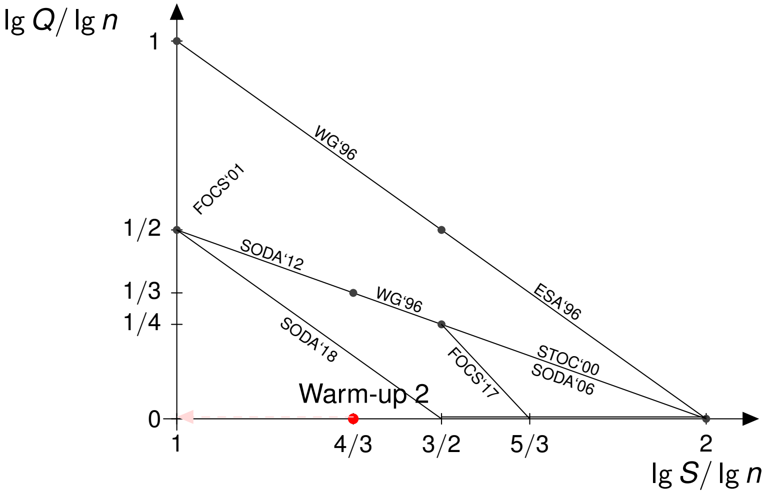
Warm-up 2: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(1)$

- Compute an r-division.
- For each piece P, for each node $u \in P$, store additive weights $d_G(u, p)$ 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 $\tilde{O}(n/r \cdot n)$.

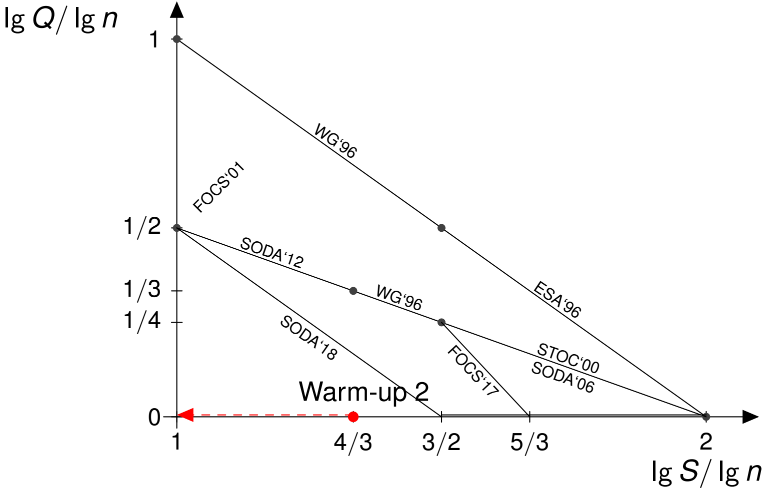


At query, perform point location in $\tilde{O}(1)$ time!

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



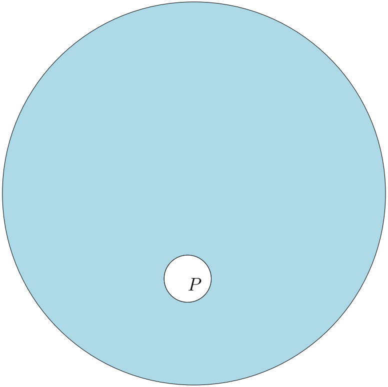
Warm-up 2: Space $\tilde{O}(n^{4/3})$, Query-time $\tilde{O}(1)$



Second goal

Shrink pieces.

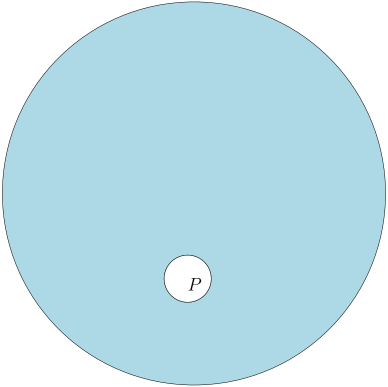
- Compute an n^ϵ-division.
- For each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$. Preprocess these for point location.
- For each piece P, store the required information to support:
  - distance queries from ∂P to nodes outside P;
  - point location.



