An Almost Optimal Edit Distance Oracle

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

[1]The Interdisciplinary Center Herzliya, Israel

[2]University of Wrocław, Poland

[3]University of Haifa, Israel

ICALP 2021

Problem Definition

Edit Distance

Input: Two strings of total length n.

Problem Definition

Edit Distance

Input: Two strings of total length n.
Goal: Compute the minimum number of letter insertions, deletions, and substitutions required to transform one string into the other.

Problem Definition

Edit Distance

Input: Two strings of total length n.
Goal: Compute the minimum number of letter insertions, deletions, and substitutions required to transform one string into the other.

$$
\begin{array}{rccccccc}
X = & \mathtt{a} & \mathtt{a} & \mathtt{c} & - & \mathtt{b} & \mathtt{c} & \mathtt{d} \\
 & \cdot & | & | & & | & | & \\
Y = & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{d} & \mathtt{b} & \mathtt{c} & -
\end{array}
$$

Problem Definition

Edit Distance

Input: Two strings of total length n.
Goal: Compute the minimum number of letter insertions, deletions, and substitutions required to transform one string into the other.

$$
\begin{array}{cccccccc}
X = & \texttt{a} & \texttt{a} & \texttt{c} & - & \texttt{b} & \texttt{c} & \texttt{d} \\
 & \cdot & | & | & & | & | & \\
Y = & \texttt{b} & \texttt{a} & \texttt{c} & \texttt{d} & \texttt{b} & \texttt{c} & -
\end{array}
$$

Problem Definition

Edit Distance

Input: Two strings of total length n.
Goal: Compute the minimum number of letter insertions, deletions, and substitutions required to transform one string into the other.

$$\begin{array}{rccccccc} X = & \mathtt{a} & \mathtt{a} & \mathtt{c} & - & \mathtt{b} & \mathtt{c} & \mathtt{d} \\ & \cdot & | & | & & | & | & \\ Y = & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{d} & \mathtt{b} & \mathtt{c} & - \end{array}$$

Problem Definition

Edit Distance

Input: Two strings of total length n.
Goal: Compute the minimum number of letter insertions, deletions, and substitutions required to transform one string into the other.

$$
\begin{array}{rccccccc}
X = & \mathtt{a} & \mathtt{a} & \mathtt{c} & - & \mathtt{b} & \mathtt{c} & \color{red}{\mathtt{d}} \\
 & \cdot & | & | & & | & | & \\
Y = & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{d} & \mathtt{b} & \mathtt{c} & \color{red}{-}
\end{array}
$$

Problem Definition

Edit Distance

Input: Two strings of total length n.
Goal: Compute the minimum number of letter insertions, deletions, and substitutions required to transform one string into the other.

$$\begin{array}{ccccccccc} X = & \mathtt{a} & \mathtt{a} & \mathtt{c} & - & \mathtt{b} & \mathtt{c} & \mathtt{d} \\ & \cdot & | & | & & | & | & \\ Y = & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{d} & \mathtt{b} & \mathtt{c} & - \end{array}$$

The edit distance of X and Y is 3.

Classic Dynamic Programming Solution

There is a textbook $\mathcal{O}(n^2)$-time dynamic programming algorithm.

[Vintsyuk, Cybernetics 1968]

[Needleman & Wunsch, Journal of Molecular Biology 1970]

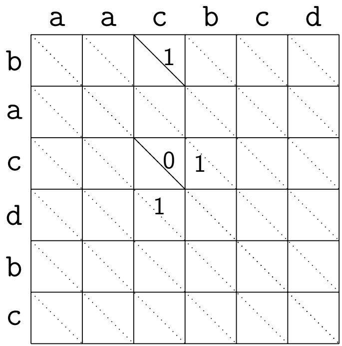
[Wagner & Fischer, Journal of the ACM 1974]

There is a textbook $\mathcal{O}(n^2)$-time dynamic programming algorithm.

[Vintsyuk, Cybernetics 1968]

[Needleman & Wunsch, Journal of Molecular Biology 1970]

[Wagner & Fischer, Journal of the ACM 1974]



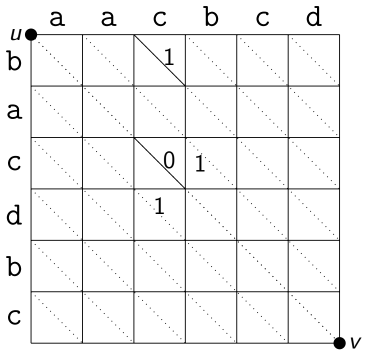
Classic Dynamic Programming Solution

There is a textbook $\mathcal{O}(n^2)$-time dynamic programming algorithm.

[Vintsyuk, Cybernetics 1968]

[Needleman & Wunsch, Journal of Molecular Biology 1970]

[Wagner & Fischer, Journal of the ACM 1974]



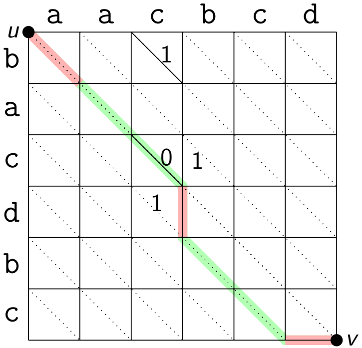
Classic Dynamic Programming Solution

There is a textbook $\mathcal{O}(n^2)$-time dynamic programming algorithm.

[Vintsyuk, Cybernetics 1968]

[Needleman & Wunsch, Journal of Molecular Biology 1970]

[Wagner & Fischer, Journal of the ACM 1974]



Related Work

Several works improved the complexity by polylogarithmic factors.

[Masek & Paterson; Journal of Computer and System Sciences 1980]

[Crochemore, Landau, Ziv-Ukelson, SIAM Journal on Computing 2003]

[Grabowski, Discrete Applied Mathematics 2016]

Related Work

Several works improved the complexity by polylogarithmic factors.

[Masek & Paterson; Journal of Computer and System Sciences 1980]

[Crochemore, Landau, Ziv-Ukelson, SIAM Journal on Computing 2003]

[Grabowski, Discrete Applied Mathematics 2016]

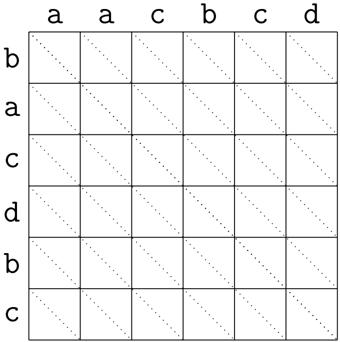
A strongly subquadratic-time algorithm would refute the Strong Exponential Time Hypothesis (SETH).

[Backurs & Indyk, SIAM Journal on Computing 2018]

[Bringmann & Künnemann, FOCS 2015]

Edit Distance Oracle

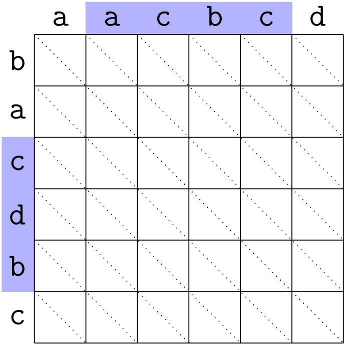
Input: Two strings X and Y of total length n.



The Oracle Version

Edit Distance Oracle

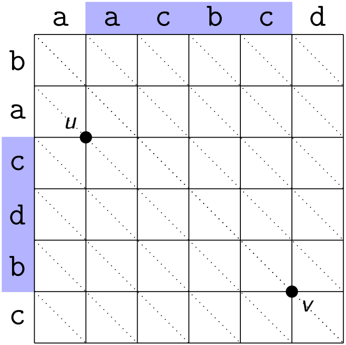
Input: Two strings X and Y of total length n.
Query: Compute the edit distance of $X[i\,..\,j]$ and $Y[a\,..\,b]$.



The Oracle Version

Edit Distance Oracle

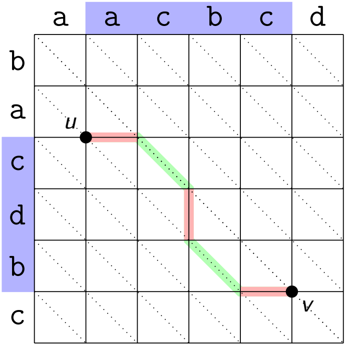
Input: Two strings X and Y of total length n.
Query: Compute the edit distance of $X[i\,..\,j]$ and $Y[a\,..\,b]$.



The Oracle Version

Edit Distance Oracle

Input: Two strings X and Y of total length n.
Query: Compute the edit distance of $X[i\,..\,j]$ and $Y[a\,..\,b]$.



Results

Let $N = n^2$.

Results

Let $N = n^2$.

Near-optimal data structures for restricted variants using efficient $(\min, +)$-multiplication of simple unit-Monge matrices. [Tiskin, 2007]

Results

Let $N = n^2$.

Solution	Preprocessing	Space	Query
[Sakai, TCS 2019]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(\sqrt{N})$

Results

Let $N = n^2$.

Solution	Preprocessing	Space	Query
[Sakai, TCS 2019]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(\sqrt{N})$
[Long & Pettie, SODA 2021]	$N^{3/2+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$

An exact distance oracle for arbitrary planar graphs.

Results

Let $N = n^2$.

Solution	Preprocessing	Space	Query
[Sakai, TCS 2019]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(\sqrt{N})$
[Long & Pettie, SODA 2021]	$N^{3/2+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
Our results:	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$

We specialize recent techniques for planar distance oracles and exploit the structure of the alignment grid.

[Gawrychowski, Mozes, Weimann, Wulff-Nilsen, SODA 2018]

[C., Gawrychowski, Mozes, Weimann, STOC 2019]

[Long & Pettie, SODA 2021]

Let $N = n^2$.

Solution	Preprocessing	Space	Query
[Sakai, TCS 2019]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(\sqrt{N})$
[Long & Pettie, SODA 2021]	$N^{3/2+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
Our results:	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$

Our data structure is simpler and easier to understand, but includes many of the high-level ideas for planar distance oracles.

Results

Let $N = n^2$.

Solution	Preprocessing	Space	Query
[Sakai, TCS 2019]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(\sqrt{N})$
[Long & Pettie, SODA 2021]	$N^{3/2+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
Our results:	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$

Conditional lower bound for edit distance $\Rightarrow$ preprocessing time + query time cannot be strongly sublinear in N unless SETH fails.

Results

Let $N = n^2$.

