Window Queries for Intersecting Objects, Maximal Points and Approximations using Coresets[⋆]

Farah Chanchary[*], Anil Maheshwari, Michiel Smid

School of Computer Science, Carleton University, Ottawa, ON, K1S 5B6, Canada

Abstract

We present data structures that can answer *window queries* for a sequence of geometric objects, such as points, line segments, triangles and convex c-gons. We first present data structures to solve windowed intersection decision problems for line segments, triangles and convex c-gons. We also present data structures to count points on maximal layer, k-dominated and k-dominant points for some fixed integer k, and to decide whether a given point belongs to a maximal layer for a sequence of points in $\mathbb{R}^d$, $d \geq 2$. Finally we present techniques to approximate window-aggregate queries for $(1+\epsilon)$-approximations for various geometric measures such as diameter, width, radius of a minimum enclosing ball, volume of the smallest bounding box, and the cost of ℓ-center clustering ($\ell \geq 2$) for a set of points using coresets. All data structures presented in this paper answer queries in polylogarithmic time and use subquadratic space.

Keywords: coreset, intersection decision problem, maximal point, maximal layer, window query

1. Introduction

We construct data structures for various *geometric objects* (e.g., points, line segments, triangles and convex c-gons) to efficiently answer *window queries*. In a *window query* we are given two positive integers i and j, with $i < j$, such that the interval $[i, j]$ represents a query window of width $\mathcal{W} = j - i + 1$. Let $S = (s_1, s_2, \ldots, s_n)$ be a sequence of n geometric objects. For $1 \leq i < j \leq n$, let $S_{i,j}$ denote the subsequence $(s_i, s_{i+1}, \ldots, s_j)$. We want to preprocess S into

[⋆]This research work was supported by NSERC Research Grants and Ontario Graduate Scholarship. The conference version of this paper was published in CALDAM 2018 [1].

[*]Corresponding author

Email address: `farah.chanchary@carleton.ca` (Farah Chanchary) some data structures such that given a query interval $q = [i, j]$ and a predicate $\mathcal{P}$, we can answer window queries using the objects in $S_{i,j}$ that match $\mathcal{P}$.

Recently the same model of data structure has been considered in various studies (see [2],[3],[4],[5],[6],[7]), where the authors mapped a sequence of geometric objects (or graph edges) to a sequence of *timestamped events*, where for each k, with $1 \leq k \leq n$, an object s_k has a unique timestamp k.

In this paper, we present new results for windowed intersection decision problems and a variety of windowed reporting problems using points on *maximal layers*. Let P be a set of n points in $\mathbb{R}^2$. The first maximal layer L_1 of P is defined to be the maximal points under the dominance relation where a point p is said to be dominated by a point p' if $p[x] \leq p'[x]$ and $p[y] \leq p'[y]$, and $p \neq p'$. For $\gamma > 1$, the γ-th maximal layer L_γ is the set of maximal points in $P - \bigcup_{l=1}^{\gamma-1} L_l$ [8]. We define the *windowed intersection decision problem* as 'Given a pair of indices (i, j), where $1 \leq i < j \leq n$, report whether there is any intersection between objects in $(s_i, \ldots, s_j)$'. In [5], Chan and Pratt presented orthogonal segment intersection decision problems, whereas our algorithms can preprocess sequences of objects, i.e., line segments, triangles, and convex c-gons, with arbitrary orientations. For our second set of problems, we consider a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^d$, where $d \geq 2$, and we answer queries related to maximal layers and dominance. More specifically we solve three types of windowed queries of the following forms: Given a query interval $[i, j]$ (i) count the number of maximal points in $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$, (ii) given an integer k, with $i \leq k \leq j$, decide if point p_k is on the maximal layer L_γ of $P_{i,j}$, where $\gamma = 1$, or 2, or ≥ 3, and (iii) for a fixed integer $k \geq 1$, (a) report all points in the query interval that are dominated by at least k points of $P_{i,j}$ (i.e., *k-dominated points*) and (b) report all maximal points in $P_{i,j}$ such that each point dominates at least k points of $P_{i,j}$ (i.e., *k-dominant points*). Lastly we show techniques to approximate window-aggregate queries using *decomposable coresets* of size $f_d(\epsilon)$ to compute $(1 + \epsilon)$-approximations for various geometric problems such as the diameter, width, radius of a minimum enclosing ball, volume of the smallest bounding box and ℓ-center clustering of a queried point set $P_{i,j}$. Coresets that can be computed based on the *extent measures* of a point set are called decomposable coresets, and the extent measure of a point set P with respect to a point $x \in \mathbb{R}^d$ is defined as $w(P, x) = \max_{p,q \in P}(p - q) \cdot x$ [9].

1.1. Previous Work

Bannister et al. [3] were the first to consider this window model for preprocessing timestamped graph edges into data structures that can answer windowed queries. Subsequently more results on windowed graph problems were presented in [6] and [7]. Similar time window model for geometric objects was first studied in [2], where the authors presented results for reporting the convex hull of points in the plane, and skyline and proximity relations of point sets in $\mathbb{R}^d$. They used a hierarchical decomposition in time to construct binary decomposition trees on a given set of temporal points to answer windowed queries related to convex hull and proximity relations. The authors solve various problems related to the convex hull in polylogarithmic time and approximations for the nearest neighbour queries and the construction of proximity graphs. However, their skyline queries use a different preprocessing technique based on the rectangle stabbing data structures. Mouratidis et al. [10] considered problems of monitoring top-k maximal layers (mentioned as *k-skyband* in [10]) using fixed-width sliding query windows. Our results for points on maximal layers presented in this paper consider variable-width query windows and are different from those of [2] and [10].

Subsequently more results have been presented by Bokal et al. [4], and Chan and Pratt [5] on window queries. They mainly focused on answering decision problems on hereditary properties, such as the convex hull area decision problem (in 2D), the diameter decision problem (in 2D and 3D), the width decision problem (in 2D) and the orthogonal segment intersection detection problem. Bokal et al. [4] showed a sketch-based general methodology for finding all maximal subsequences for a set of n points in plane, i.e., for all i, with $1 \leq i \leq n$, they find the largest index of the maximal interval starting at i that holds some hereditary property $\mathcal{P}$.

The authors solved problems for finding all maximal subsequences with unit diameter, all maximal subsequences whose convex hull area is at most 1 and all maximal subsequences that define monotone paths in some (subpath-dependent) direction. Later, Chan and Pratt [5] improved preprocessing times for diameter decision problems and convex hull area decision problems. The authors presented techniques to solve the diameter decision problem in 2D and in 3D, and the orthogonal line segment intersection detection problems by reducing the windowed decision problems into range successor problems. As the second approach, the authors used dynamic data structures and a first-in-first-out sequence of processing geometric objects to find all maximal subsequences of intervals that satisfies some property $\mathcal{P}$. Authors named this process *FIFO updates* and used this technique to solve the 2D convex hull area decision problem and the 2D width decision problem. Table 1 summarizes previous results of window queries for geometric problems.

1.2. New Results

The main contributions of this paper are listed below, and are also summarized in Table 2.

1. *Intersection decision problems*: Given a sequence S of n geometric objects, we can preprocess S for the windowed intersection decision problem in $O(n^{4/3} \cdot \text{polylog}(n))$ time using $O(n^{4/3} \cdot \text{polylog}(n))$ space so that queries can be answered in $O(\log n)$ time.
2. *Problems on points on maximal layers*: Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^d$ we can preprocess P into data structures to report the following.
    - Given a query interval $[i, j]$ and a point p_k with $i \leq k \leq j$, we can report whether p_k is on the maximal layer of the sequence of points $P_{i,j} = (p_i, \ldots, p_j)$ in $O(1)$ time. Preprocessing takes $O(n \log^{d-1} n)$ time using $O(n \log^{d-2} n)$ space .
    - Given a query interval $[i, j]$ and a point p_k with $i \leq k \leq j$, we can report whether p_k is on layer 2 or ≥ 3 of the sequence $P_{i,j}$

Table 1: Summary of previous results on window queries for geometric problems. Here n is the number of input objects, h is the size of the convex hull, w is the size of the output, $\mathcal{W}$ is the size of the query window and α is the inverse Ackermann function.