Second goal

Shrink pieces.

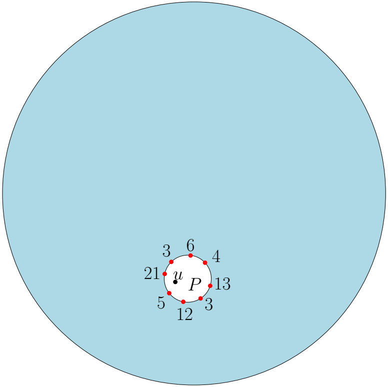
- Compute an n^{ϵ}-division.
- For each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$. Preprocess these for point location.
- For each piece P, store the required information to support:
  - distance queries from ∂P to nodes outside P;
  - point location.



Second goal

Shrink pieces.

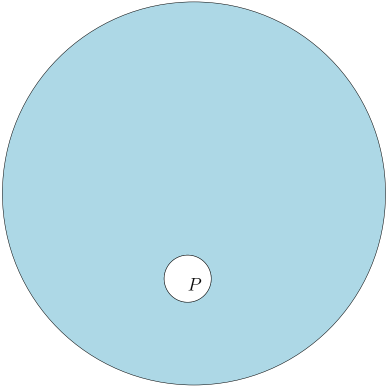
- Compute an n^{ϵ}-division.
- For each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$. Preprocess these for point location.
- For each piece P, store the required information to support:
  - distance queries from ∂P to nodes outside P;
  - point location.



Second goal

Shrink pieces.

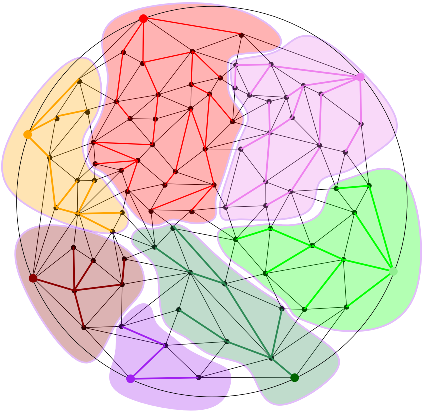
- Compute an n^ϵ-division.
- For each node $u \in P$, store additive weights $d_G(u, p)$ for $p \in \partial P$. Preprocess these for point location.
- For each piece P, store the required information to support:
  - distance queries from ∂P to nodes outside P;
  - point location.



Additively weighted Voronoi diagrams

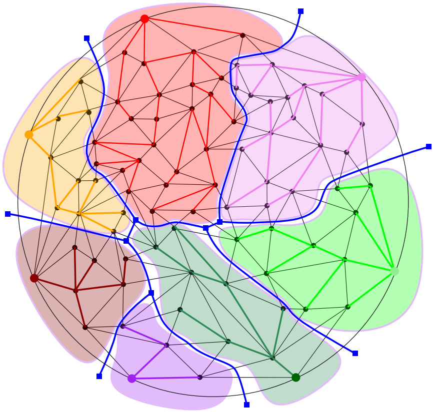
Internals of point location.

Additively weighted Voronoi diagrams



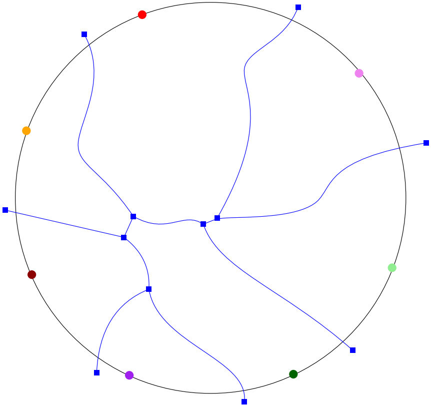
The Voronoi cell of each site consists of all nodes closer to it with respect to the additive distances.