Solution	Preprocessing	Space	Query
[Sakai, TCS 2019]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(\sqrt{N})$
[Long & Pettie, SODA 2021]	$N^{3/2+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
Our results:	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$

Any data structure with query time t must use $N/(t^2 \cdot \log^{\mathcal{O}(1)} N)$ space, assuming the Strong Set Disjointness Conjecture.

Results

Let $N = n^2$.

Solution	Preprocessing	Space	Query
[Sakai, TCS 2019]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(\sqrt{N})$
[Long & Pettie, SODA 2021]	$N^{3/2+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
Our results:	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
$t \in [\sqrt{N}, N]$	$\tilde{\mathcal{O}}(N)$	$\tilde{\mathcal{O}}(N/\sqrt{t})$	$\tilde{\mathcal{O}}(t)$

Any data structure with query time t must use $N/(t^2 \cdot \log^{\mathcal{O}(1)} N)$ space, assuming the Strong Set Disjointness Conjecture.

Results

Let $N = n^2$.

Solution	Preprocessing	Space	Query
[Sakai, TCS 2019]	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(\sqrt{N})$
[Long & Pettie, SODA 2021]	$N^{3/2+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
Our results:	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
$t \in [\sqrt{N}, N]$	$\tilde{\mathcal{O}}(N)$	$\tilde{\mathcal{O}}(N/\sqrt{t})$	$\tilde{\mathcal{O}}(t)$

MSSP for Planar Graphs

Multiple Source Shortest Paths (MSSP) [Klein, SODA 2005]

We can construct in nearly-linear time (in the size of the graph) a data structure that can report in logarithmic time the distance between any vertex on the infinite face and any vertex in the graph.

MSSP for Planar Graphs

Multiple Source Shortest Paths (MSSP) [Klein, SODA 2005]

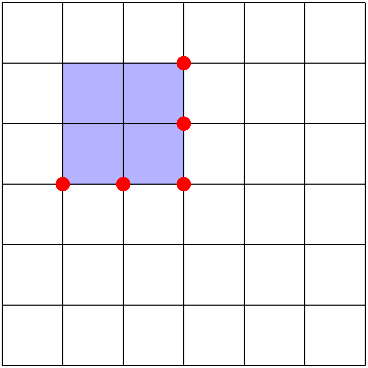
We can construct in nearly-linear time (in the size of the graph) a data structure that can report in logarithmic time the distance between any vertex on the infinite face and any vertex in the graph.

First developed for alignment grids. [Schmidt, SICOMP 1998]

MSSP for Planar Graphs

Multiple Source Shortest Paths (MSSP) [Klein, SODA 2005]

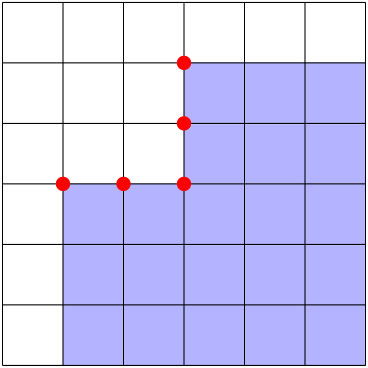
We can construct in nearly-linear time (in the size of the graph) a data structure that can report in logarithmic time the distance between any vertex on the infinite face and any vertex in the graph.



MSSP for Planar Graphs

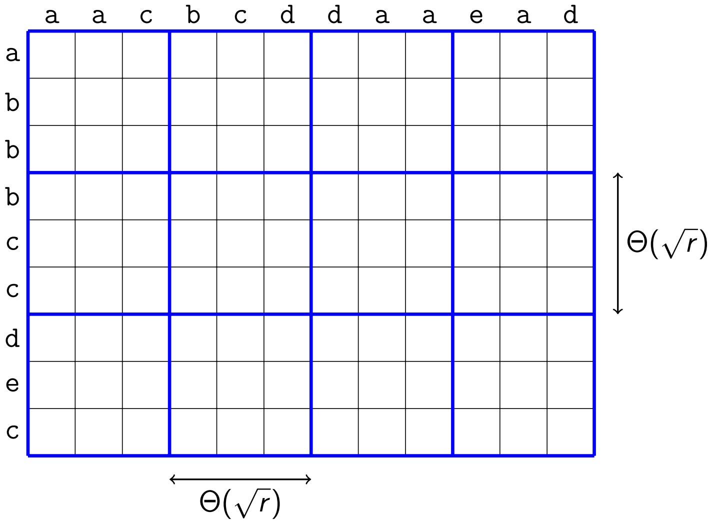
Multiple Source Shortest Paths (MSSP) [Klein, SODA 2005]

We can construct in nearly-linear time (in the size of the graph) a data structure that can report in logarithmic time the distance between any vertex on the infinite face and any vertex in the graph.

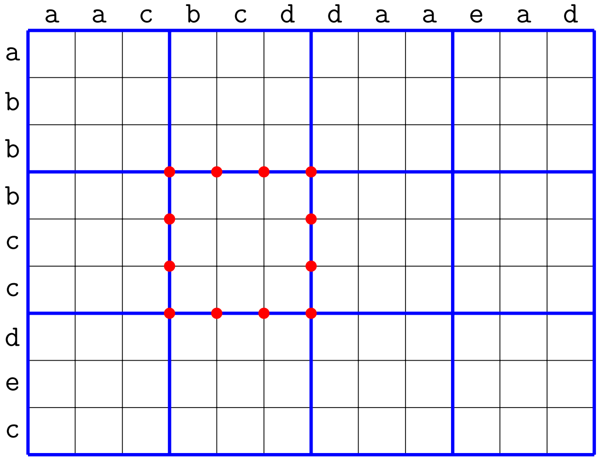


	a	a	c	b	c	d	d	a	a	e	a	d
a												
b												
b												
b												
c												
c												
d												
e												
c												

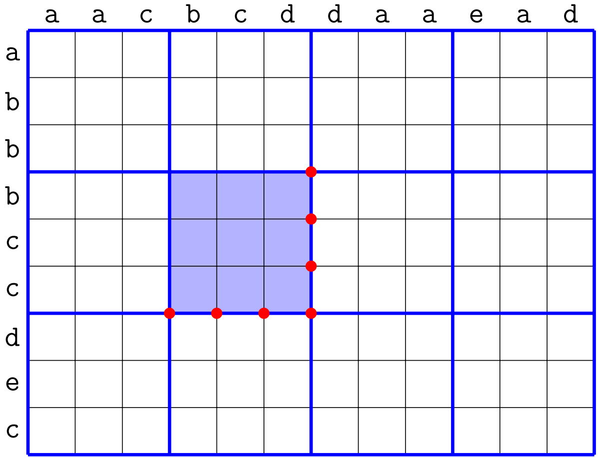
Warm-up I: Prep-time $\tilde{O}(N^2/r)$, Query Time $\tilde{O}(\sqrt{r})$



Warm-up I: Prep-time $\tilde{O}(N^2/r)$, Query Time $\tilde{O}(\sqrt{r})$



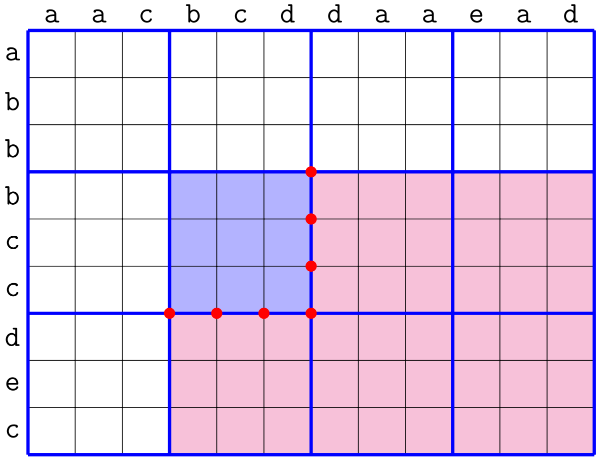
For each piece P, we denote the set of "boundary" vertices by ∂P.
$|P| = \Theta(r)$, $|\partial P| = \Theta(\sqrt{r})$.



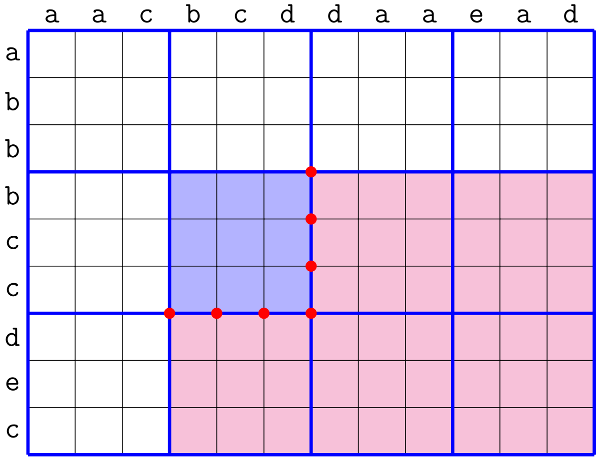
For each piece P, we store an MSSP data structure for P

.

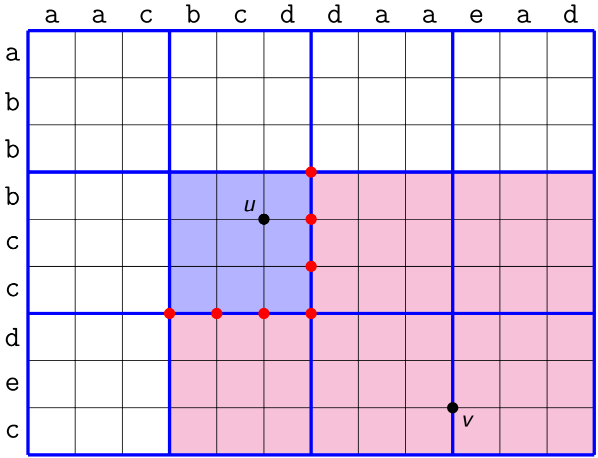
Warm-up I: Prep-time $\tilde{O}(N^2/r)$, Query Time $\tilde{O}(\sqrt{r})$



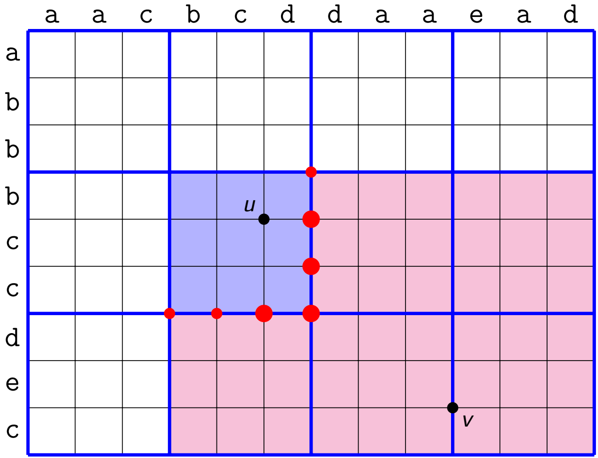
For each piece P, we store an MSSP data structure for P and one for P^{out}.