Problems	**Preprocess time**	**Query time**	**Ref.**
Convex hull reporting	$O(n \log n)$	$O(h \log^2 \mathcal{W})$	[2]
Gift wrapping, line stabbing, tangent queries	$O(n \log n)$	$O(\log^2 \mathcal{W})$	[2]
Linear prog., line decision queries	$O(n \log n)$	$O(\log \mathcal{W})$	[2]
Skyline reporting	$O(n^{1+\epsilon})$	$O(w)$	[2]
Spherical range reporting	$O(n \log n)$	$O(\log \mathcal{W} + w)$	[2]
Approx. nearest neighbor	$O(n \log n)$	$O(\log \mathcal{W})$	[2]
Diameter decision problem (2D)	$O(n \log^2 n)$	$O(1)$	[4]
	$O(n \log n)$	$O(1)$	[5]
Diameter decision problem (3D)	$O(n \log^2 n)$	$O(1)$	[5]
Convex hull area decision problems	$O(n \log n \log \log n)$	$O(1)$	[4]
	$O(n\alpha(n) \log n)$	$O(1)$	[5]
Monotone paths	$O(n)$	$O(1)$	[4]
Orthogonal segment intersection detection	$O(n \log n \log \log n)$	$O(1)$	[5]
Width decision problem (2D)	$O(n \log^8 n)$	$O(1)$	[5]

in $O(\log^{d+1} n)$ time. Preprocessing takes $O(n \log^{d+1} n)$ time using $O(n \log^{d+1} n)$ space.

- We can count the total number of maximal points of $P_{i,j}$ in $O(\log^2 n)$ time. Preprocessing takes $O(n \log^{d-1} n)$ time using $O(n \log^{d-2} n)$ space when $d \geq 4$. However, for $d \in \{2, 3\}$ preprocessing takes $O(n \log^2 n)$ time and $O(n \log^2 n)$ space.
- Given a fixed integer k, we can report all points in $P_{i,j}$ that are dominated by at least k points (i.e., *k-dominated points*) in $O(\log^2 n + kw)$ time, where w is the size of the output. Preprocessing takes

$O(kn \log^2 n)$ time using $O(kn \log n)$ space when $2 \leq d \leq 3$. For $d \geq 4$, preprocessing takes $O(kn \log^{d-1} n)$ time and $O(kn \log^{d-2} n)$ space.

- Given a fixed integer k, we can report all maximal points in $P_{i,j}$ each dominating at least k points (i.e., *k-dominant points*) in $O(\log^4 n + kw)$ time, where w is the size of the output. Preprocessing takes $O(kn \log^4 n)$ time using $O(kn \log^3 n)$ space when $2 \leq d \leq 5$. For $d \geq 6$, preprocessing takes $O(kn \log^{d-1} n)$ time and $O(kn \log^{d-2} n)$ space.

3. *Approximations using coresets:*

- Let P be a sequence of n points in $\mathbb{R}^d$, where d is a fixed dimension. The sequence P can be preprocessed into a data structure of size $O(n \log n)$ such that for a query interval $[i, j]$, with $1 \leq i < j \leq n$, we can compute a $(1+\epsilon)$-coreset of size $f_d(\epsilon)$ in $O(f_d(\epsilon) \log n)$ time for geometric problems that admit some decomposable coresets. Here $f_d(\epsilon)$ is the smallest integer such that for any real number $\epsilon > 0$ the ϵ-coreset of P has size at most $f_d(\epsilon)$.
- Let P be a sequence of n points in $\mathbb{R}^2$ that can be preprocessed into a data structure of size $O(n \log n)$ such that given a query window $[i, j]$ and two parameters $\ell \geq 2$ and $\epsilon > 0$, a coreset of size $O(\ell(f_2(\ell)/(c\epsilon))^2)$ for the ℓ-center clustering can be computed in $O(\ell((f_2(\ell)/(c\epsilon)) \log n) + \ell(f_2(\ell)/(c\epsilon))^2 + \mathcal{W})$ time, where $\mathcal{W}$ is the width of the query window and c is a positive constant.

1.3. Organization

This paper is organized as follows. In Section 2, we present algorithms for windowed intersection decision problems using various geometric objects such as segments, bichromatic segments, triangles and c-gons. Section 3 presents more results for windowed queries using points on maximal layers. In Section 4 we present two techniques to obtain $(1+\epsilon)$-approximations for geometric problems using coresets. Section 5 concludes this paper.

Table 2: Summary of results. Here n is the number of input objects in dimension $d \geq 2$, $\gamma \geq 2$ is the number of maximal layer, w is the output size, k is a fixed parameter, $\mathcal{W} = j - i + 1$ is the width of the query window, $\ell \geq 2$ is an input parameter, $\epsilon > 0$ and $c > 0$ are small constants.

Problems	**Preprocessing time**	**Space**	**Query time**	
Intersection Decision				
Segment, Triangle				Th. 1,
and c-gons	$O(n^{4/3} \operatorname{polylog}(n))$	$O(n)$	$O(\log n)$	Cor. 5- 8
Points on Maximal Layers				
p_k on maximal layer:				
L_1	$O(n \log^{d-1} n)$	$O(n \log^{d-2} n)$	$O(1)$	Th. 8
$L_\gamma, \gamma = 2, \geq 3$	$O(n \log^{d+1} n)$	$O(n \log^{d+1} n)$	$O(\log^{d+1} n)$	Th. 12
Count maximal points:				
$d = 2, 3$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(\log^2 n)$	Th. 9
$d \geq 4$	$O(n \log^{d-1} n)$	$O(n \log^{d-2} n)$	$O(\log^2 n)$	Th. 9
k-dominated points:				
$2 \leq d \leq 3$	$O(kn \log^2 n)$	$O(kn \log n)$	$O(\log^2 n + kw)$	Th. 10
$d \geq 4$	$O(kn \log^{d-1} n)$	$O(kn \log^{d-2} n)$	$O(\log^2 n + kw)$	Th. 10
k-dominant points:				
$2 \leq d \leq 5$	$O(kn \log^4 n)$	$O(kn \log^3 n)$	$O(\log^4 n + kw)$	Th. 11
$d \geq 6$	$O(kn \log^{d-1} n)$	$O(kn \log^{d-2} n)$	$O(\log^4 n + kw)$	Th. 11
Approximation using Coresets				
Decomposition	$O(n \log n)$	$O(n \log n)$	$O(f_d(\epsilon) \log n)$	Th. 13
ℓ-clustering cost	$O(n \log n)$	$O(n \log n)$	$O(\ell((f_2(\ell)/(c\epsilon)) \log n)$	Th. 14
			$+\ell(f_2(\ell)/(c\epsilon))^2 + \mathcal{W})$	

2. Geometric Object Intersections

In this section, we discuss *windowed intersection decision problems* on a given sequence of geometric objects (e.g., line segments, triangles, and constant-size polygons) within a query interval $[i, j]$. Input consists of a sequence of n geometric objects $S = (s_1, s_2, \ldots, s_n)$. We will represent the sequence S in an array A, where $A[i] = s_i$, for $i = 1, 2, \ldots, n$. The *windowed intersection decision problem* is 'Given a pair of indices (i, j), where $1 \leq i < j \leq n$, report whether there is any intersection between objects in $(s_i, \ldots, s_j)$'.

2.1. Preliminaries

To solve windowed intersection decision problem of geometric objects given in $\mathbb{R}^d$, we use the following structures described in [11].