Additively weighted Voronoi diagrams



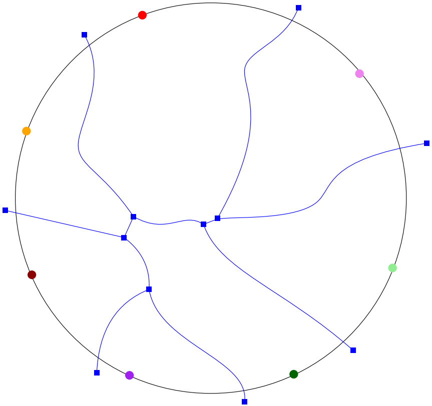
Because all sites are adjacent to one face, the diagram can be described by a tree on $O(|\partial P|) = O(\sqrt{r})$ nodes (independent of n!).

Point location



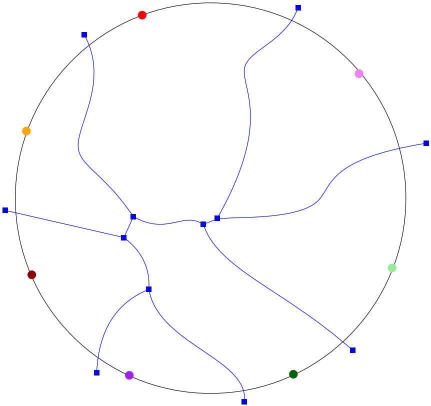
Point location

Point locating v essentially reduces to $O(\log r)$ distance queries from the sites to v.



Point location

Point locating v essentially reduces to $O(\log r)$ distance queries from the sites to v.

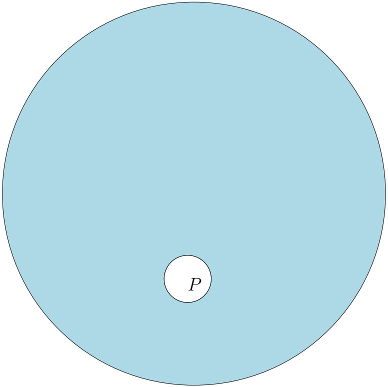


Idea

Handle such distance queries recursively

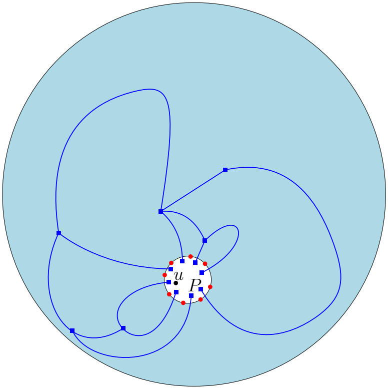
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

- Compute a recursive r-division for $r_i = n^{i \cdot \epsilon}$.
- For each piece P of the n^ϵ-division, 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 $\tilde{O}(n \cdot \sqrt{r_1})$.
- For each piece P, store the required information to answer distance queries from ∂P to nodes outside P.



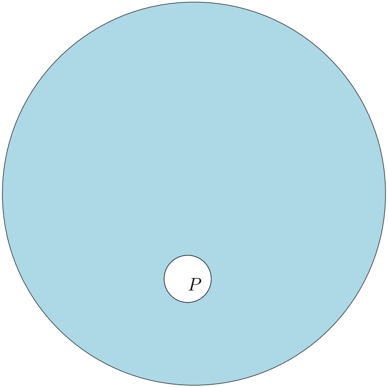
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

- Compute a recursive r-division for $r_i = n^{i\cdot\epsilon}$.
- For each piece P of the n^ϵ-division, 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 $\tilde{O}(n \cdot \sqrt{r_1})$.
- For each piece P, store the required information to answer distance queries from ∂P to nodes outside P.



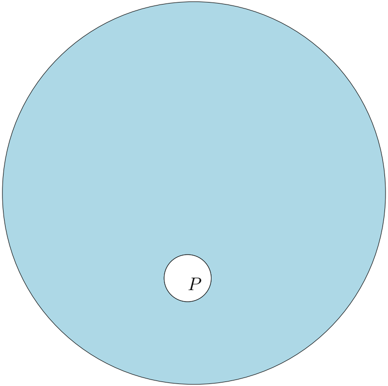
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

- Compute a recursive r-division for $r_i = n^{i \cdot \epsilon}$.
- For each piece P of the n^ϵ-division, 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 $\tilde{O}(n \cdot \sqrt{r_1})$.
- 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 $\tilde{O}(1)$