For each piece P, we store an MSSP data structure for P and one for P^{out}. Prep-time: $N/r \cdot \tilde{\mathcal{O}}(r + N) = \tilde{\mathcal{O}}(N^2/r)$.

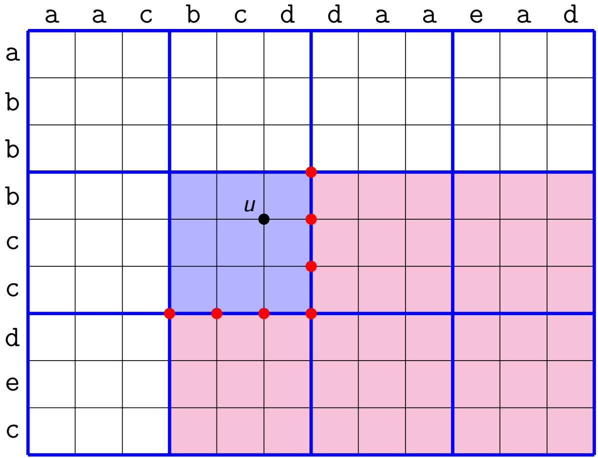


We can answer a query in $\mathcal{O}(\sqrt{r} \cdot \log n)$ time by trying all the boundary vertices of a piece that contains u, using MSSP.

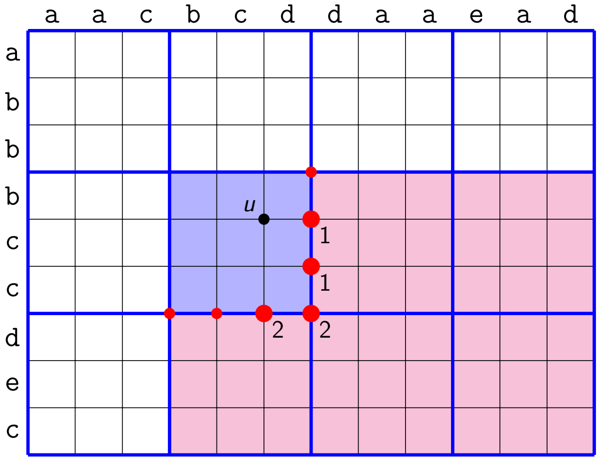


We can answer a query in $\mathcal{O}(\sqrt{r} \cdot \log n)$ time by trying all the boundary vertices of a piece that contains u, using MSSP.

Warm-up I: Prep-time $\tilde{O}(N^2/r)$, Query Time $\tilde{O}(\sqrt{r})$

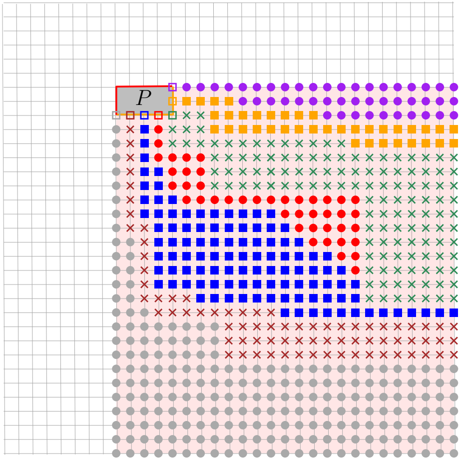


Next: We will store more information for u to speed up the query.



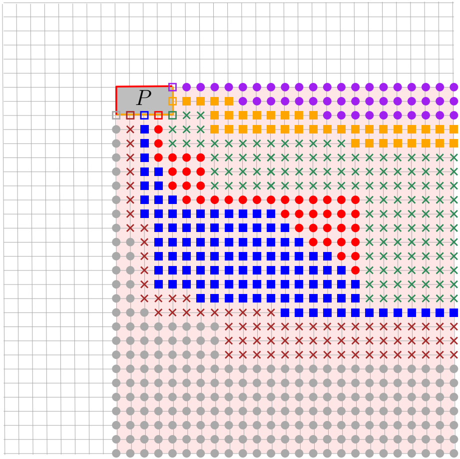
Next: We will store more information for u to speed up the query. Its distances to each of the relevant boundary vertices and...

Voronoi Diagrams on the Alignment Grid

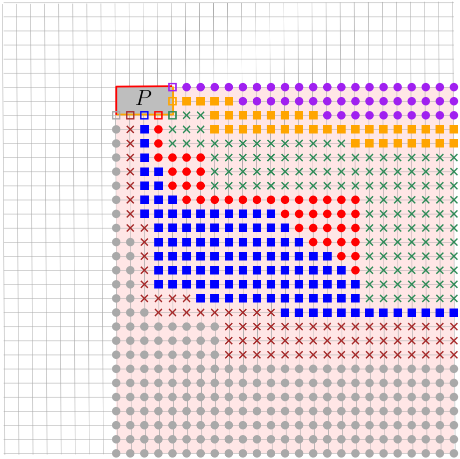


We are given weights for a set S of contiguous vertices of ∂P, called *sites*.

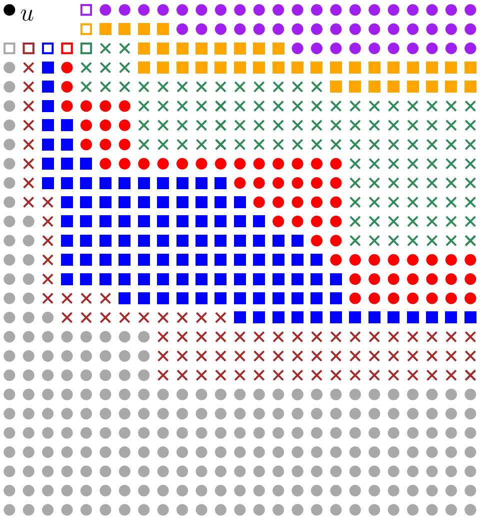
Voronoi Diagrams on the Alignment Grid



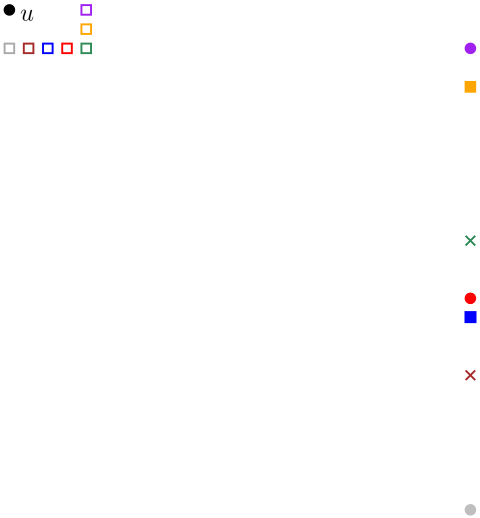
The Voronoi cell of each site consists of all vertices in P^{out} that are closer to it with respect to the additive distances.



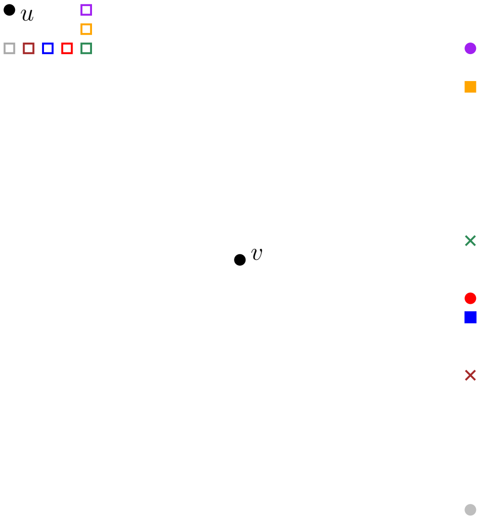
The Voronoi cell of each site s is bounded by a "double-staircase" and has a bottom-right vertex $\ell(s)$.
$\{(s, \ell(s)) : s \in S\}$ is all we store (for now). Space: $\mathcal{O}(N \cdot \sqrt{r})$.


u

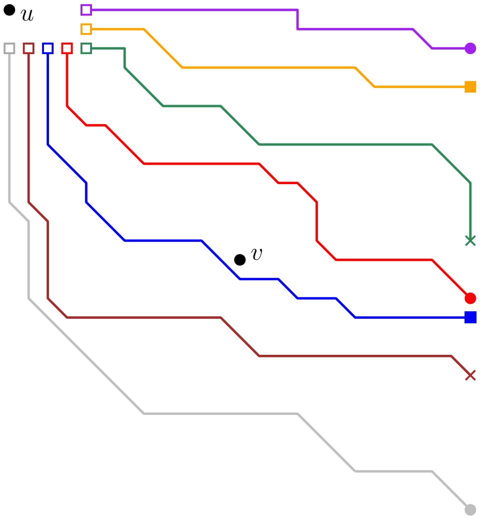
Reducing to 2 Candidate Sites



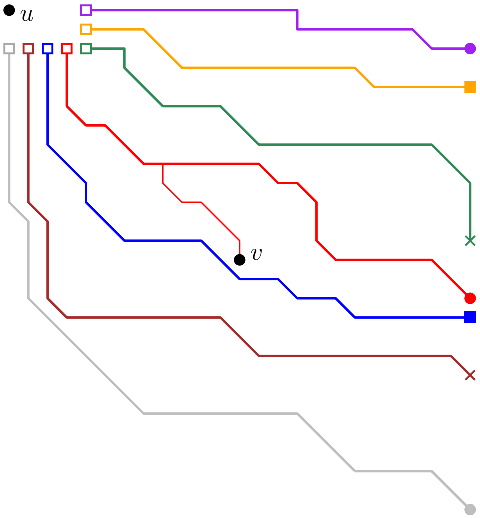
Reducing to 2 Candidate Sites



Reducing to 2 Candidate Sites

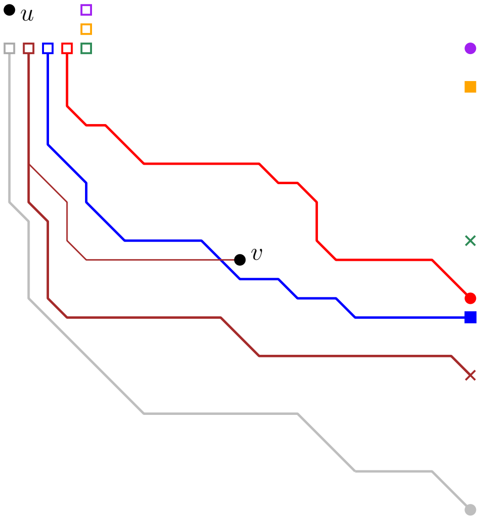


Reducing to 2 Candidate Sites



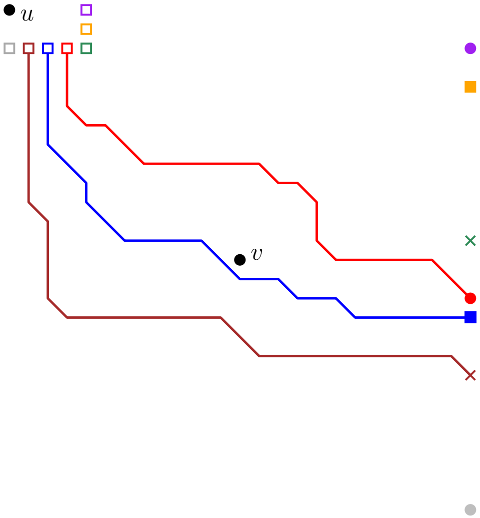
MSSP can answer whether v is left or right of a shortest s-to-$\ell(s)$ path in logarithmic time.