Corollary 1. *[11, Corollary 7.3(i)] Given n points in $\mathbb{R}^d$, we can form $O(n)$ canonical subsets of total size $O(n \log n)$ in $O(n \log n)$ time, such that the subset of all points inside any query simplex can be reported as a union of disjoint canonical subsets C_i with $\sum_i |C_i|^{1-1/d} \leq O(n^{1-1/d} \log n)$ in time $O(n^{1-1/d} \log n)$ with high probability with respect to n.*

Corollary 2. *[11, Corollary 7.5] Given n simplices in R^d, there is a data structure with $O(n \log^{d+1} n)$ preprocessing time and $O(n \log^d n)$ space, such that we can find all simplices containing a query point in $O(n^{1-1/d} \log^d n)$ expected time; and we can find all simplices contained inside a query simplex in $O(n^{1-1/d} \log^d n)$ expected time.*

Corollary 3. *[11, Corollary 7.7(i)] Suppose there is a d-dimensional halfspace range counting data structure for point sets of size at most B with $P(B)$ preprocessing time, $S(B)$ space, and $Q(B)$ (expected) query time. Then there is a d-dimensional halfspace range counting data structure for point sets of size at most n with $O(n/B)P(B) + O(n \log n)$ preprocessing time, $O(n/B)S(B) + O(n)$ space, and $O(n/B)^{1-1/d}Q(B) + O(n/B)^{1-1/d}$ expected query time, assuming $B < n/\log^{\omega(1)} n$.*

Corollary 4. *[11, Corollary 7.8]] There is a d-dimensional halfspace range counting data structure with $O(m2^{O(\log^* n)})$ preprocessing time and $O((n/m^{1/d})2^{O(\log^* n)})$ expected query time for any given $m \in [n \log n, n^d / \log^d n]$.*

2.2. Overview of Our Data Structure

Before we discuss our data structure, we define a *valid pair* of indices (α, β) with $1 \le \alpha < \beta \le n$ as follows: For each $1 \le \alpha \le n$, let β be the smallest index larger than α such that the object $A[\beta]$ intersects $A[\alpha]$. If there is no $A[\beta]$ that intersects $A[\alpha]$ then there is no valid pair (α, β). See Figure 1 for an illustration.

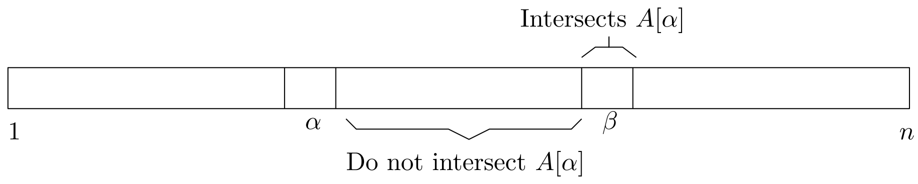

Figure 1: A valid pair (α, β).

Suppose, there exists a data structure that can find all valid pairs (α, β). Then we can reduce the windowed intersection decision problem into a range query problem as follows. For each valid pair (α, β), we store a point $(\alpha, \beta) \in \mathbb{R}^2$ using a priority search tree (PST) data structure [12]. A PST takes linear space to store $O(n)$ points in the plane and it can be built in $O(n \log n)$ time. For a given query interval $[i, j]$, we perform a range search in PST with the query rectangle $R_q = [i, \infty) \times (-\infty, j]$. Note that, there will be an intersecting pair of objects $(A[\alpha], A[\beta])$ in query interval $[i, j]$ if and only if there is a point $(\alpha, \beta) \in R_q$. Hence, if the range searching query returns a positive count of points in R_q, then we report that some objects intersect in the interval $[i, j]$. This query can be answered in $O(\log n)$ time. Thus we obtain the following lemma.

Lemma 1. *Suppose a sequence of n geometric objects is stored in an array $A[1..n]$, where i is the index of the object stored in $A[i]$. Given all valid pairs (α, β) for every $1 \le \alpha \le n$ in A, we can build a data structure of size $O(n)$ that can answer windowed intersection decision queries in $O(\log n)$ time.*

Next we show how to find all the valid pairs. Let X be a set of n geometric objects. We assume that we have a data structure $DS(X)$ that tells us whether a query object q intersects any member of X. Furthermore, $DS(X)$ takes $M(n)$ space, $P(n)$ preprocessing time and $Q(n)$ query time, where $M(n)/n$, $P(n)/n$ and $Q(n)$ are all non-decreasing functions. To find all the valid pairs, we maintain a tree T defined as follows. The leaves of T store objects $A[1], A[2], \ldots, A[n]$ in order from left to right. For each internal node v of T, let $P[v]$ be the set of objects at the leaves of the subtree rooted at v. Each node v of T stores all the objects in its subtree in a secondary data structure $DS[P[v]]$. Each level i of T has 2^i nodes. So each node at level i requires $M(n/2^i)$ space. The total space requirement is

$$\begin{aligned}
O(n) + \sum_{i=0}^{\log n} 2^i \cdot M(n/2^i) &= O(n) + \sum_{i=0}^{\log n} n \cdot \frac{M(n/2^i)}{n/2^i} \\
&\leq O(n) + \sum_{i=0}^{\log n} n \cdot \frac{M(n)}{n} \\
&\quad \text{[since } \frac{M(n/2^i)}{n/2^i} \text{ is non-decreasing]} \\
&\leq O(n) + \sum_{i=0}^{\log n} M(n) \\
&= O(M(n) \cdot \log(n)).
\end{aligned}$$

The total preprocessing time can analogously be computed as $O(P(n) \log n)$. Now for any $1 \leq \alpha \leq n$, we can find β in time $O(Q(n) \log n)$ as follows. To identify a valid pair (α, β), we first search T to find the leaf v' containing α. It requires $O(\log n)$ time using a standard binary search. Then we move up from v' towards the root node and at every step perform the following search. Each time we move from a child node v' towards its parent node $p(v')$, we query the secondary structure stored at the right child of $p(v')$ to decide whether it contains an object β that intersects with α. If the search is unsuccessful, we move upwards one more level in T, and repeat the process. Otherwise, we find the node that contains the intersecting object and we continue descending from

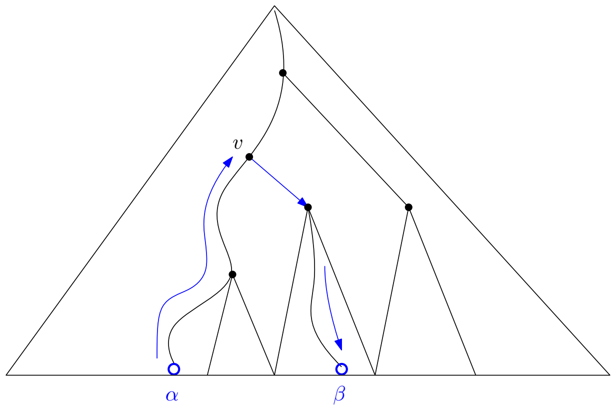

Figure 2: Query path from α to β in our multilevel data structure.

$p(v')$ to locate the leaf node containing β (see Figure 2). In this way, the total time required to find all valid pairs is $O(n \cdot Q(n) \log n)$. From Lemma 1 we obtain the following result.

Theorem 1. *Given a sequence S of n geometric objects, we can preprocess S into a data structure of size $O(n)$ in time $O(P(n) \log n + n \cdot Q(n) \log n)$ so that it can answer windowed intersection decision queries in $O(\log n)$ time. The space used during preprocessing is $O(M(n) \log n)$.*

Next, we discuss the construction of the secondary data structure $DS(X)$ for different problems.