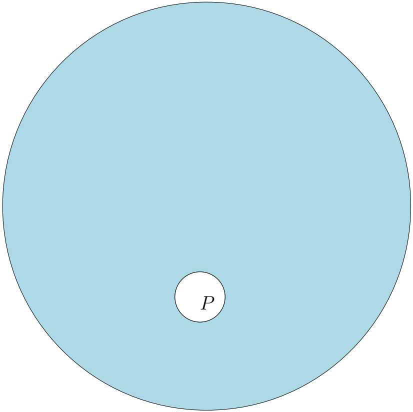
- Compute a recursive r-division for $r_i = n^{i \cdot \epsilon}$.
- For each piece P of the n^{ϵ}-division, for each node $u \in P$, store a Voronoi diagram for the outside of P with sites ∂P and additive weight $d_G(u, p)$ for $p \in \partial P$. Space $\tilde{O}(n \cdot \sqrt{r_1})$.
- For each piece P, store the required information to answer distance queries from ∂P to nodes outside P.



At query, perform point location.

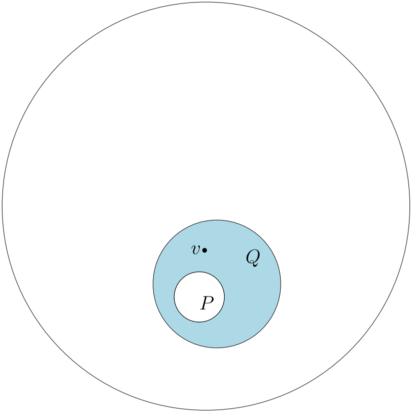
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

We can not afford to store an $\Omega(n)$-sized MSSP for each of the $n^{1-\epsilon}$ pieces.

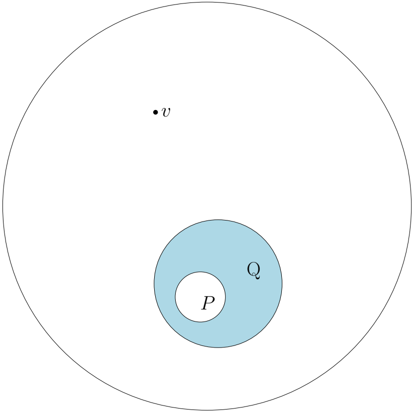


Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

Store an MSSP for piece Q of the $n^{2\cdot\epsilon}$-division that contains P. This handles the case $v \in Q$.
Space: $\tilde{O}(n^{1-\epsilon} \cdot n^{2\epsilon}) = \tilde{O}(n^{1+\epsilon})$.

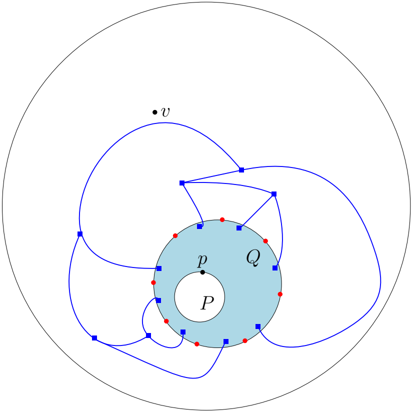


Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$



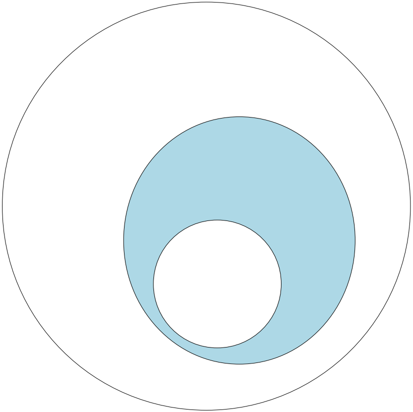
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

Case $v \notin Q$: each $p \in \partial P$ stores a Voronoi diagram for the outside of Q with sites ∂Q.
Space: $\tilde{O}(n^{1-\epsilon} \cdot n^{\epsilon/2} \cdot n^{2\epsilon/2}) == \tilde{O}(n^{1+\epsilon/2})$.