Reducing to 2 Candidate Sites



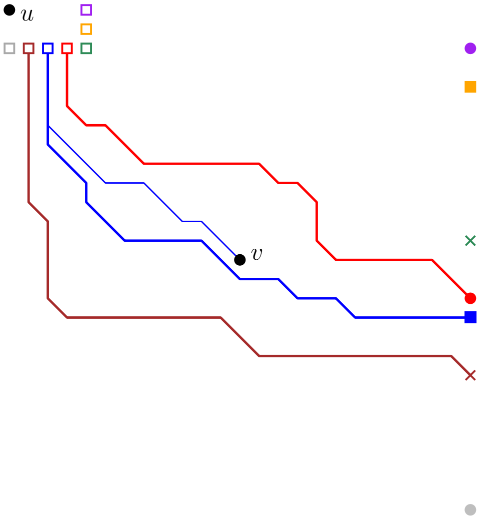
We can thus perform binary search.

Reducing to 2 Candidate Sites



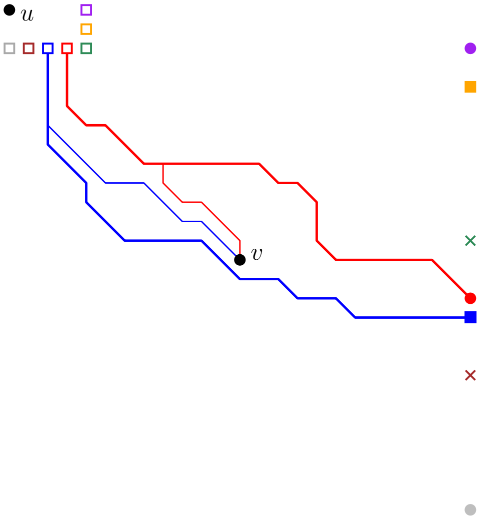
We can thus perform binary search.

Reducing to 2 Candidate Sites



We can thus perform binary search.

Reducing to 2 Candidate Sites



We end up with 2 candidate sites in $\mathcal{O}(\log^2 n)$ time.

Component
Internal MSSPs
External MSSPs
Voronoi diagrams

Warm-up II: Prep-time $\tilde{\mathcal{O}}(N^{4/3})$, Query Time $\mathcal{O}(\log^2 n)$

Component
Internal MSSPs
External MSSPs
Voronoi diagrams

Query time: $\mathcal{O}(\log^2 n)$. We first compute two candidates, and then compute the distance to each of them using the MSSP structures.

Warm-up II: Prep-time $\tilde{\mathcal{O}}(N^{4/3})$, Query Time $\mathcal{O}(\log^2 n)$

Component	Prep-time	Space
Internal MSSPs		
External MSSPs		
Voronoi diagrams		

Query time: $\mathcal{O}(\log^2 n)$. We first compute two candidates, and then compute the distance to each of them using the MSSP structures.

Warm-up II: Prep-time $\tilde{\mathcal{O}}(N^{4/3})$, Query Time $\mathcal{O}(\log^2 n)$

Component	Prep-time	Space
Internal MSSPs	$N/r \cdot \tilde{\mathcal{O}}(r)$	$N/r \cdot \tilde{\mathcal{O}}(r)$
External MSSPs		
Voronoi diagrams		

Query time: $\mathcal{O}(\log^2 n)$. We first compute two candidates, and then compute the distance to each of them using the MSSP structures.

Warm-up II: Prep-time $\tilde{\mathcal{O}}(N^{4/3})$, Query Time $\mathcal{O}(\log^2 n)$

Component	**Prep-time**	**Space**
Internal MSSPs	$N/r \cdot \tilde{\mathcal{O}}(r)$	$N/r \cdot \tilde{\mathcal{O}}(r)$
External MSSPs	$N/r \cdot \tilde{\mathcal{O}}(N)$	$N/r \cdot \tilde{\mathcal{O}}(N)$
Voronoi diagrams		

Query time: $\mathcal{O}(\log^2 n)$. We first compute two candidates, and then compute the distance to each of them using the MSSP structures.

Warm-up II: Prep-time $\tilde{\mathcal{O}}(N^{4/3})$, Query Time $\mathcal{O}(\log^2 n)$

Component	Prep-time	Space
Internal MSSPs	$N/r \cdot \tilde{\mathcal{O}}(r)$	$N/r \cdot \tilde{\mathcal{O}}(r)$
External MSSPs	$N/r \cdot \tilde{\mathcal{O}}(N)$	$N/r \cdot \tilde{\mathcal{O}}(N)$
Voronoi diagrams		$N \cdot \mathcal{O}(\sqrt{r})$

Query time: $\mathcal{O}(\log^2 n)$. We first compute two candidates, and then compute the distance to each of them using the MSSP structures.

Warm-up II: Prep-time $\tilde{\mathcal{O}}(N^{4/3})$, Query Time $\mathcal{O}(\log^2 n)$

Component	Prep-time	Space
Internal MSSPs	$N/r \cdot \tilde{\mathcal{O}}(r)$	$N/r \cdot \tilde{\mathcal{O}}(r)$
External MSSPs	$N/r \cdot \tilde{\mathcal{O}}(N)$	$N/r \cdot \tilde{\mathcal{O}}(N)$
Voronoi diagrams	??	$N \cdot \mathcal{O}(\sqrt{r})$

Query time: $\mathcal{O}(\log^2 n)$. We first compute two candidates, and then compute the distance to each of them using the MSSP structures.

Warm-up II: Prep-time $\tilde{\mathcal{O}}(N^{4/3})$, Query Time $\mathcal{O}(\log^2 n)$

Component	Prep-time	Space
Internal MSSPs	$N/r \cdot \tilde{\mathcal{O}}(r)$	$N/r \cdot \tilde{\mathcal{O}}(r)$
External MSSPs	$N/r \cdot \tilde{\mathcal{O}}(N)$	$N/r \cdot \tilde{\mathcal{O}}(N)$
Voronoi diagrams	$N \cdot \tilde{\mathcal{O}}(\sqrt{r})$	$N \cdot \mathcal{O}(\sqrt{r})$

Query time: $\mathcal{O}(\log^2 n)$. We first compute two candidates, and then compute the distance to each of them using the MSSP structures.

We next show how to construct each VD in time roughly proportional to the number of sites.

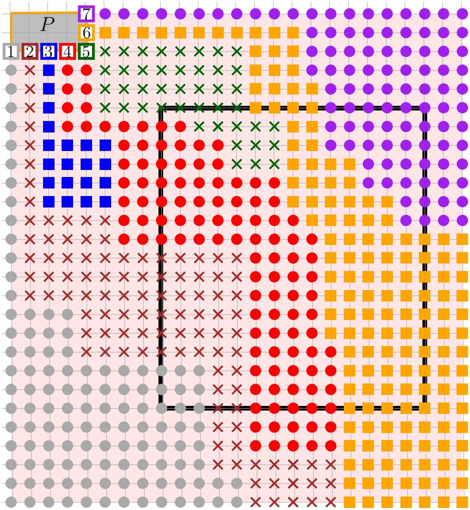
Warm-up II: Prep-time $\tilde{\mathcal{O}}(N^{4/3})$, Query Time $\mathcal{O}(\log^2 n)$

Component	**Prep-time**	**Space**
Internal MSSPs	$N/r \cdot \tilde{\mathcal{O}}(r)$	$N/r \cdot \tilde{\mathcal{O}}(r)$
External MSSPs	$N/r \cdot \tilde{\mathcal{O}}(N)$	$N/r \cdot \tilde{\mathcal{O}}(N)$
Voronoi diagrams	$N \cdot \tilde{\mathcal{O}}(\sqrt{r})$	$N \cdot \mathcal{O}(\sqrt{r})$
Total:	$\tilde{\mathcal{O}}(N^2/r + N \cdot \sqrt{r})$	$\tilde{\mathcal{O}}(N^2/r + N \cdot \sqrt{r})$

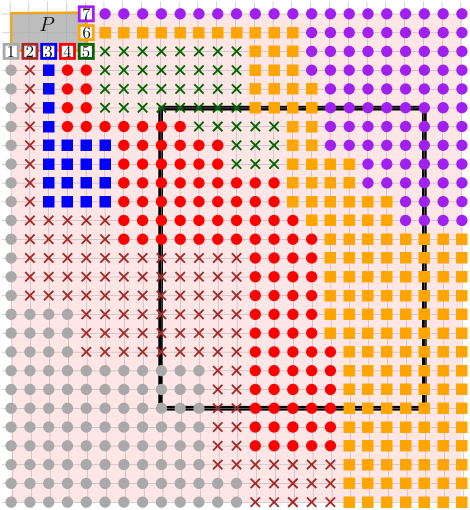
Query time: $\mathcal{O}(\log^2 n)$. We first compute two candidates, and then compute the distance to each of them using the MSSP structures.

We next show how to construct each VD in time roughly proportional to the number of sites.

