Dynamic Unit-Disk Range Reporting

Haitao Wang and **Yiming Zhao**

University of Utah   Metropolitan State University of Denver

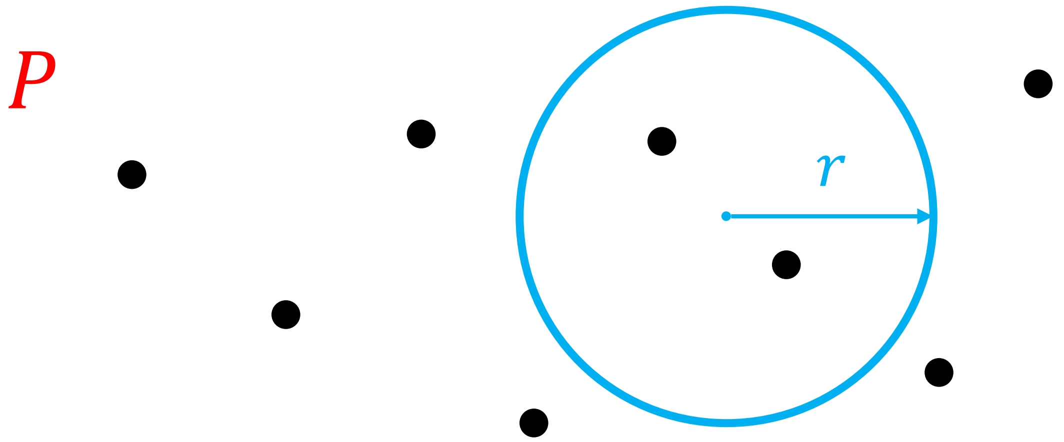
42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025)
March 05, 2025
Jena, Germany

Full version: https://arxiv.org/pdf/2501.00120

Unit disk range reporting (UDRR) problem

Static UDRR: Given a set P of n points in the plane and a value $r > 0$,

build a data structure such that given any query disk of radius r, all points of P within the query disk can be reported.



Dynamic UDRR: Points of P can be inserted and deleted.

Static UDRR: Related work and our results

	Preprocessing Performance	**Query Time, k is the output size**
Chazelle and Edelsbrunner, 1985	$O(n^2)$ time and $O(n)$ space	$O(\log n + k)$
Reduce UDRR to the half-space range reporting queries in 3D (*works for query disk of arbitrary radius*), and Afshani and Chan's 3D half-space range reporting data structure (2009)	$O(n \log n)$ expected time (due to Ramos' algorithm (1999), to construct shallow cuttings for a set of planes in 3D) and $O(n)$ space	$O(\log n + k)$
Same as above but using a deterministic shallow cutting algorithm by Chan and Tsakalidis (2016)	$O(n \log n)$ deterministic time and $O(n)$ space	$O(\log n + k)$
Our results, much simpler	$O(n \log n)$ deterministic time and $O(n)$ space	$O(\log n + k)$

Dynamic UDRR: Related work and our results

Previous work:

1. Reduce it to dynamic halfspace range reporting in 3D using the standard lifting transformation.
2. The currently best result: $O(n \log n)$ space, $O(\log^{3+\varepsilon} n)$ amortized insertion time, $O(\log^{5+\varepsilon} n)$ amortized deletion time, and $\boldsymbol{O}(\frac{\log^2 n}{\log \log n} + k)$ query time, where ε is an arbitrarily small positive constant and k is the output size.

Our result:

Optimal $\boldsymbol{O}(\log n + k)$ query time, and the space and the update time complexities are the same as above.

Dynamic unit-disk range emptiness queries

P: a dynamic set of points in the plane. Determine whether a query unit disk contains any point of P, and if so, return such a point.