Segment Intersections: Given a sequence of n line segments $S = (s_1, s_2, \ldots, s_n)$ in the plane, we want to preprocess S to answer windowed queries for segment intersections. As we have described previously, our primary data structure T stores n input segments at the leaf nodes sorted in order from left to right. At each node v of T we build a multi-level partition tree that answers queries of the form *'Given a query segment s_q, does s_q intersect any segment of $\{s_a, s_{a+1}, \ldots, s_b\}$, where $1 \leq a < b \leq n$?'*. For a sequence of n line segments in the plane, we can obtain a data structure with $O(n \log^3 n)$ preprocessing time and $O(n \log^2 n)$ space such that we can report whether a query line segment s_q

intersects any input segment in $O(\sqrt{n}\log^2 n)$ expected time by applying Corollary 7.3(i) in [11] three times, where $d = 2$. Finally, by repeated applications of Corollary 7.3(i) and Corollary 7.8 in [11] with $d = 2$, we can build a data structure to answer the above mentioned queries. This requires preprocessing time $P(n) = O(m \cdot \text{polylog}(n))$ and query time $Q(n) = O(n/\sqrt{m} \cdot \text{polylog}(n))$, where $m = n^{4/3}$. So by Corollaries 1- 4, the total time required for segment intersection preprocessing is $O(n^{4/3} \cdot \text{polylog}(n) + n \cdot n^{1/3} \cdot \text{polylog}(n)) = O(n^{4/3} \cdot \text{polylog}(n))$.

Corollary 5. *Given a sequence S of n segments, we can preprocess S for the windowed segment intersection decision problem in $O(n^{4/3} \cdot \text{polylog}(n))$ time using $O(n^{4/3} \cdot \text{polylog}(n))$ space.*

Bichromatic Segment Intersections: Let $S = (B \cup R)$ be a sequence of bichromatic line segments, where B is a sequence of b pairwise disjoint *blue* segments, R is a sequence of r pairwise disjoint *red* segments, and $N = b + r$. Our data structure for segment intersection problem can be extended for reporting windowed bichromatic segment intersection problem (intersections of red segments with blue segments) using the same preprocessing time and space bound. We assume that every segment in S has a unique timestamp. We build two sets of the same data structure we presented in this section. Let T_B be one structure where we store b blue segments, make queries with r red segments, and find *valid pairs* $(s_{r'}, s_{b'})$, where a red segment $s_{r'}$ intersects with a blue segment $s_{b'}$. T_R is the analogous structure that gives us all *valid pairs* $(s_{b'}, s_{r'})$. The only minor change occurs when searching the primary data structures with a query segment. For example, when we query T_B with any red segment s_r, first we have to find the leaf node containing a blue segment with the smallest timestamp such that $t(s_b) > t(s_r)$. The rest of the search algorithm remains unchanged.

Corollary 6. *Given a sequence $S = (B \cup R)$ of N bichromatic line segments, where B is a sequence of b pairwise disjoint blue segments, R is a sequence of r pairwise disjoint red segments, and $N = b + r$. We can preprocess S for* *the windowed bichromatic segment intersection decision problem in* $O(N^{4/3} \cdot \text{polylog}(N))$ *time using space* $O(N^{4/3} \cdot \text{polylog}(N))$.

Triangle Intersections: The input for this problem is a sequence of n triangles $T = (t_1, t_2, \ldots, t_n)$ and we want to preprocess them to answer queries for windowed triangle intersections. First, we categorize all possible orientations of triangle intersections. Figure 3 illustrates three orientations of a query triangle t_q that intersects with a triangle t_i. We describe inputs and query types for each of the cases. Note that, these cases are not mutually exclusive. However, our data structure returns true if at least one of these intersections is present.

Case (a): A sequence of triangles $()t_1, t_2, \ldots, t_n)$ is stored and we ask the query: Given a point p, is p contained in some triangle t_i?
Case (b): A sequence of points $(p_1, p_2, \ldots, p_n)$ (one vertex of each triangle in $(t_1, t_2, \ldots, t_n)$) is stored and we ask the query: Given a triangle t, does t contain some point p_i?
Case (c): A sequence of triangles $(t_1, t_2, \ldots, t_n)$ is stored and we ask the query: Given a triangle t, does t overlap some triangle t_i?

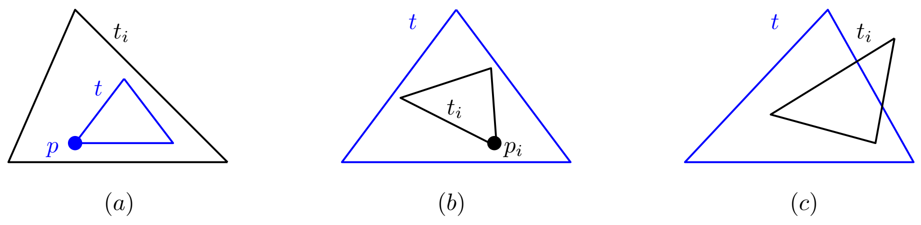

Figure 3: Three possible orientations of intersections of a query triangle t_q (blue) with some triangle t_i (black).

For cases (a) and (c), by repeated applications of Corollary 7.3(i) and Corollary 7.5 in [11] we can build a data structure that can answer such queries with preprocessing time $O(m \cdot \text{polylog}(n))$ and query time $O(n/\sqrt{m} \cdot \text{polylog}(n))$, where $m = n^{4/3}$. For case (b), we build a data structure by applying Corollary

7.5 and Corollary 7.7(i) in [11], which requires the same preprocessing and query time as mentioned for the previous two cases. Finally, we put together all cases in a single data structure $DS(T)$ that we use as the secondary data structure stored at each node of our main search tree.

Corollary 7. *Given a sequence T of n triangles, we can preprocess T for the windowed triangle intersection decision problem in $O(n^{4/3} \cdot \text{polylog}(n))$ time using $O(n^{4/3} \cdot \text{polylog}(n))$ space.*

We observe that our data structure for the windowed triangle intersection problem can be extended to any convex polygon with c sides, where c is a constant. Solving the *windowed c-gon intersection decision problem* will add some extra levels to our structure, and thus the preprocessing time increaseas by a polylogarithmic factor. Hence we obtain the following result.

Corollary 8. *For some constant c, given a sequence S of n convex c-gons (polygons with c sides), we can preprocess S for the windowed c-gon intersection decision problem in $O(n^{4/3} \cdot \text{polylog}(n))$ time using $O(n^{4/3} \cdot \text{polylog}(n))$ space.*

3. Points on Maximal Layers

Initially we present results for windowed queries on points on maximal layers of a given sequence of points in $\mathbb{R}^2$. The generalized solutions for all problems in $\mathbb{R}^d$, where $d \geq 2$, are presented in Section 3.6.

Definition 1. *Let P be a set of n points in $\mathbb{R}^2$. The first maximal layer L_1 of P is defined to be the maximal points under the dominance relation where a point p is said to be dominated by a point p' if $p[x] \leq p'[x]$ and $p[y] \leq p'[y]$, and $p \neq p'$. For $\gamma > 1$, the γ-th maximal layer L_γ is the set of maximal points in $P - \bigcup_{l=1}^{\gamma-1} L_l$ [8].*

Definition 2. *For a point $q = (q_1, q_2) \in \mathbb{R}^2$, we define $NE(q)$ to be the set of points in $\mathbb{R}^2$ that lie in the North-East quadrant of q, i.e., $NE(q) = \{(a, b) \in \mathbb{R}^2 : a > q_1$ and $b > q_2\}$ and $SW(q)$ to be the set of points in $\mathbb{R}^2$ that lie in the*

South-West quadrant of q, i.e., $SW(q) = \{(c, d) \in \mathbb{R}^2 : c < q_1$ *and* $d < q_2\}$ *(see Figure 4).*

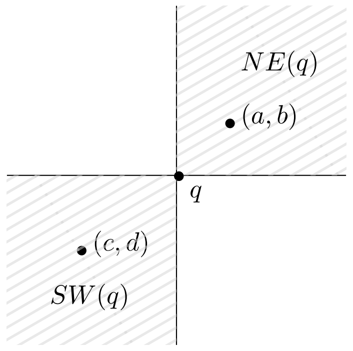

Figure 4: A point $(a, b) \in NE(q)$ and a point $(c, d) \in SW(q)$.