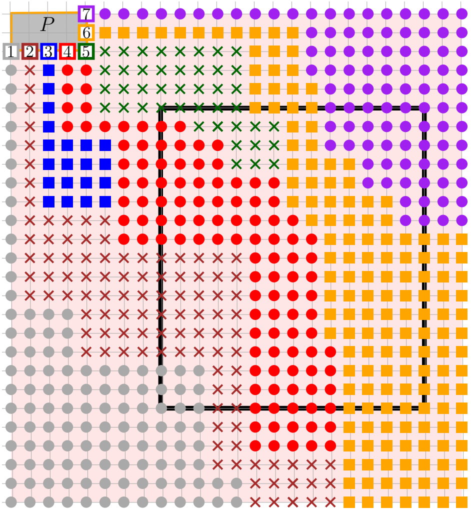
Computing VDs in $\tilde{\mathcal{O}}(|S|)$ site-to-vertex Distance Queries



Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.



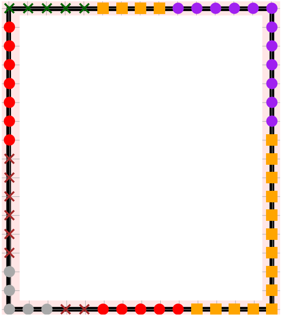
Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.



Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

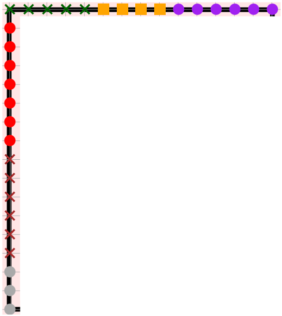
Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.



Computing VDs in $\tilde{\mathcal{O}}(|S|)$ site-to-vertex Distance Queries

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

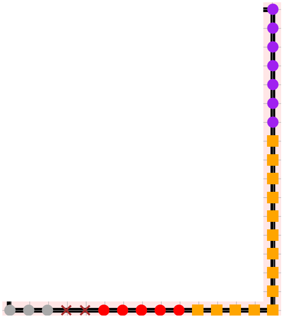
Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.



Top-left: $\{$ ● × ● × ■ ● $\}$

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.

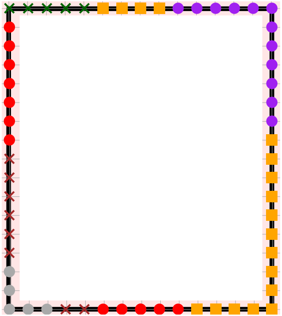


Top-left: $\{$ ● × ● × ■ ● $\}$

Bottom-right: $\{$ ● × ● ■ ● $\}$

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

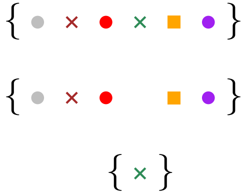
Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.



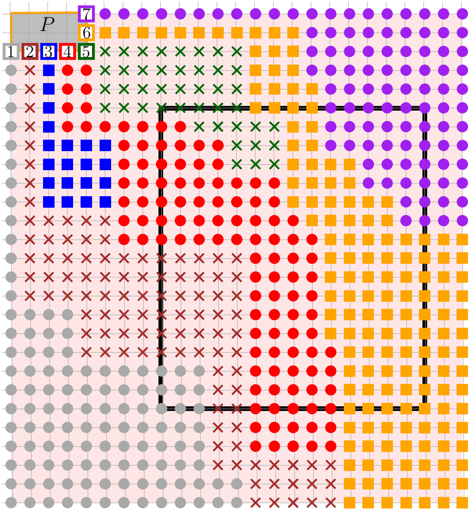
Top-left:

Bottom-right:

Diff:



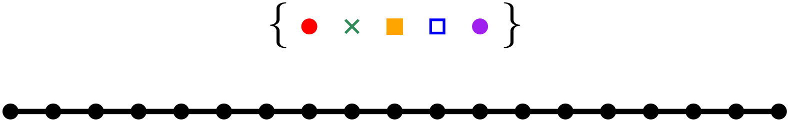
Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.



Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.

We can decompose the boundary using $\tilde{\mathcal{O}}(|S|)$ site-to-vertex distance queries via binary search.

Computing VDs in $\tilde{O}(|S|)$ site-to-vertex Distance Queries

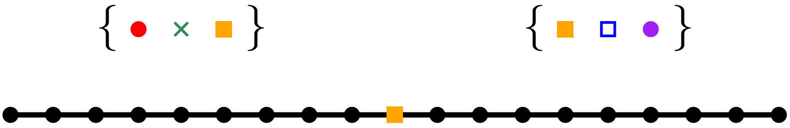


- We know that each color appears in a contiguous interval, and the order of those intervals.

Computing VDs in $\tilde{O}(|S|)$ site-to-vertex Distance Queries

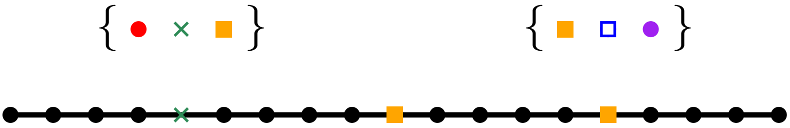
- We know that each color appears in a contiguous interval, and the order of those intervals.
- We first check using $|S|$ queries the color of the middle vertex.

Computing VDs in $\tilde{O}(|S|)$ site-to-vertex Distance Queries

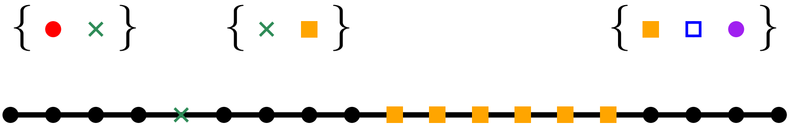


- We know that each color appears in a contiguous interval, and the order of those intervals.
- We first check using $|S|$ queries the color of the middle vertex.
- Our palette is then split, with the color of the middle vertex inherited by both sides.

Computing VDs in $\tilde{O}(|S|)$ site-to-vertex Distance Queries

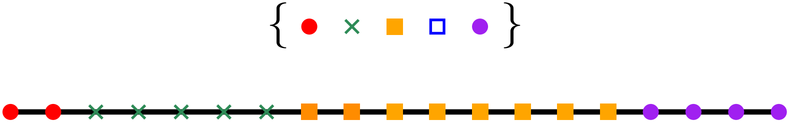


- We know that each color appears in a contiguous interval, and the order of those intervals.
- We first check using $|S|$ queries the color of the middle vertex.
- Our palette is then split, with the color of the middle vertex inherited by both sides.
- We repeat this procedure $\log n$ times.



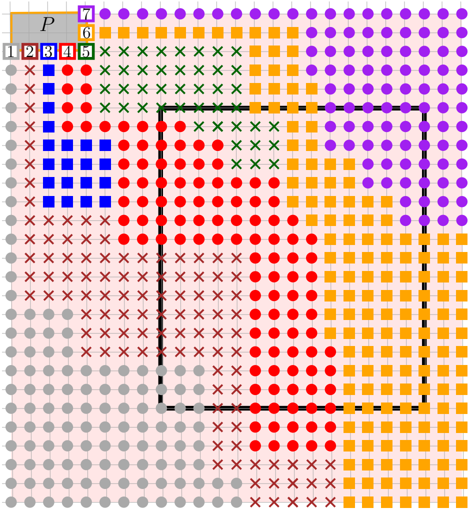
- We know that each color appears in a contiguous interval, and the order of those intervals.
- We first check using $|S|$ queries the color of the middle vertex.
- Our palette is then split, with the color of the middle vertex inherited by both sides.
- We repeat this procedure $\log n$ times.
- In every level, each color is active in at most two intervals.

Computing VDs in $\tilde{O}(|S|)$ site-to-vertex Distance Queries



- We know that each color appears in a contiguous interval, and the order of those intervals.
- We first check using $|S|$ queries the color of the middle vertex.
- Our palette is then split, with the color of the middle vertex inherited by both sides.
- We repeat this procedure $\log n$ times.
- In every level, each color is active in at most two intervals.
- Hence, the algorithm makes $\leq 2|S| \cdot \log n$ queries.

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

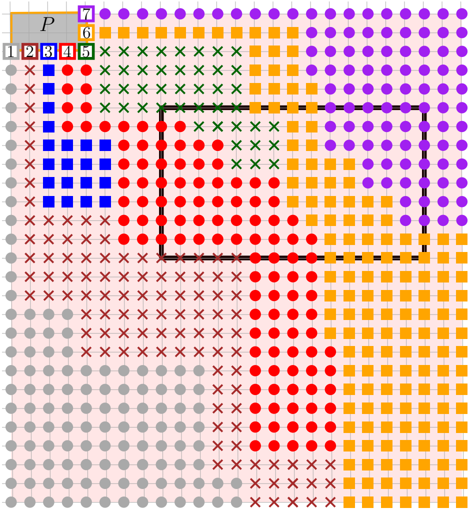


Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.

We can decompose the boundary using $\tilde{\mathcal{O}}(|S|)$ site-to-vertex distance queries via binary search.

This yields an algorithm that uses $\tilde{\mathcal{O}}(|S|^2)$ site-to-vertex distance queries in total: decompose the graph into 2 rectangles and recursively zoom in to interesting ones.

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

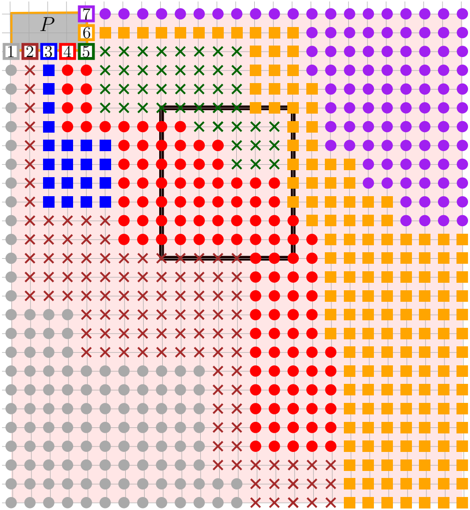


Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.

We can decompose the boundary using $\tilde{\mathcal{O}}(|S|)$ site-to-vertex distance queries via binary search.

This yields an algorithm that uses $\tilde{\mathcal{O}}(|S|^2)$ site-to-vertex distance queries in total: decompose the graph into 2 rectangles and recursively zoom in to interesting ones.

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

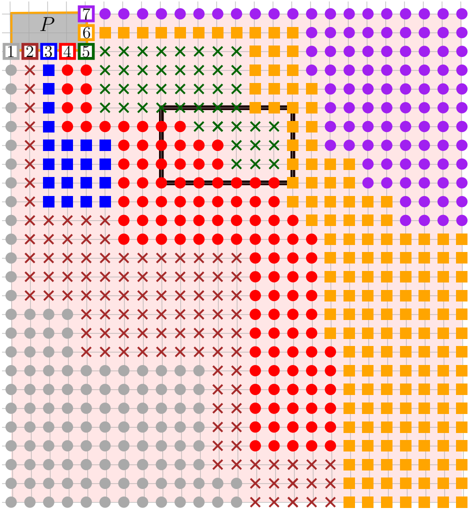


Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.