Previous work:

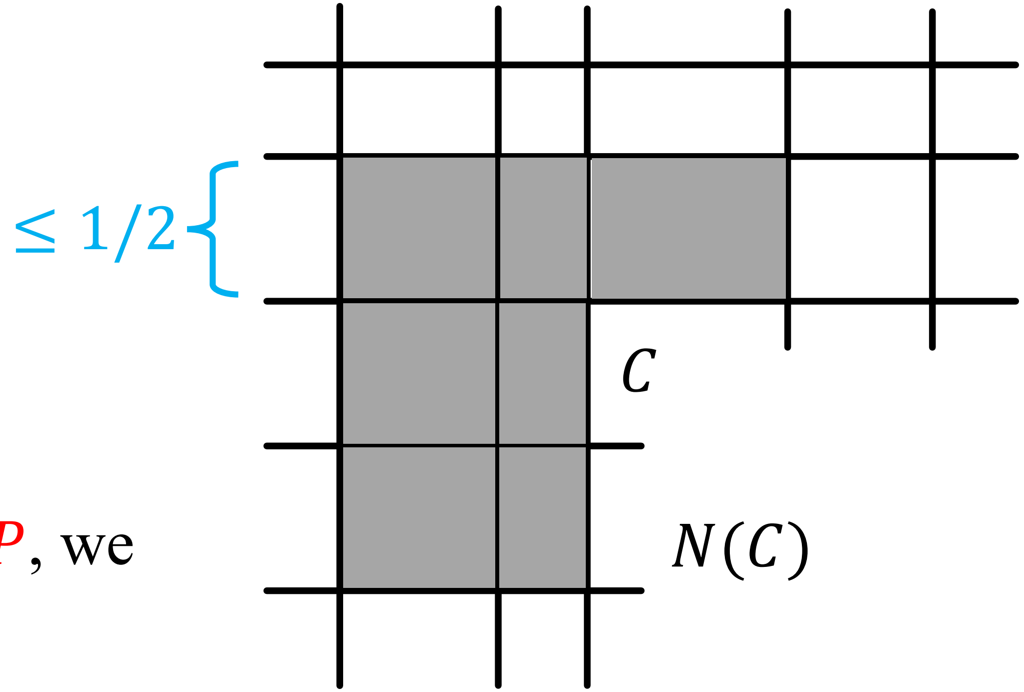
Using a dynamic nearest neighbor search data structure [Chan, 2020]. $O(n)$ space, $O(\log^2 n)$ amortized insertion time, $O(\log^4 n)$ amortized deletion time, and $O(\log^2 n)$ query time.

Our result:

$O(n)$ space, $O(\log^{1+\varepsilon} n)$ amortized insertion time, $O(\log^{1+\varepsilon} n)$ amortized deletion time, and $O(\log n)$ query time.

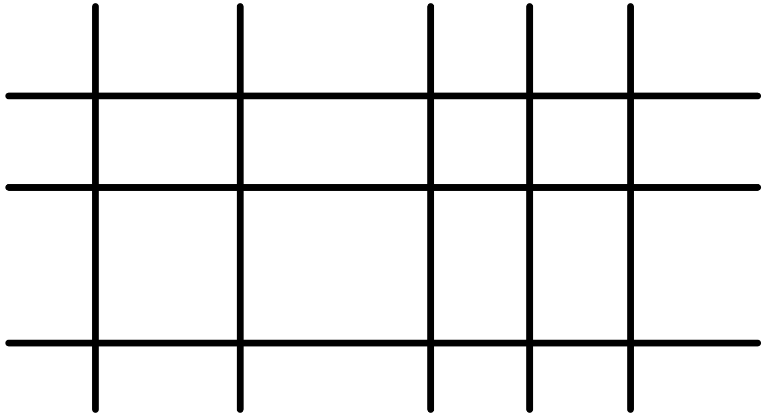
A grid technique

R: a region in the plane
$P(R)$: the subset of points of P inside R.
D_q: the unit disk centered at point q.



We use a set of **grid cells** to capture the **proximity information** for the points of P. To handle updates to P, we define a **conforming coverage** $\mathcal{C}$ for P as a set of cells (axis-parallel rectangle).

1. The union of all cells of $\mathcal{C}$ covers all the points of P.
2. Each cell $C \in \mathcal{C}$ is associated with a subset $N(C) \in \mathcal{C}$ of $O(1)$ neighboring cells, such that for any point $q \in C$, $P(D_q) \subseteq \bigcup_{C' \in N(C)} P(C')$.
3. For any point q, if q is not in any cell of $\mathcal{C}$, then $P \cap D_q = \emptyset$.