3.1. Is p_k on the Maximal Layer L_1?

Suppose $P = (p_1, p_2, \ldots, p_n)$ is a sequence of n points in $\mathbb{R}^2$. We want to preprocess P into a data structure such that given a query interval $[i, j]$ and an integer k with $i \leq k \leq j$ we can report whether the point p_k is on the maximal layer L_1 of $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$. We assume that no two points have the same x-coordinate or the same y-coordinate. Let $p_0 = (\infty, \infty)$ and $p_{n+1} = (\infty, \infty)$ be two new points added to P. Let $A[0..n+1]$ be an array, where $A[i] = p_i$ for all $0 \leq i \leq n+1$. For any k with $1 \leq k \leq n$, we define the following two functions: $\alpha(k) = \min\{i\colon i > k \text{ and } p_i \in NE(p_k)\}$ and $\beta(k) = \max\{i\colon i < k \text{ and } p_i \in NE(p_k)\}$

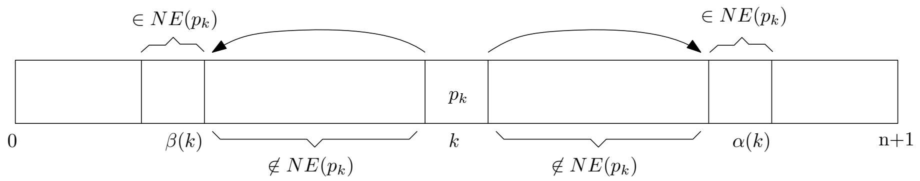

Figure 5: $A[\alpha(k)]$ and $A[\beta(k)]$ of a point p_k in array $A[0..n+1]$.

A point p_k is on the maximal layer L_1 of a sequence of points $P_{i,j} =$

$(p_i, p_{i+1}, \ldots, p_j)$ if none of these points dominates p_k. We have the following lemma.

Lemma 2. *Suppose $1 \leq i \leq k \leq j \leq n$. The point p_k is on the maximal layer L_1 of $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$ if and only if $\alpha(k) > j$ and $\beta(k) < i$.*

Suppose that we have a data structure that computes $\alpha(k)$ and $\beta(k)$ for each $p_k \in P$. Now we augment array A such that every element $A[k]$ stores two pointers pointing to $A[\alpha(k)]$ and $A[\beta(k)]$, respectively. This data structure requires $O(n)$ space. Now according to Lemma 2, we can answer the query whether a given point p_k is on the maximal layer of $P_{i,j}$ with $i \leq k \leq j$ in $O(1)$ time by checking $A[\alpha(k)]$ and $A[\beta(k)]$.

Lemma 3. *Suppose a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$ is given and there exists a data structure that computes $\alpha(k)$ and $\beta(k)$ for each $p_k \in P$ using $S(n)$ space and $T(n)$ time. Then P can be preprocessed into a data structure of size $O(n)$ in $O(T(n))$ time such that given a query interval $[i, j]$ and a point p_k with $i \leq k \leq j$, we can decide whether p_k is on the maximal layer L_1 of $P_{i,j}$ in $O(1)$ time.*

Data structure for computing α and β for points in $\mathbb{R}^2$: Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in the plane, we want to build a data structure to compute $\alpha(k)$ and $\beta(k)$ for $1 \leq k \leq n$. We present the technique for computing $\alpha(k)$ here (see Algorithm 1). We initialize an empty priority search tree (PST) T. For $1 \leq i \leq n$, we query T with $q = (-\infty, p_{i,x}] \times (-\infty, p_{i,y}]$, where $p_{i,x}$ and $p_{i,y}$ are respectively the x-coordinate and the y-coordinate of point p_i. This yields points that appear before p_i in the sequence and are dominated by p_i. Let this set of points be S_i. According to the definition of α, i becomes the α value for all these points. For each $p_k \in S$ we set $\alpha(k) = i$ and delete p_k from T. Now we insert p_i into T. More specifically, we maintain the following invariant.

- For each k with $1 \leq k \leq i$: if p_k is in T, then $\alpha(k) \geq i$. If p_k is not in T, then $\alpha(k) < i$ and $\alpha(k)$ has been determined.

This data structure requires $O(n)$ space and $O(n \log n)$ time to set $\alpha(k)$ for all $p_k \in P$, where $1 \leq k \leq n$. Similarly we can compute $\beta(k)$ for all p_k by reversing their order of insertion into T.

Algorithm 1: SetAlpha(P)

```
  Input : A sequence of $n$ points $P = (p_1, p_2, \ldots, p_n) \in \mathbb{R}^2$.
1 Initialize an empty PST $T$.
2 for $i = 1$ to $n$ do
3     $S_i = T$.Query($(-\infty, p_{i,x}] \times (-\infty, p_{i,y}]$).
4     foreach $p_k \in S_i$ do
5         Set $\alpha(k) = i$.
6         $T$.Delete($p_k$).
7     $T$.Insert($p_i$).
```

Lemma 4. *Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$, we can compute the values of $\alpha(k)$ and $\beta(k)$ for all $p_k \in P$, in $O(n \log n)$ total time using $O(n)$ space.*

Data structure for computing α and β for points in $\mathbb{R}^d$: We build a range tree for n points on their first $d-2$ coordinates. At each canonical node v of the last level of the range tree we add a PST that is built on the last two coordinates of the subset of points stored at v. So this structure can be built in $O(n \log^{d-1} n)$ time using $O(n \log^{d-2} n)$ space. Thus for the general case, where $d \geq 2$, we obtain the following result.

Theorem 2. *Given a sequence of n points, we can compute the values of $\alpha(k)$ and $\beta(k)$ for $k \in \{1, 2, \ldots, n\}$, in $O(n \log^{d-1} n)$ total time using $O(n \log^{d-2} n)$ space.*

Remark: Bannister et al. [2] used a dynamic data structure for dominance queries by Mortensen [13] to compute α and β values for all points in $\mathbb{R}^d$. Our data structure for computing all α and β values is faster by a factor of $O(\log n)$ and uses less space by a factor of $O(\log^2 n)$.

Finally from Lemmas 3 and 4 we obtain the following theorem for points in $\mathbb{R}^2$.

Theorem 3. *A sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O(n)$ in $O(n \log n)$ time such that given a query interval $[i, j]$ and a point p_k with $i \le k \le j$, we can report whether p_k is on the maximal layer L_1 of points $P_{i,j}$ in $O(1)$ time.*

3.2. Count Points on Maximal Layer L_1

Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$, and a query interval $[i, j]$ with $1 \le i \le j \le n$, we want to count the total number of points on the maximal layer L_1 of $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$.

Following Lemma 2, we transform each point $p_k = (p_{k,x}, p_{k,y}) \in P$ into a point $p'_k = (k, \alpha(k), \beta(k)) \in \mathbb{R}^3$ for $1 \le k \le n$. Now we have a set of n points in $\mathbb{R}^3$. We can compute all $\alpha(k)$ and $\beta(k)$ in $O(n \log n)$ time by Lemma 4. We build a standard 3-dimensional range tree [14], where the first level of the tree is based on the time of the points. At the second level of the tree, for each canonical node we build a range tree using the second $(\alpha(k))$ and the third coordinates $(\beta(k))$ of each point p'_k. The total space requirement is $O(n \log^2 n)$ and this data structure can be built in $O(n \log^2 n)$ time [14].

We transform a given query interval $[i, j]$ into a query box $[i, j] \times [j+1, +\infty) \times (-\infty, i-1]$. The first level of the range tree is queried using interval $[i, j]$. This requires $O(\log n)$ query time. For each canonical subset in the second level, we query using $[j+i, +\infty) \times (-\infty, i-1]$. This step requires $O(\log n)$ time for each canonical node on the search path. Thus, given any query interval $q = [i, j]$ we can report the total number of maximal points in $P_{i,j}$ in $O(\log^2 n)$ time.