We can decompose the boundary using $\tilde{\mathcal{O}}(|S|)$ site-to-vertex distance queries via binary search.

This yields an algorithm that uses $\tilde{\mathcal{O}}(|S|^2)$ site-to-vertex distance queries in total: decompose the graph into 2 rectangles and recursively zoom in to interesting ones.

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

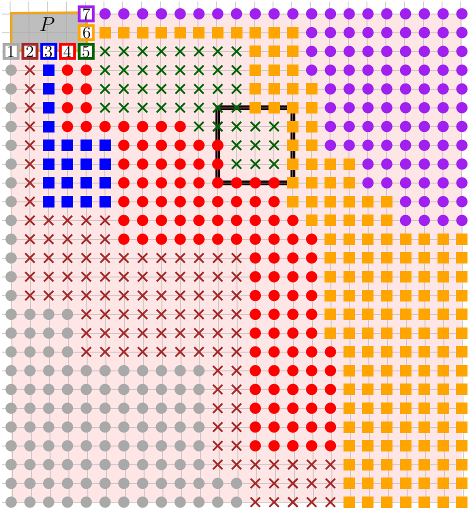


Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.

We can decompose the boundary using $\tilde{\mathcal{O}}(|S|)$ site-to-vertex distance queries via binary search.

This yields an algorithm that uses $\tilde{\mathcal{O}}(|S|^2)$ site-to-vertex distance queries in total: decompose the graph into 2 rectangles and recursively zoom in to interesting ones.

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

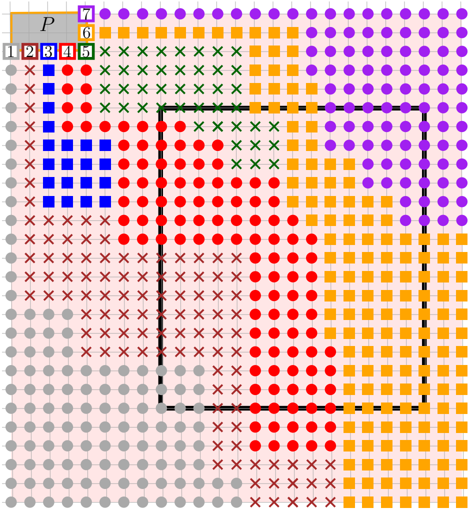


Auxiliary operation: Decide whether a rectangle contains $\ell(s)$ for any $s \in S$ by looking at its boundary.

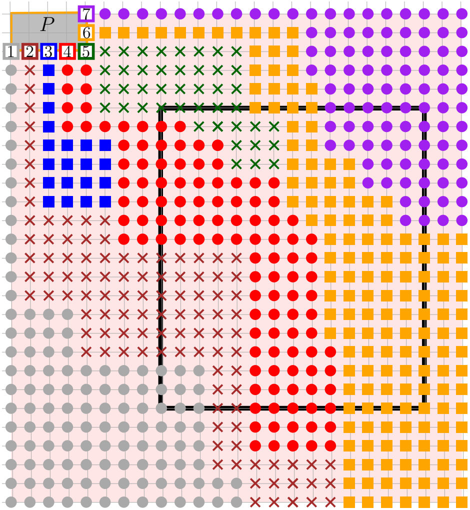
We can decompose the boundary using $\tilde{\mathcal{O}}(|S|)$ site-to-vertex distance queries via binary search.

This yields an algorithm that uses $\tilde{\mathcal{O}}(|S|^2)$ site-to-vertex distance queries in total: decompose the graph into 2 rectangles and recursively zoom in to interesting ones.

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$. Disregard irrelevant sites in recursion.



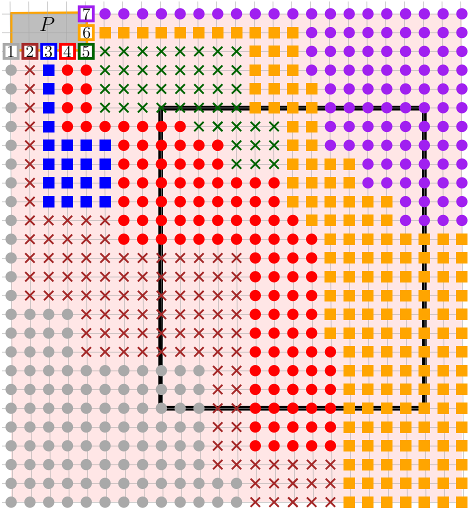
Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.



Disregard irrelevant sites in recursion.

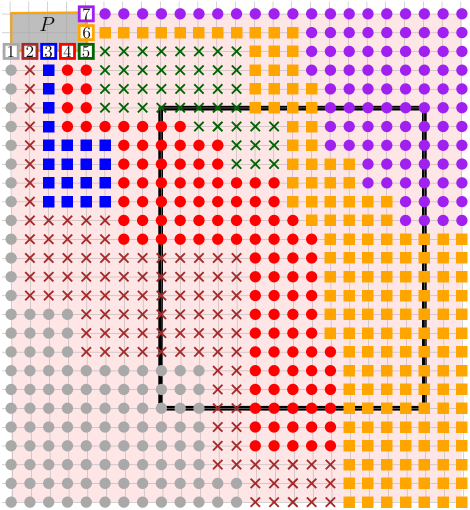
Sites whose cells do not touch the rectangle.

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$. Disregard irrelevant sites in recursion.



Sites whose cells do not touch the rectangle. E.g. the blue site (no. 3).

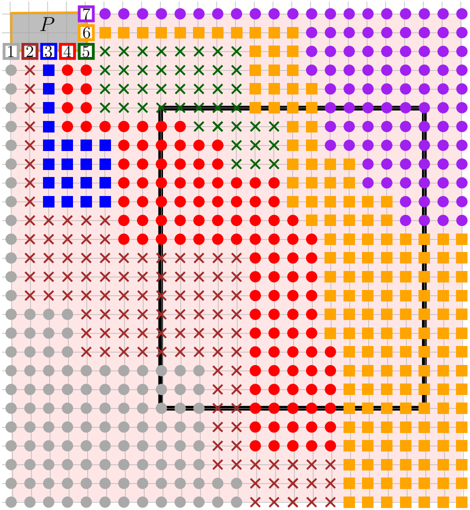
Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$. Disregard irrelevant sites in recursion.



Sites whose cells do not touch the rectangle. E.g. the blue site (no. 3).

If there are three sites that "enter" and "exit" the rectangle next to each other, we can remove the middle one.

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.

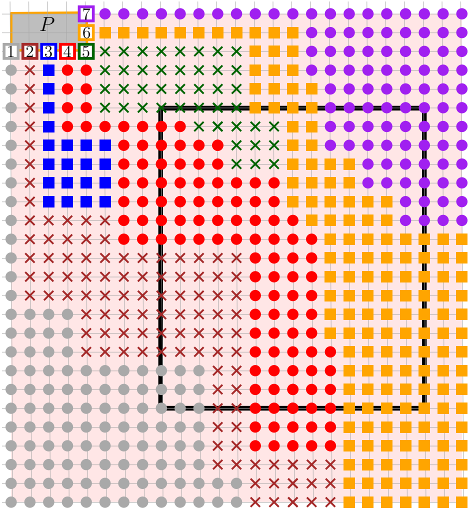


Disregard irrelevant sites in recursion.

Sites whose cells do not touch the rectangle. E.g. the blue site (no. 3).

If there are three sites that "enter" and "exit" the rectangle next to each other, we can remove the middle one. E.g. the brown site (no. 2).

Aim: Compute $L = \{(s, \ell(s)) : s \in S\}$.



Disregard irrelevant sites in recursion.

Sites whose cells do not touch the rectangle. E.g. the blue site (no. 3).

If there are three sites that "enter" and "exit" the rectangle next to each other, we can remove the middle one. E.g. the brown site (no. 2).

In each rectangle $\square$, we consider $\mathcal{O}(|L \cap \square|)$ sites in our binary search.

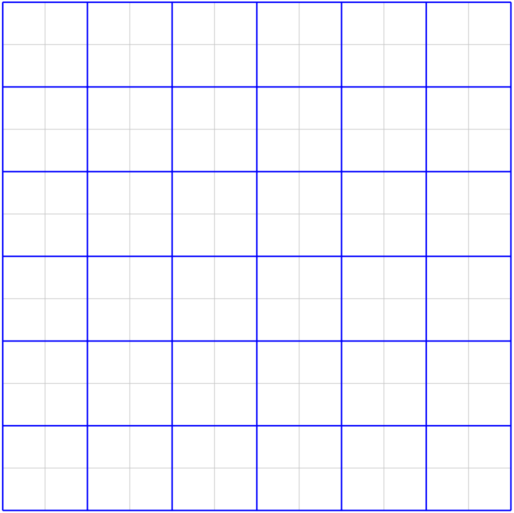
Reminder of Warm-up II

Component	Prep-time	Space
Internal MSSPs	$N/r \cdot \tilde{\mathcal{O}}(r)$	$N/r \cdot \tilde{\mathcal{O}}(r)$
External MSSPs	$N/r \cdot \tilde{\mathcal{O}}(N)$	$N/r \cdot \tilde{\mathcal{O}}(N)$
Voronoi diagrams	$N \cdot \tilde{\mathcal{O}}(\sqrt{r})$	$N \cdot \mathcal{O}(\sqrt{r})$
Total:	$\tilde{\mathcal{O}}(N^2/r + N \cdot \sqrt{r})$	$\tilde{\mathcal{O}}(N^2/r + N \cdot \sqrt{r})$

Almost-optimality via Recursion

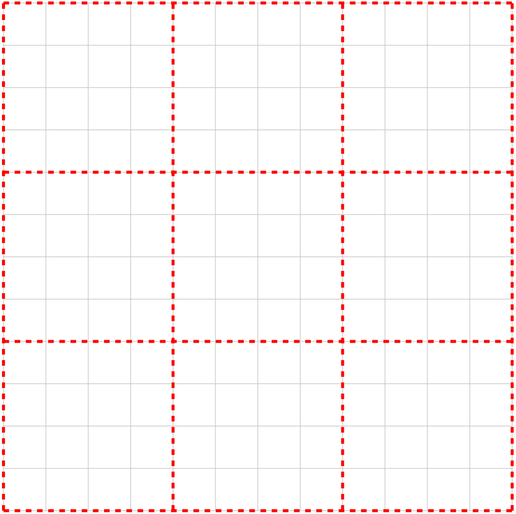
We will see a two-level approach.