Maintain the grid dynamically

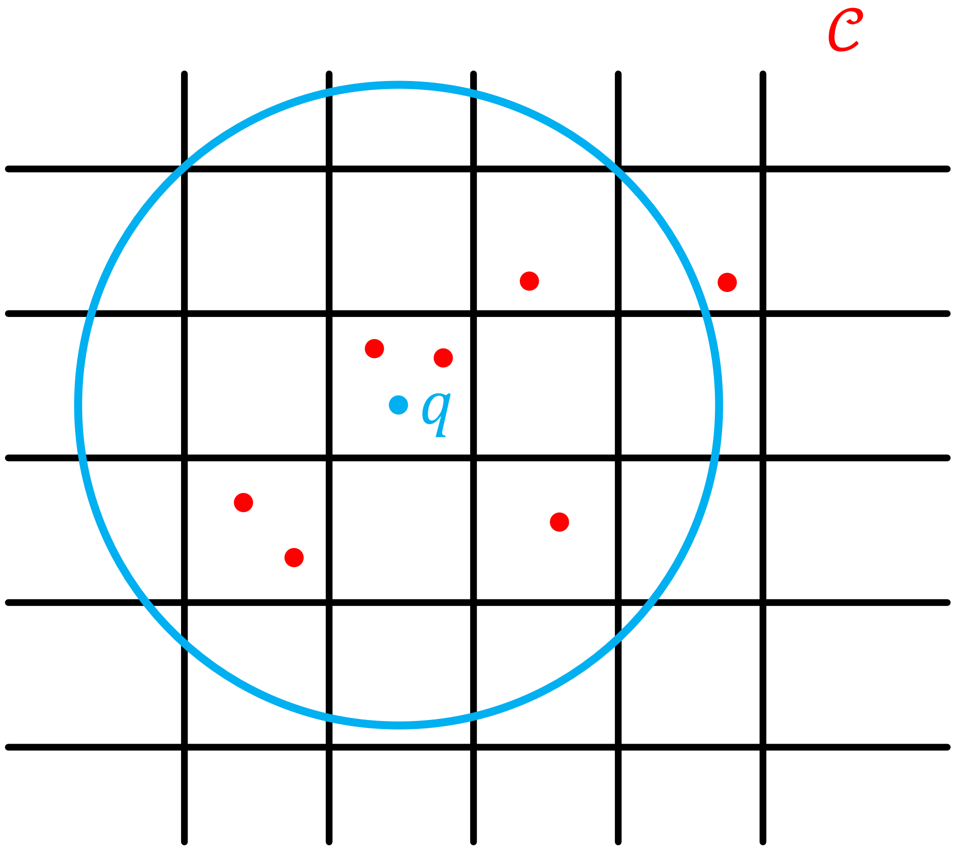
1. A conforming coverage set $\mathcal{C}$ of $O(n)$ cells for P can be maintained in $O(n)$ space (n is the size of the current set P).
2. Each point insertion of P: $O(\log n)$ worst-case time.
3. Each point deletion of P: $O(\log n)$ amortized time.
4. Given any point q, determine whether q is in a cell C of $\mathcal{C}$, and if so, return C and $N(C)$: $O(\log n)$ time.

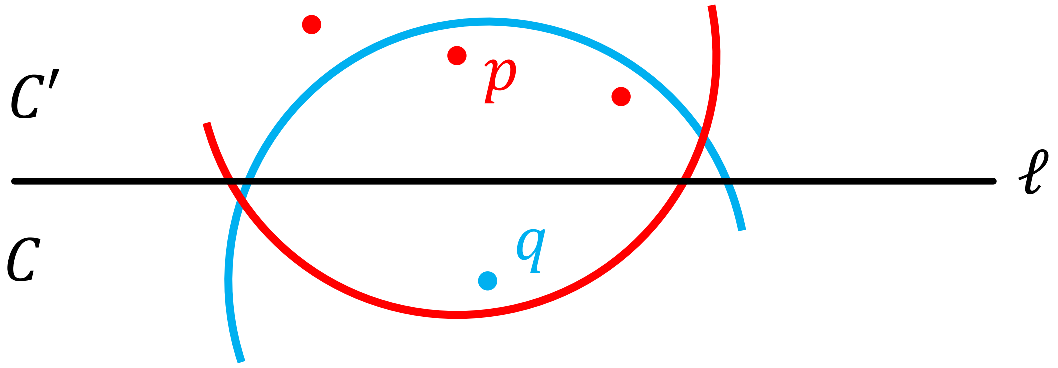
Corollary: A static conforming coverage can be built in $O(n)$ space and $O(n \log n)$ time.