Theorem 4. *A sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O(n \log^2 n)$ in $O(n \log^2 n)$ time such that* *given a query interval $[i, j]$ we can report the total number of maximal points of $P_{i,j}$ in $O(\log^2 n)$ time.*

3.3. Is p_k on Maximal Layer L_γ, where $\gamma = 2$ or $\gamma \geq 3$?

Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$, a query interval $[i, j]$ and an integer k with $1 \leq i \leq k \leq j \leq n$, we want to report if point p_k is on maximal layer L_γ, where $\gamma = 2$ or $\gamma \geq 3$ of $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$. First we solve the problem for $\gamma \geq 3$ and then show that the result for $\gamma = 2$ follows.

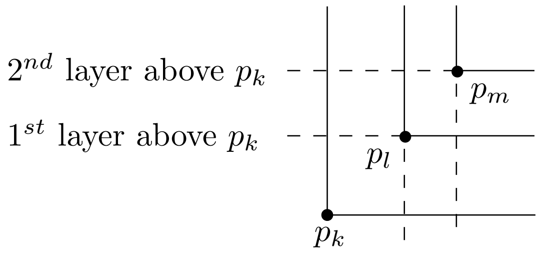

Figure 6: Point p_k on some maximal layer ≥ 3 iff $p_l \in NE(p_k)$ and $p_m \in NE(p_l)$.

Lemma 5. *Recall the definitions of α and β from Section 3.1. Let $1 \leq i \leq k \leq j \leq n$. The point p_k is on layer L_γ, where $\gamma \geq 3$ of $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$ if and only if at least one of the following is true.*

1. *There exists some l such that $l \geq i$, $p_l \in NE(p_k)$, and $\alpha(l) \leq j$*
2. *There exists some l such that $l \leq j$, $p_l \in NE(p_k)$, and $\beta(l) \geq i$.*

Proof 1. *The 'if' part is obvious from the definitions of $\alpha(l)$ and $\beta(l)$, and by Lemma 2. To prove the converse, we assume p_k to be a point on some maximal layer L_γ, where $\gamma \geq 3$, of points in $P_{i,j}$. Then there must exist some l and m such that $i \leq l \leq j$, $i \leq m \leq j$, $p_l \in NE(p_k)$ and $p_m \in NE(p_l)$ (see Figure 6). Now, point p_m can come at one of two positions with respect to p_l.*

Case 1*: Suppose p_m comes after p_l, i.e., $m > l$. Since $\alpha(l) \leq m$ by the definition of $\alpha(l)$, we obtain $\alpha(l) \leq m \leq j$.*

Case 2*: Suppose p_m comes before p_l, i.e., $m < l$. By the similar argument since $\beta(l) \geq m$ by the definition of $\beta(l)$, we obtain $\beta(l) \geq m \geq j$.* □

We map each point $p_l = (p_{l,x}, p_{l,y}) \in P$, where $1 \le l \le n$, to a point in $\mathbb{R}^4$ as follows. For part (1) of Lemma 5, we define a function $f(l) = (p_{l,x}, p_{l,y}, l, \alpha(l)) \in \mathbb{R}^4$. We set $S = \{f(l) : 1 \le l \le n\}$. Similarly, for part (2) of Lemma 5, we define a function $g(l) = (p_{l,x}, p_{l,y}, l, \beta(l)) \in \mathbb{R}^4$. We set $T = \{g(l) : 1 \le l \le n\}$. For each $1 \le l \le n$, $\alpha(l)$ and $\beta(l)$ can be computed in $O(n \log n)$ time. Now we have two sets S and T each having n points in $\mathbb{R}^4$. We store S and T using two 4-dimensional range trees. A standard 2-dimensional range tree requires $O(n \log n)$ space and can be built in $O(n \log n)$ time. For each additional level the required time and space increase by a logarithmic factor. Therefore our 4-dimensional range tree can be built using $O(n \log^3 n)$ space in $O(n \log^3 n)$ time. Now to answer the query whether some point p_k is on layer ≥ 3 in $P_{i,j}$, we define two functions to map our original query (i, j, k) to equivalent queries in $\mathbb{R}^4$ as follows. For $1 \le i \le k \le j \le n$, let $F(i, j, k)$ be a function such that $F(i, j, k) = [p_{k,x}, \infty) \times [p_{k,y}, \infty) \times [i, \infty) \times (-\infty, j]$. Similarly, let $G(i, j, k)$ be a function such that $G(i, j, k) = [p_{k,x}, \infty) \times [p_{k,y}, \infty) \times (-\infty, j] \times [i, \infty)$. It gives us the following lemma.

Lemma 6.

1. *There exists some l such that $l \ge i$, $p_l \in NE(p_k)$, and $\alpha(l) \le j$ if and only if $F(i, j, k) \cap S \ne \emptyset$.*
2. *There exists some l such that $l \le j$, $p_l \in NE(p_k)$, and $\beta(l) \ge i$ if and only if $G(i, j, k) \cap T \ne \emptyset$.*

Combining Lemma 5 and Lemma 6, the query of the form 'Given $1 \le i \le k \le j \le n$, decide if p_k is on maximal layer $\gamma \ge 3$ of $P_{i,j}$' becomes an equivalent query of the form 'Given a set of points in $\mathbb{R}^4$, count the number of points in the range of 4-dimensional quadrants'. This new query can be answered by querying the data structures on the point sets S and T using 4-dimensional quadrants defined by $F(i, j, k)$ and $G(i, j, k)$, respectively. If at least one of these queries returns some point (i.e., the range count is non-zero) then p_k is on layer $\gamma \ge 3$. Each query takes $O(\log^3 n)$ time. The following theorem summarizes the results.

Theorem 5. *Suppose $P = (p_1, p_2, \ldots, p_n)$ is a sequence of n points in $\mathbb{R}^2$. P can be preprocessed into a data structure of size $O(n \log^3 n)$ in $O(n \log^3 n)$ time such that given a query interval $[i, j]$ and k, where $i \leq k \leq j$, we can answer whether p_k is on some maximal layer L_γ, where $\gamma \geq 3$, of $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$ in $O(\log^3 n)$ time.*

Corollary 9 follows from the results of Theorem 3 and Theorem 5.

Corollary 9. *Suppose $P = (p_1, p_2, \ldots, p_n)$ is a sequence of n points in $\mathbb{R}^2$. P can be preprocessed into a data structure of size $O(n \log^3 n)$ in $O(n \log^3 n)$ time such that given a query interval $[i, j]$ and k, where $i \leq k \leq j$, we can answer whether p_k is on the second maximal layer L_2 of $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$ in $O(\log^3 n)$ time.*

3.4. Report k-dominated Points

In this section we shift our interest to reporting points that are dominated by at least a fixed number of points in the query interval. Note that this problem is different from reporting the maximal layer that a point is on. More specifically a point p can be dominated by k points, where $k \geq 1$ is an integer, and still p can be on some maximal layer L_ϕ, where $\phi \leq k+1$. See Figure 7 for an example.

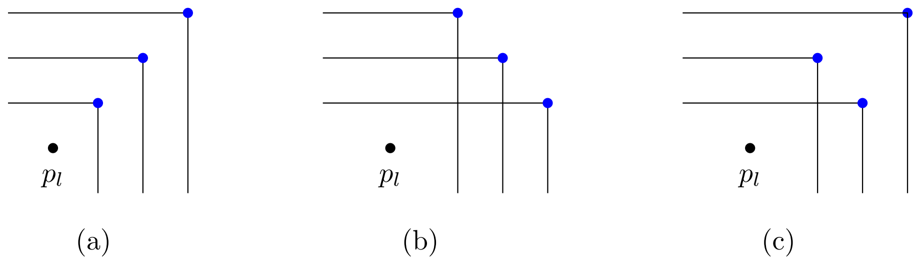

Figure 7: Suppose k=3, then p_l is a k-dominated point in all examples. However, the maximal layer point p_l is on L_4 in (a), on L_2 in (b), and on L_3 in (c).