Almost-optimality via Recursion



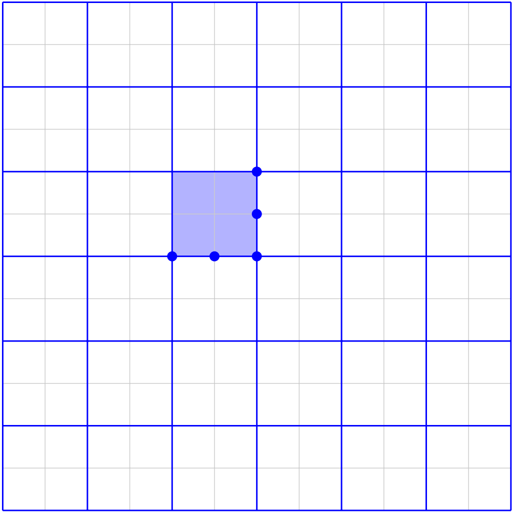
Small pieces of size r.

Almost-optimality via Recursion



Large pieces of size R.

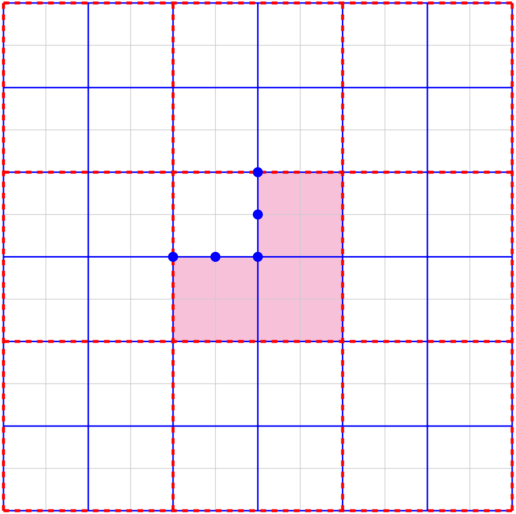
Almost-optimality via Recursion



Internal MSSPs: $\tilde{\mathcal{O}}(N)$

We store internal MSSPs for small pieces. Prep-time: $\tilde{\mathcal{O}}(N)$.

Almost-optimality via Recursion

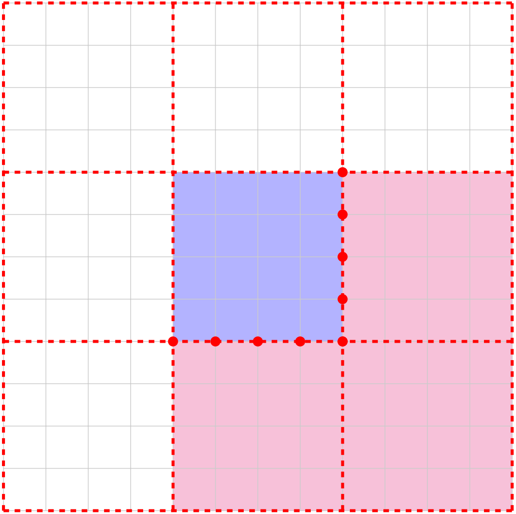


Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R)$

We store restricted external MSSPs for small pieces.
Prep-time: $N/r \cdot \tilde{\mathcal{O}}(R)$.

Almost-optimality via Recursion

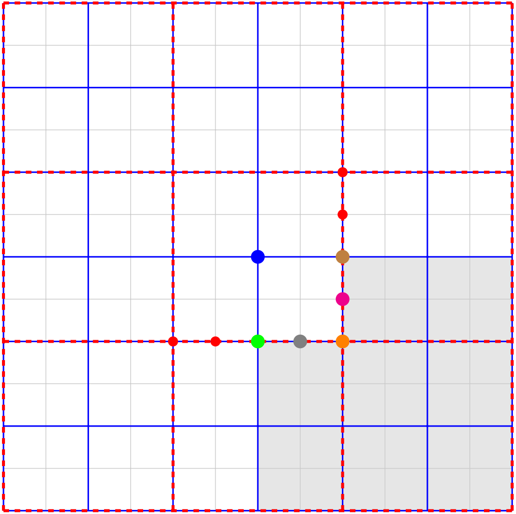


Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

For large pieces, we store standard internal and external MSSPs.
Prep-time: $\tilde{\mathcal{O}}(N + N^2/R)$.

Almost-optimality via Recursion



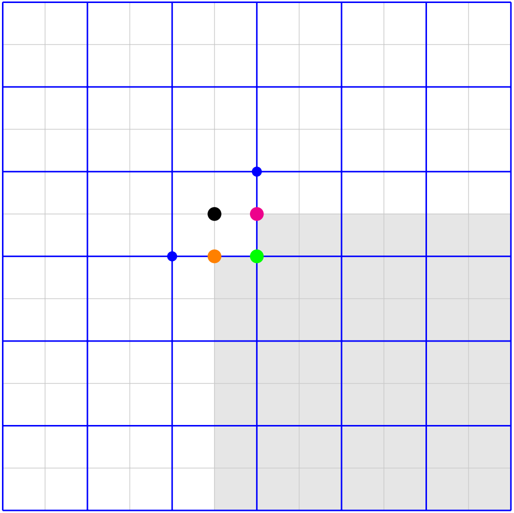
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

For each blue vertex, we store a Voronoi diagram wrt a large piece containing it. Prep-time: $N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$.

Almost-optimality via Recursion



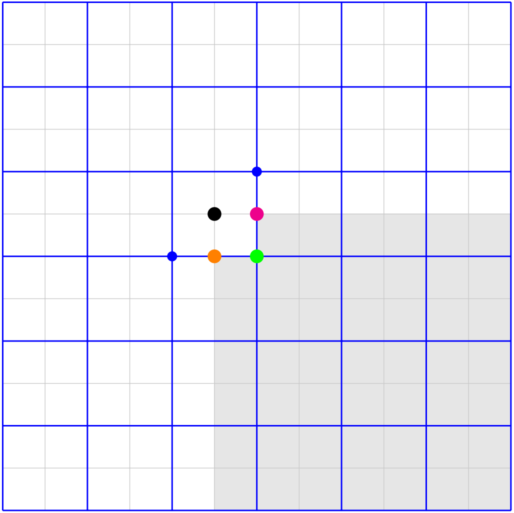
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

For each vertex, we store a Voronoi diagram wrt a small piece containing it.

Almost-optimality via Recursion



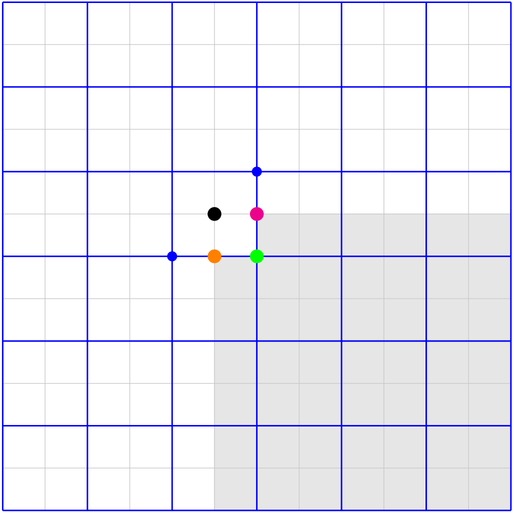
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

We already know how to answer site-to-vertex distance queries!

Almost-optimality via Recursion



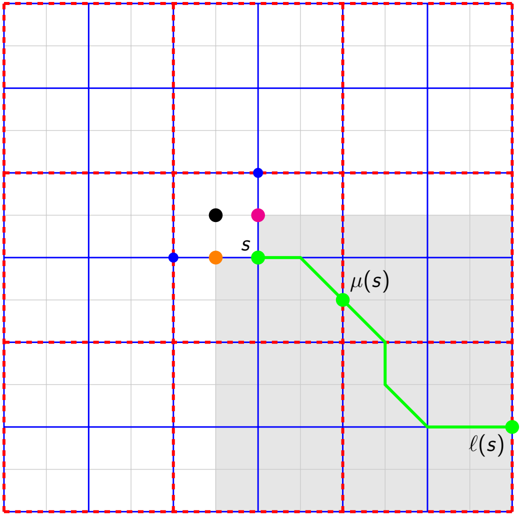
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

For each vertex, we store a Voronoi diagram wrt a small piece containing it. Prep-time: $N \cdot \tilde{\mathcal{O}}(\sqrt{r})$.

Almost-optimality via Recursion



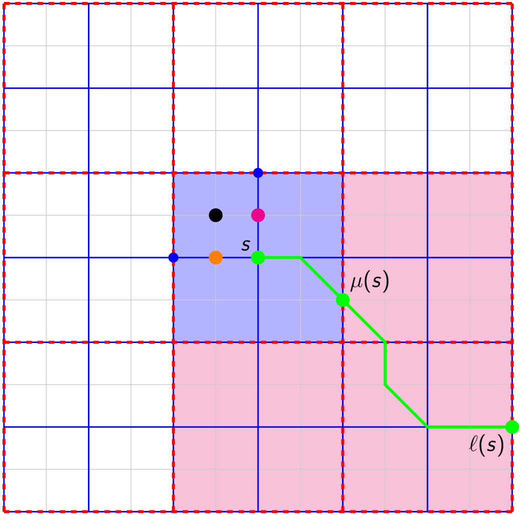
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

For each site $s \in S$, we also store a middle vertex $\mu(s)$, to enable the left/right procedure that ends up with two candidates.

Almost-optimality via Recursion



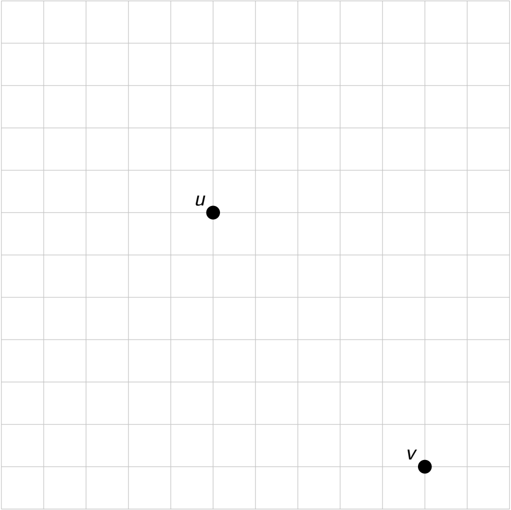
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

For each site $s \in S$, we also store a middle vertex $\mu(s)$, to enable the left/right procedure that ends up with two candidates.