Line-separable UDRR problem

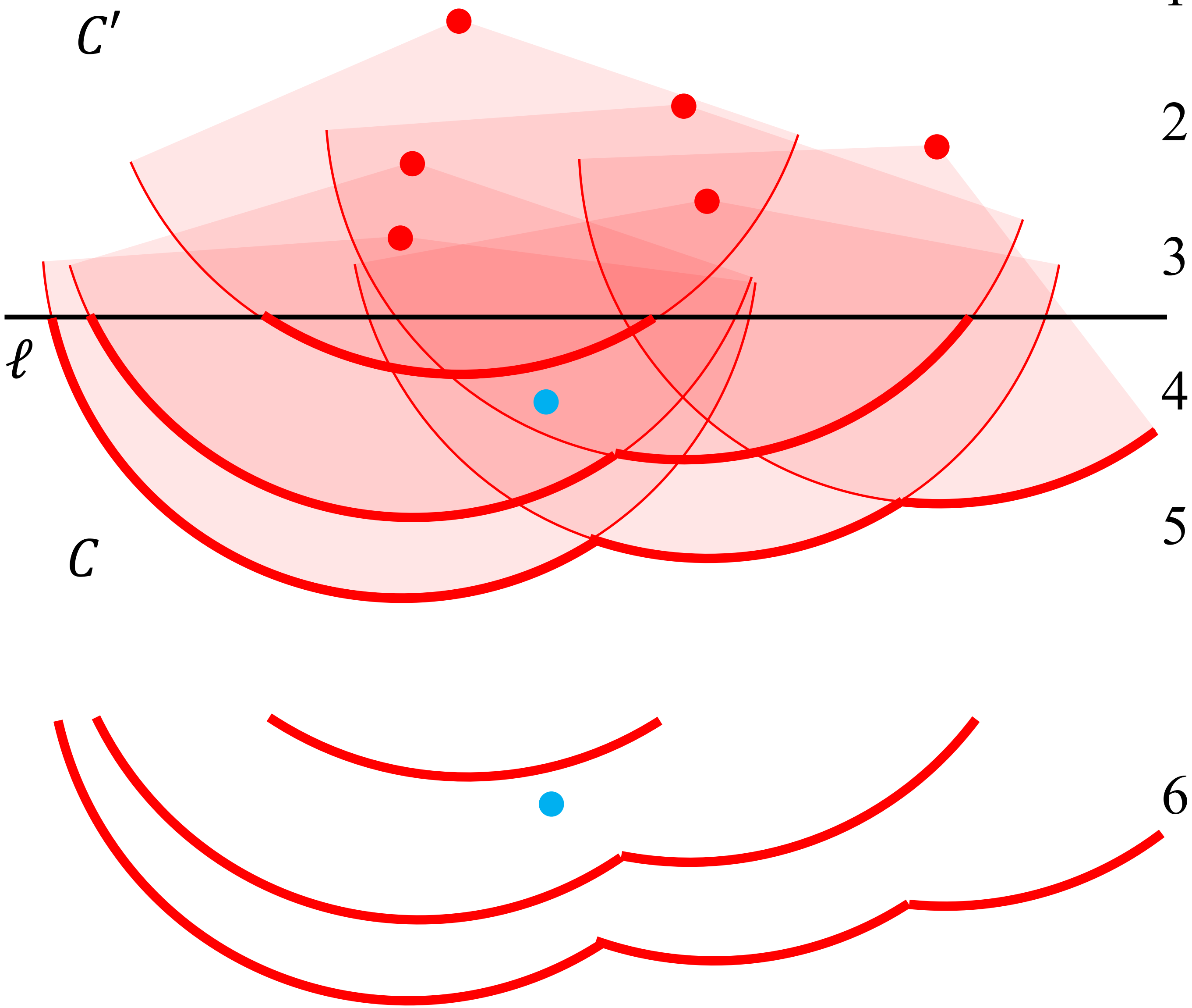
Given a query disk D_q centered at q,

1. If q is not in a cell of $\mathcal{C}$, then $P \cap D_q = \emptyset$, we return null.
2. If $q \in C \in \mathcal{C}$, we report all points of P in C. For every neighboring cell $C' \in N(C)$, we have the the **line-separable UDRR** in both **static** and **dynamic** version.
3. A point $p \in P$ is in D_q iff q is above the arc centered at p. The arc below ℓ is ***x*-monotone**.