The problem statement that we investigate here is as follows. Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$, and a fixed integer $k \geq 1$, we want to report all points in $P_{i,j}$ that are dominated by at least k points of $P_{i,j}$. We call these points *k-dominated points.*

For any l with $1 \leq l \leq n$, we define the following. Let $p_{l_1}, p_{l_2}, \ldots, p_{l_k}$ be the first k points that dominate p_l, where $l < l_1 < l_2 < \ldots < l_k$. Similarly, let $p_{l'_k}, \ldots, p_{l'_2}, p_{l'_1}$ be the last k points that dominate p_l, where $l'_k < l'_{k-1} < \ldots < l'_1 < l$ (see Figure 8). Then each interval (l'_{k-a}, l_a) with $0 \leq a \leq k$ represents k points that dominate point p_l. Here $l'_0 = l_0 = l$. There exist at most $k+1$ such intervals for each point in P. We obtain the following lemma.

Lemma 7. *Suppose $1 \leq i < j \leq n$, and k is a fixed integer. A point $p_l \in P_{i,j}$ is dominated by at least k points in $P_{i,j}$ if and only if there exists some a with $0 \leq a \leq k$, such that $i \leq l'_{k-a}$ and $l_a \leq j$.*

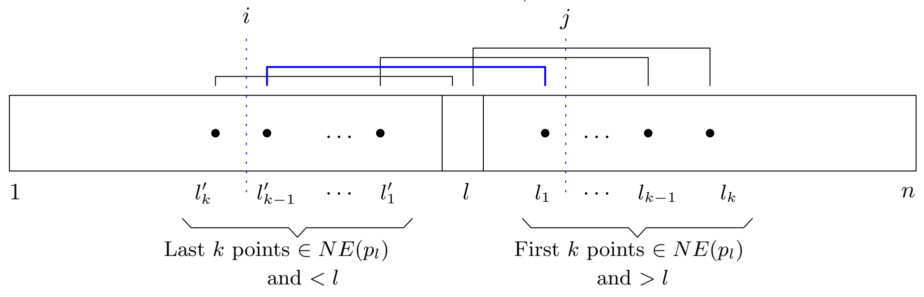
i
j
1
l'_k
l'_{k-1}
$\cdots$
l'_1
l
l_1
$\cdots$
l_{k-1}
l_k
n
Last k points $\in NE(p_l)$ and $< l$
First k points $\in NE(p_l)$ and $> l$

Figure 8: The highlighted interval (l'_{k-1}, l_1) satisfies the query interval $[i, j]$ for k-dominated points.

For each $1 \leq l \leq n$, we map the point p_l to at most $k+1$ points $(l, l_a, l'_{k-a}) \in \mathbb{R}^3$, where $0 \leq a \leq k$. This gives us a set of at most $(k+1)n$ points in total.

We can find the points $p_{l_1}, p_{l_2}, \ldots, p_{l_k}$ and $p_{l'_k}, \ldots, p_{l'_2}, p_{l'_1}$ for each p_l by extending the data structure from Section 3.1 (see Algorithm 1) as follows. Each point p_l now has two additional arrays $B_l[1..k]$ and $A_l[1..k]$ of size k that store k points that dominate p_l and appear, respectively, before and after time l. To achieve this, we replace line 6 of Algorithm 1 by lines 9 – 11 in Algorithm 2. Moreover, we do not delete any points from T (line 7 of Algorithm 1 is omitted). Starting with $i = 1$ we insert point p_i to T and for all points p_l that are dominated by p_i we store p_i in array A_l if it is not full already. Next we repeat inserting points to T in the reverse order (i.e., going from p_n to p_1) and store the first k points that dominate p_l in array B_l. This process takes $O(nk + n \log n)$ time and $O(nk)$ space.

```
Algorithm 2: SetArrayA($P$)
   Input   : A sequence of $n$ points $P = (p_1, p_2, \ldots, p_n) \in \mathbb{R}^2$.
 1 Initialize an empty PST $T$.
 2 for $i = 1$ to $n$ do
 3     Initialize an empty array $A_i[1..k]$.
 4     Set $SizeA[i] = 0$.
 5 for $i = 1$ to $n$ do
 6     $S_i = T$.Query($(-\infty, p_{i,x}] \times (-\infty, p_{i,y}]$).
 7     foreach $p_l \in S_i$ do
 8         if $SizeA[l] < k$ then
 9             Increase $SizeA[l]$ by 1.
10             $A_l[SizeA]$.Store($i$).
11     $T$.Insert($p_i$).
```

We build a range tree on the first coordinate of the $(k+1)n$ points (l, l_a, l'_{k-a}). For each canonical node we add a PST that is built on the last two coordinates of the points stored in that node. This takes $O(kn \log^2 n)$ time and $O(kn \log n)$ space in total. We map our query interval $q = [i, j]$ to $q' = [i, j] \times (-\infty, j] \times [i, +\infty)$. The query takes $O(\log^2 n + kw)$ time, where w is the output size. If we query our data structure with q', each point p_l will be reported at most $k + 1$ times. To report each point exactly once, we build an array $R[1..n]$ initially storing 0 in each $R[l]$, with $1 \leq l \leq n$. Each time a point p_l is reported during query, we first check the value stored in $R[l]$. If $R[l]$ contains 0 then p_l is seen for the first time; we report p_l and set $R[l]$ to 1. If $R[l]$ contains 1 then p_l has already been reported before and we do not report it this time. Reset $R[1 - n]$

to 0 in $O(w)$ time. The entire query takes $O(\log^2 n + kw)$ time, where w is the output size.

Theorem 6. *Suppose $P = (p_1, p_2, \ldots, p_n)$ is a sequence of n points in $\mathbb{R}^2$ and k is a fixed integer. Then P can be preprocessed into a data structure of size $O(kn \log n)$ in $O(kn \log^2 n)$ time such that given a query interval $[i, j]$ we can report all k-dominated points in $P_{i,j}$ in $O(\log^2 n + kw)$ time, where w is the output size.*

3.5. Report k-Dominant Points

Let a *k-dominant point* be a maximal point in a query interval that dominates at least k other points in the same interval. In this section we solve the following problem. Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$, and a fixed constant k, we want to report all k-dominant points in $P_{i,j}$ in the query interval $[i, j]$. The preprocessing technique for this problem is similar to that of k-dominated points. We highlight the key differences here. For any l with $1 \leq l \leq n$, let $p_{l_1}, p_{l_2}, \ldots, p_{l_k}$ be the first k points that are dominated by the point p_l, where $l < l_1 < \ldots < l_k$. Similarly, let $p_{l'_k}, \ldots, p_{l'_2}, p_{l'_1}$ be the last k points that are dominated by p_l, where $l'_k < l'_{k-1} < \ldots < l'_1 < l$ (see Figure 9). As shown before, where each interval (l'_{k-a}, l_a) with $0 \leq a \leq k$ represents k points that are dominated by point p_l and $l'_0 = l_0 = l$. Now we obtain the following lemma.

Lemma 8. *Suppose $1 \leq i < j \leq n$, and k is a fixed integer. A point $p_l \in P_{i,j}$ dominates at least k points in $P_{i,j}$ if and only if there exists some a with $0 \leq a \leq k$, such that $i \leq l \leq j$, $i \leq l'_{k-a}$ and $l_a \leq j$.*

Recall that $\alpha(k)$ (similarly, $\beta(k)$) is the point with the smallest index greater than (similarly, the highest index smaller than) p_k that dominates p_k. We use the mapping technique as follows. For each $1 \leq l \leq n$, we map point p_l to at most $k+1$ points $(l, \alpha(l), \beta(l), l_a, l'_{k-a}) \in \mathbb{R}^5$, where $0 \leq a \leq k$. This again gives us a set of at most $(k+1)n$ points. From Lemma 2 and Lemma 8 we obtain the following.