Almost-optimality via Recursion

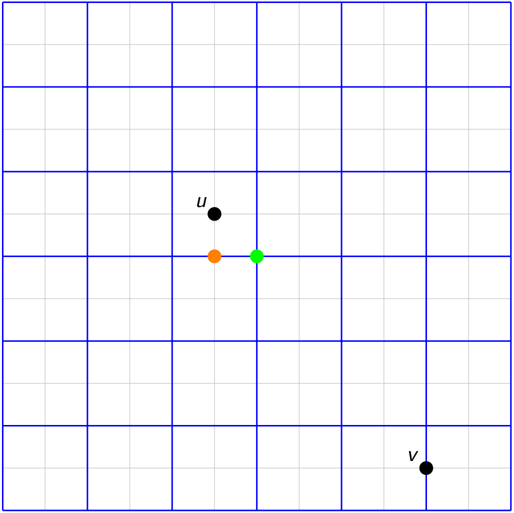


Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

Almost-optimality via Recursion



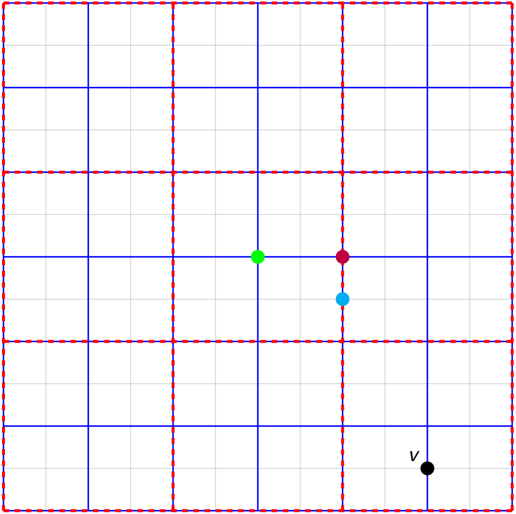
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

First, we obtain two candidate sites in the boundary of a small piece containing u.

Almost-optimality via Recursion



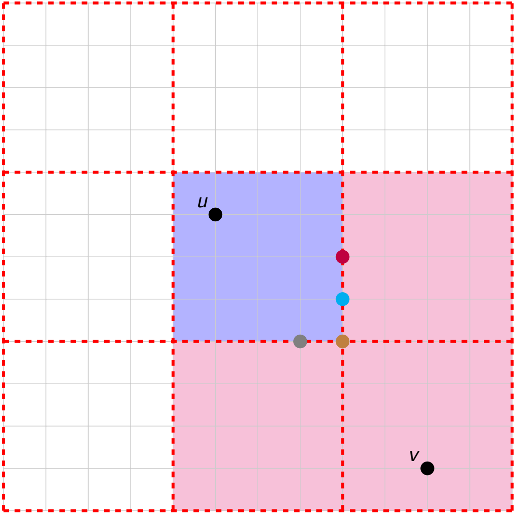
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

For each of them, we obtain two candidate sites on the boundary of a large piece containing u.

Almost-optimality via Recursion



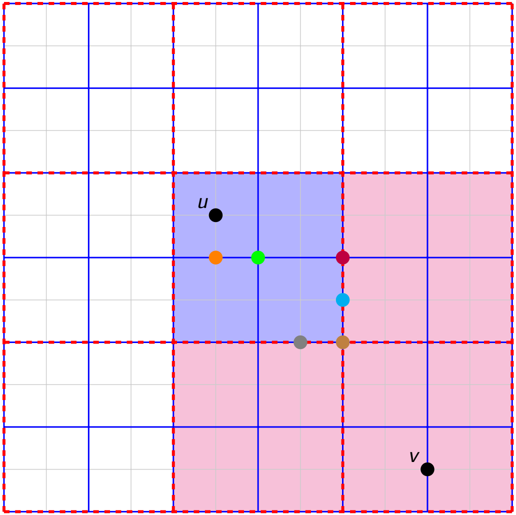
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

Finally, we check all candidates using our MSSP data structures.
Query time: $\mathcal{O}(\log^2 n)$.

Almost-optimality via Recursion



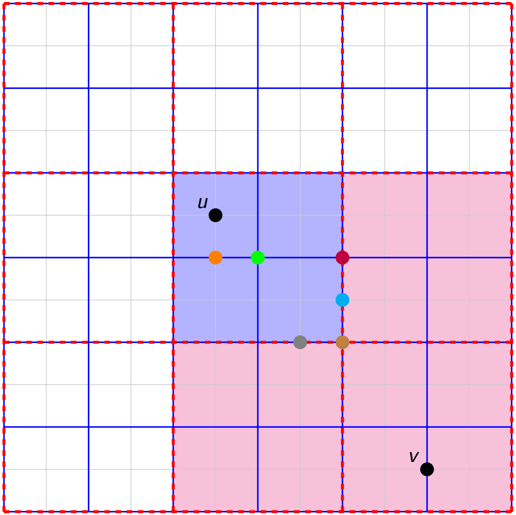
Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

Total:
$\tilde{\mathcal{O}}(N \cdot (\sqrt{r} + R/r + N/R))$

Almost-optimality via Recursion



Internal MSSPs: $\tilde{\mathcal{O}}(N)$

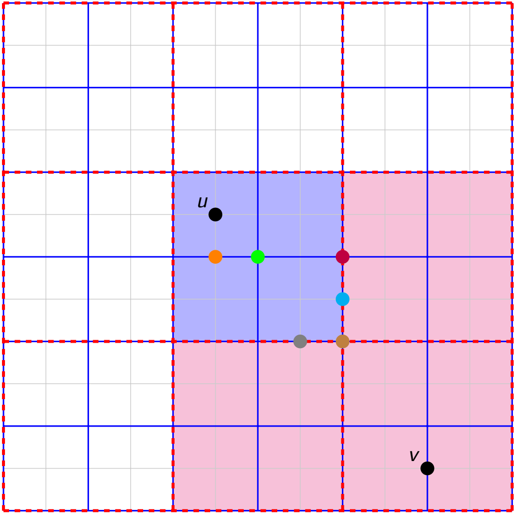
External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

Total:
$\tilde{\mathcal{O}}(N \cdot (\sqrt{r} + R/r + N/R))$

By setting $r = \sqrt{N}$ and $R = N^{3/4}$, we get $\tilde{\mathcal{O}}(N^{5/4})$ prep-time.

Almost-optimality via Recursion



Internal MSSPs: $\tilde{\mathcal{O}}(N)$

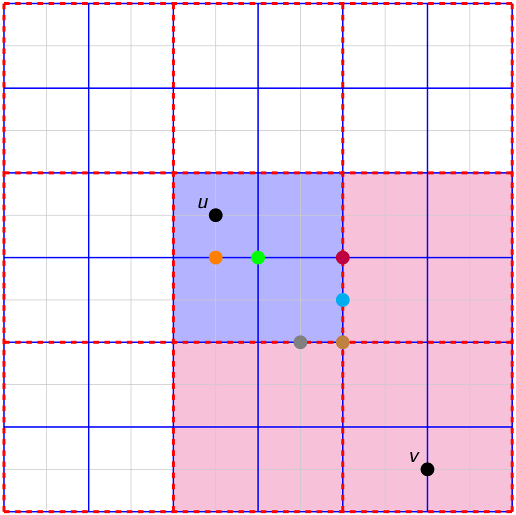
External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

Total:
$\tilde{\mathcal{O}}(N \cdot (\sqrt{r} + R/r + N/R))$

Using t levels, with piece-sizes $r_1 = \Theta(1), \ldots, r_t = \Theta(N)$: query time $\tilde{\mathcal{O}}(2^t)$, space $\tilde{\mathcal{O}}(N \cdot \sum_i \frac{r_{i+1}}{r_i})$, prep-time $\tilde{\mathcal{O}}(\text{space} \cdot 2^t)$.

Almost-optimality via Recursion



Internal MSSPs: $\tilde{\mathcal{O}}(N)$

External MSSPs:
$N/r \cdot \tilde{\mathcal{O}}(R) + N/R \cdot \tilde{\mathcal{O}}(N)$

Voronoi diagrams:
$N \cdot \tilde{\mathcal{O}}(\sqrt{r}) + N/\sqrt{r} \cdot \tilde{\mathcal{O}}(\sqrt{R})$

Total:
$\tilde{\mathcal{O}}(N \cdot (\sqrt{r} + R/r + N/R))$

Using t levels, with piece-sizes $r_1 = \Theta(1), \ldots, r_t = \Theta(N)$: query time $\log^{2+o(1)} n$, space $N^{1+o(1)}$, prep-time $N^{1+o(1)}$.

Open Problems

	Preprocessing	**Space**	**Query**
	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(N)$	$N^{o(1)}$
$t \in [\sqrt{N}, N]$	$\tilde{\mathcal{O}}(N)$	$\tilde{\mathcal{O}}(N/\sqrt{t})$	$\tilde{\mathcal{O}}(t)$

Open Problems

	Preprocessing	**Space**	**Query**
	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(N)$	$N^{o(1)}$
$t \in [\sqrt{N}, N]$	$\tilde{\mathcal{O}}(N)$	$\tilde{\mathcal{O}}(N/\sqrt{t})$	$\tilde{\mathcal{O}}(t)$

- How close to $\mathcal{O}(N)$ prep-time, $\mathcal{O}(1)$ query time can we get?

Open Problems

	Preprocessing	Space	Query
	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(N)$	$N^{o(1)}$
$t \in [\sqrt{N}, N]$	$\tilde{\mathcal{O}}(N)$	$\tilde{\mathcal{O}}(N/\sqrt{t})$	$\tilde{\mathcal{O}}(t)$

- How close to $\mathcal{O}(N)$ prep-time, $\mathcal{O}(1)$ query time can we get?
- Further investigate the space vs query time tradeoff.

Open Problems

	Preprocessing	Space	Query
	$N^{1+o(1)}$	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(1)$
	$N^{1+o(1)}$	$\tilde{\mathcal{O}}(N)$	$N^{o(1)}$
$t \in [\sqrt{N}, N]$	$\tilde{\mathcal{O}}(N)$	$\tilde{\mathcal{O}}(N/\sqrt{t})$	$\tilde{\mathcal{O}}(t)$

- How close to $\mathcal{O}(N)$ prep-time, $\mathcal{O}(1)$ query time can we get?
- Further investigate the space vs query time tradeoff.
- Do our ideas extend to any subclass of planar graphs?

Thank You

Thank you for your attention!