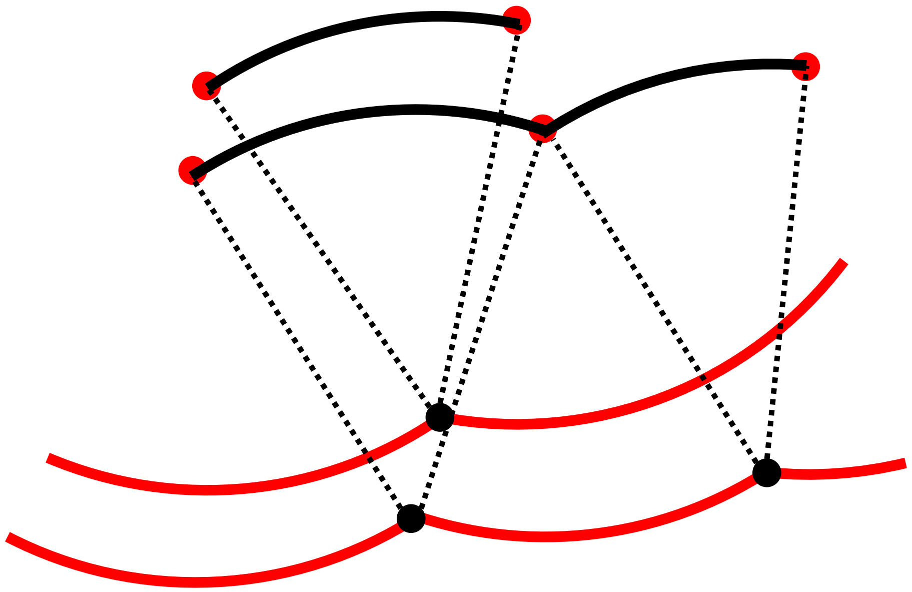


Static line-separable UDRR algorithm



1. $\mathcal{U}_1$: the lower envelope of all arcs centered at points in C' below line ℓ.
2. $\mathcal{U}_1$ is spliced with several arcs. We find out the arc that spans point q.
3. If q is below this arc, then no arc is below q and we return null.
4. If q is above this arc, then the center of such an arc can be reported.
5. Moving leftwards and rightwards on the lower envelope to check whether q is above the next arc until we firstly see q is below an arc or we see line ℓ.
6. We remove all arcs in $\mathcal{U}_1$ and run the above procedure on the next lower envelope layer $\mathcal{U}_2, \mathcal{U}_3, \ldots$ until q is below a lower envelope.