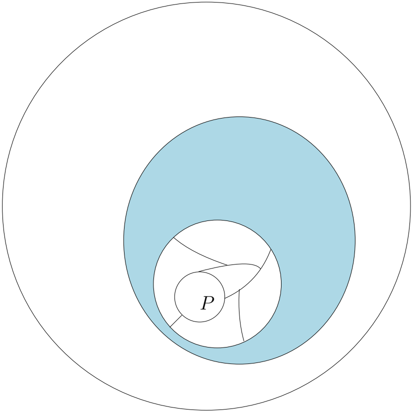
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

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 $\tilde{O}(\frac{1}{\epsilon}n^{1+\epsilon})$.



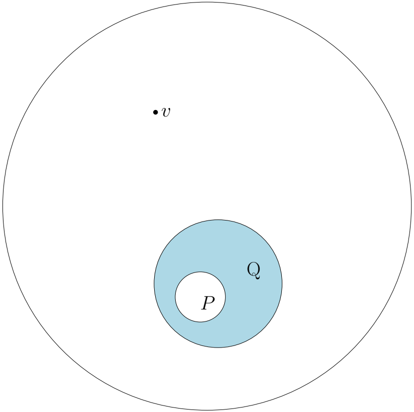
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

Smaller pieces share the MSSP data structures at higher levels.



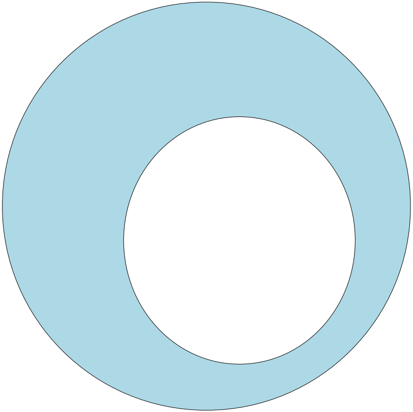
Main result: Space $\tilde{O}(n^{1+\epsilon})$, Query-time $\tilde{O}(1)$

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 answer the point location query in $O(\log^2 n)$ time.
Query time: $O(\log^{1/\epsilon} n)$.



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:

- details of point location;
- ∂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:

- details of point location;
- ∂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 $\langle S, Q \rangle$:

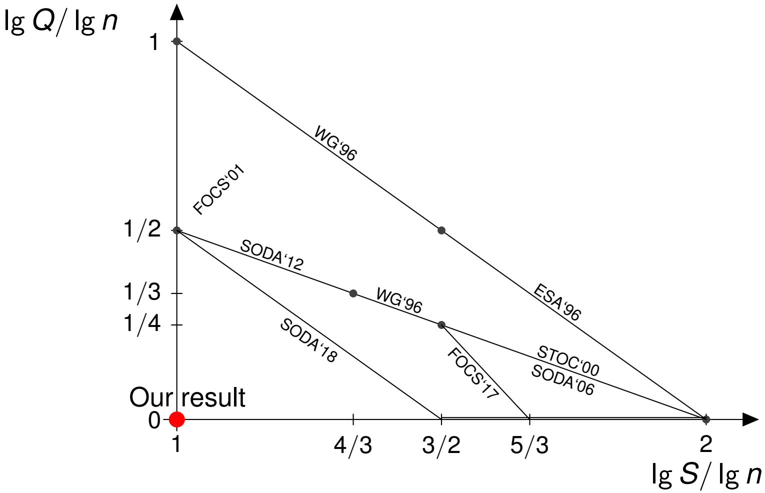
1. $\langle \tilde{O}(n^{1+\epsilon}), O(\log^{1/\epsilon} n) \rangle$, for any constant $1/2 \geq \epsilon > 0$;
2. $\langle O(n \log^{2+1/\epsilon} n), \tilde{O}(n^{\epsilon}) \rangle$, for any constant $\epsilon > 0$;
3. $\langle n^{1+o(1)}, n^{o(1)} \rangle$.

Some of the issues I shoved under the rug:

- details of point location;
- ∂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 oracles?
  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
Our result
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

Questions?