Static UDRR algorithm

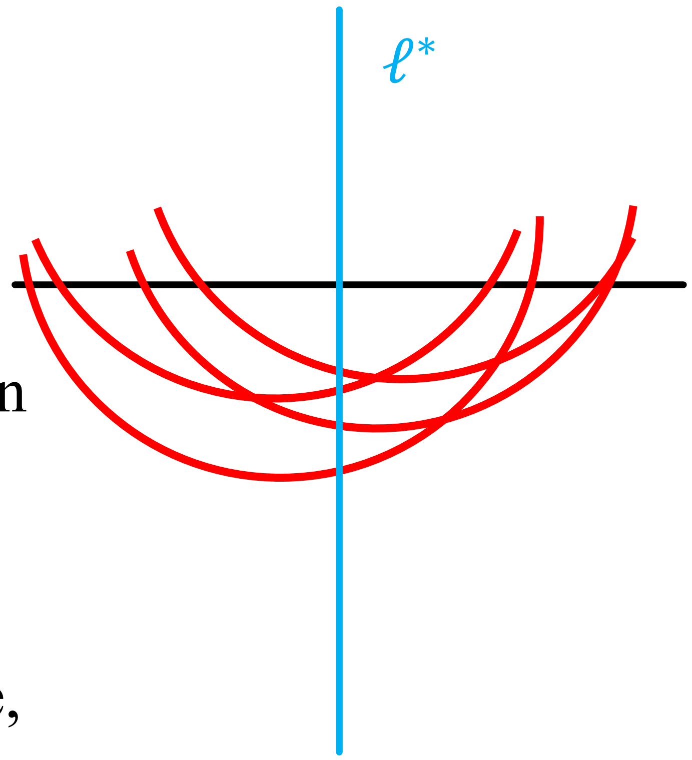


1. Lower envelope layers $\{\mathcal{U}_1, \mathcal{U}_2, \dots\}$ can be computed in $O(|C'| \log |C'|)$ time and $O(|C'|)$ space by considering its dual problem: computing **lower α-hull layers**.
2. The computation of **lower α-hull layers** follows the scheme of Chazelle's algorithm in 1985 for computing **convex hull layers**.
3. A **fractional cascading** data structure is used on the vertices of the lower envelope layers so that given any point q, the arc spanning q in $\mathcal{U}_1$ is reported in $O(\log n)$ time and the arc spanning q in other envelope layers is reported in $O(1)$ time each. The query time is $O(\log|C'| + k)$.

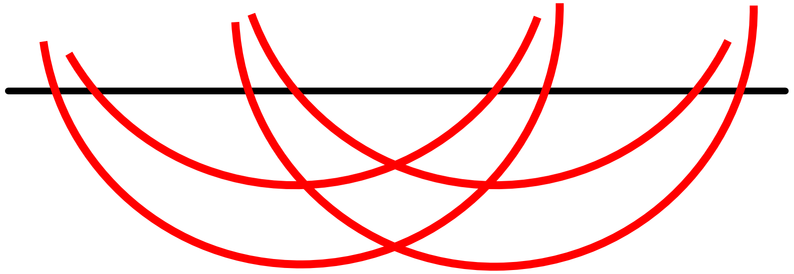
Build such a data structure for every non-empty cell in the conforming coverage set $\mathcal{C}$ with respect to each of its supporting line. The static UDRR can be solved in $O(n \log n)$ time and $O(n)$ space, and the query time is $O(\log n + k)$.

Dynamic line-separable UDRR algorithm

1. The problem requires reporting arcs of a dynamic set that are below a query point.
2. The k-lowest-arcs queries: Given a query vertical line ℓ^* and a number $k \geq 1$, report the k lowest arcs intersecting ℓ^*.
3. **[Chan, 2000]**: If each k-lowest-arcs query can be answered in $O(\log n + k)$ time, then the arcs of a dynamic set below a query point can be reported in $O(\log n + k)$ time.
4. We adapt the technique for a similar problem: Dynamically maintain a set of lines to report the ***k*-lowest lines** at a query vertical line.
   1) **[Chan, 2012]**: $O(n \log n)$ space, $O(\log^{6+\varepsilon} n)$ amortized update time, and $O(\log n + k)$ query time.
   2) **[De Berg and Staals, 2023]**: for planes in 3D, $O(n \log n)$ space, $O(\log^{3+\varepsilon} n)$ amortized insertion time, $O(\log^{5+\varepsilon} n)$ amortized deletion time, and $O(\log^2 n / \log\log n + k)$ query time.



A new shallow cutting algorithm for arcs



Theorem: There exist constants B, C, and C', such that for a parameter $k \in [1, n]$, we can compute a $(B^i k)$-shallow $(CB^i k/n)$-cutting of size at most $C' \frac{n}{B^i k}$ in the bottom-open pseudo-trapezoid form, along with conflict lists of all its cells, for all $i = 0,1,\ldots,\log_B \frac{n}{k}$, in $O(n \log \frac{n}{k})$ time.

Corollary: We can compute a k-shallow (Ck/n)-cutting of size $O(\frac{n}{k})$, along with its conflict lists, in $O(n \log \frac{n}{k})$ time.

Two deletion-only data structures for k-lowest-arcs queries

1. Using our shallow cutting algorithm for arcs and a generalized partition tree technique ([Matoušek, 1992] and [Wang, 2023]): A set of n arcs can be maintained in $O(n)$ space to support $O(\log n)$ amortized time deletions and $O(\sqrt{n}\log^{O(1)} n + k)$ time k-lowest-arcs queries.

2. Following the scheme of [De Berg and Staals, 2023] and apply our shallow cutting algorithm for arcs: For any fixed r, a set of n arcs can be maintained in $O(n \log r)$ space to support $O(r \log n)$ amortized time deletions and $O(\log r + \frac{n}{r} + k)$ time k-lowest-arcs queries.

Dynamic line-separable UDRR algorithm

Γ :a dynamic set of arcs, which initially is $\emptyset$, and undergoes n updates (insertions and deletions).

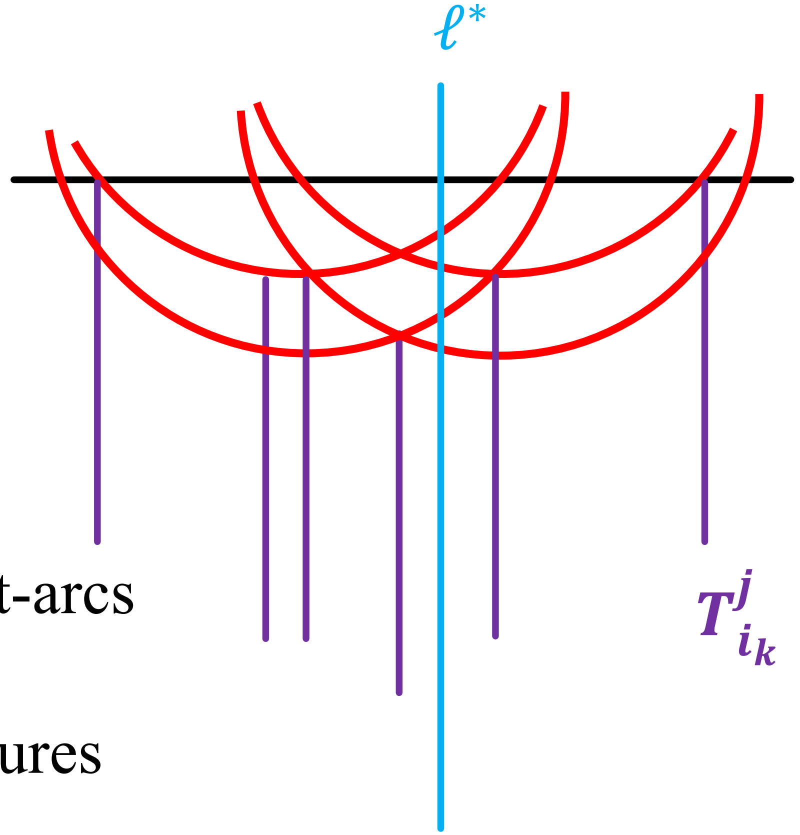
For any $b \geq 2$, we can maintain a collection of shallow cuttings $\boldsymbol{T_i^j}$ in the bottom-open pseudo-trapezoid form, $i = 1,2, \ldots, \lceil \log n \rceil$, $j = 1,2, \ldots, O(\log_b n)$, such that,

1. The conflict list of every cell of every cutting only undergoes deletions after its creation.
2. For any $k \geq 1$, let $i_k = \left\lceil \log\left(\frac{n}{Ck}\right) \right\rceil$ for a sufficiently large constant C. For any vertical line ℓ^*, if an arc $\gamma \in \Gamma$ is among the k lowest arcs at ℓ^*, then there exists a j such that γ is in the conflict list L_{Δ^j} of the cell $\Delta^j \in T_{i_k}^j$ intersecting ℓ^*.

Dynamic line-separable UDRR algorithm

Answer a k-lowest-arcs query with a query vertical line ℓ^*.

1. We have a group of shallow cuttings $\boldsymbol{T_{i_k}^j}$, where $j = 1,2,\ldots,O(\log_b n)$.
2. For every j, we compute cell $\Delta^j \in T_{i_k}^j$ intersecting ℓ^*.
   1) For each i, maintain a **dynamic interval tree** to store the intervals of the x-projections of the cuttings $\boldsymbol{T_i^j}$ for all j.
3. Find the k lowest arcs from all conflict lists L_{Δ^j} for all j.
   1) Use our different deletion-only data structures for k-lowest-arcs queries for lists L_{Δ^j} depending on whether $|L_{\Delta^j}| \geq \log^3 n$.
   2) A technique of querying multiple k-lowest-arcs data structures simultaneously [De Berg and Staals, 2023] is applied.
4. Our dynamic line-separable UDRR algorithm: $O(|C'|\log|C'|)$ space, $O(\log^{3+\varepsilon}|C'|)$ amortized insertion time, $O(\log^{5+\varepsilon}|C'|)$ amortized deletion time, and $O(\log|C'|+k)$ query time.



THANKS