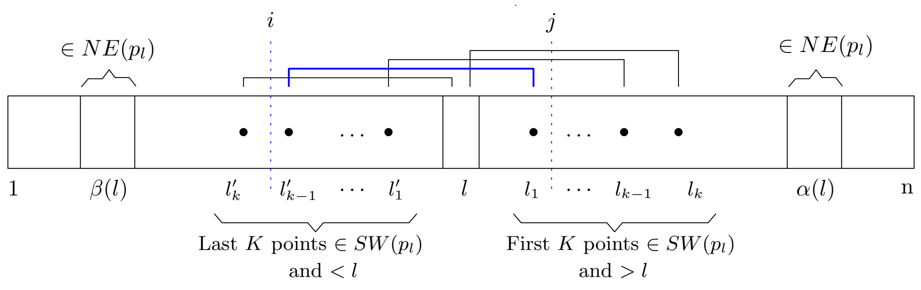

Figure 9: Proof of Lemma 9. For an example, the highlighted interval (l'_{k-1}, l_1) satisfies the query interval $[i, j]$.

Lemma 9. *Suppose $1 \leq i \leq l \leq j \leq n$ and k is a fixed integer. The point p_l is a maximal point in $P_{i,j}$ that dominates at least k points in $P_{i,j}$ if and only if there are at least k points $(l, \alpha(l), \beta(l), l_a, l'_{k-a}) \in \mathbb{R}^5$, $\alpha(l) > j$ and $\beta(l) < i$.*

We modify the query q in line 6 of Algorithm 2 by $q' = [p_{i,x}, +\infty) \times [p_{i,y}, +\infty)$ to find the k points that are dominated by each point p_i. As before, this process takes $O(nk + n \log n)$ time and $O(nk)$ space. We build a 3-dimensional range tree for the first three coordinates of the $(k+1)n$ points $(l, \alpha(l), \beta(l), l_a, l'_{k-a})$. At the last level, for each canonical node we add a PST that is built on the last two coordinates of the points stored in that node. It takes $O(kn \log^4 n)$ time and $O(kn \log^3 n)$ space in total. We next map our query interval $q = [i, j]$ to $q' = [i, j] \times [j+1, \infty) \times (-\infty, i-1] \times (-\infty, j] \times [i, \infty))$. To report each point exactly once we use the technique that we applied in Section 3.4. Each query takes $O(\log^4 n + kw)$ time, where w is the output size.

Theorem 7. *Suppose $P = (p_1, p_2, \ldots, p_n)$ is a sequence of n points in $\mathbb{R}^2$ and k is a fixed integer. Then P can be preprocessed into a data structure of size $O(kn \log^3 n)$ in $O(kn \log^4 n)$ time such that given a query interval $[i, j]$ we can report all k-dominant points in $P_{i,j}$ in $O(\log^4 n + kw)$ time, where w is the output size.*

3.6. Generalization to Higher Dimensions

In this section, we show how to generalize our results to any dimension $d \geq 2$ to solve all window query problems for points on maximal layers.

Definition 3. *The maximal layer L_1 of a set of points in $\mathbb{R}^d$ is defined to be the maximal points in the set under the dominance relation where a point p is said to be dominated by a point p' if $p[k] \leq p'[k]$ for $1 \leq k \leq d$ and $p \neq p'$.*

Recall that we presented a data structure for computing alpha- and beta-values of for points in $\mathbb{R}^d$ in Section 3.1. Note that in all problems presented in this section, except for the problem that reports whether point p_k is on some maximal layer L_3 or more, we follow a mapping technique where a point p_k in $\mathbb{R}^2$ is mapped to a point in higher dimensions using the timestamp k of p_k, $\alpha(k)$, $\beta(k)$ and possibly other points in $\mathbb{R}^2$ that satisfy some query property. So the main data structures to answer all these problems remain the same in higher dimensions as well. However, the time and space required to preprocess $\alpha(k)$ and $\beta(k)$ for all $p_k \in \mathbb{R}^d$ dominate the total preprocessing time and space requirement for these data structures. So using Theorem 2 we directly obtain the following results for the general case (for a fixed $d \geq 2$).

Theorem 8. *A sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^d$, where $d \geq 3$, can be preprocessed into a data structure of size $O(n \log^{d-2} n)$ in time $O(n \log^{d-1} n)$ such that given a query interval $[i, j]$ and a point p_k with $i \leq k \leq j$, we can report whether p_k is on the maximal layer L_1 of points $P_{i,j}$ in time $O(1)$.*

Theorem 9. *A sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^d$, where $d \geq 4$, can be preprocessed into a data structure of size $O(n \log^{d-2} n)$ in time $O(n \log^{d-1} n)$ such that given a query interval $[i, j]$ we can report the total number of maximal points of $P_{i,j}$ in time $O(\log^2 n)$. For $2 \leq d \leq 3$, $O(n \log^2 n)$ space and $O(n \log^2 n)$ time are required for preprocessing.*

Theorem 10. *Suppose $P = (p_1, p_2, \ldots, p_n)$ is a sequence of n points in $\mathbb{R}^d$ and k is a fixed integer. When $d > 3$, P can be preprocessed into a data structure* *of size $O(kn \log^{d-2} n)$ in $O(kn \log^{d-1} n)$ time such that given a query interval $[i, j]$ we can report all k-dominated points in $P_{i,j}$ in $O(\log^2 n + kw)$ time, where w is the output size. For $2 \le d \le 3$, $O(kn \log n)$ space and $O(kn \log^2 n)$ time are required for preprocessing.*

Theorem 11. *Suppose $P = (p_1, p_2, \ldots, p_n)$ is a sequence of n points in $\mathbb{R}^d$ and k is a fixed integer. When $d > 5$, P can be preprocessed into a data structure of size $O(kn \log^{d-2} n)$ in $O(n \log^{d-1} n)$ time such that given a query interval $[i, j]$ we can report all maximal points of $P_{i,j}$ that dominate at least k points in $P_{i,j}$ in $O(\log^4 n + kw)$ time, where w is the output size. For $2 \le d \le 5$, $O(kn \log^3 n)$ space and $O(kn \log^4 n)$ time are required for preprocessing.*

Now we discuss how to report whether point p_k is on some maximal layer L_3 or more in $\mathbb{R}^d$. We modify the functions $f(l)$ and $g(l)$ to map each point $p_l = (p_{l,x1}, p_{l,x2}, \cdots, p_{l,xd}) \in P$, where $1 \le l \le n$, to a point in $\mathbb{R}^{d+2}$ as follows. We redefine the functions as $f(l) = (p_{l,x1}, p_{l,x2}, \cdots, p_{l,xd}, l, l') \in \mathbb{R}^{d+2}$ and $g(l) = (p_{l,x1}, p_{l,x2}, \cdots, p_{l,xd}, l, l'') \in \mathbb{R}^{d+2}$. Our range tree data structures that store these sets of points now have $d+2$ dimensions. To query accordingly, we redefine our query functions as $F(i, j, k) = [p_{k,x1}, \infty) \times [p_{k,x2}, \infty) \times \cdots \times [p_{k,xd}, \infty) \times [i, \infty) \times (-\infty, j]$, and $G(i, j, k) = [p_{k,x1}, \infty) \times [p_{k,x2}, \infty) \times \cdots \times [p_{k,xd}, \infty) \times (-\infty, j] \times [i, \infty)$. The corresponding result is the following.

Theorem 12. *Suppose $P = (p_1, p_2, \cdots, p_n)$ is a sequence of n points in $\mathbb{R}^d$, where $d \ge 3$. Then P can be preprocessed into a data structure of size $O(n \log^{d+1} n)$ in $O(n \log^{d+1} n)$ time such that given a query interval $[i, j]$ and an integer k with $i \le k \le j$, we can decide whether p_k is on some maximal layer L_γ, where $\gamma = 2, \ge 3$, of the point set $P_{i,j} = (p_i, p_{i+1}, \cdots, p_j)$ in $O(\log^{d+1} n)$ time.*

4. Window-aggregate Queries using Coresets

Let μ be a fixed aggregate function that takes a set of points P in $\mathbb{R}^d$ and assigns a real number $\mu(P) \ge 0$ to it. Examples of $\mu(P)$ can be the diameter, width, radius of the minimum enclosing ball, volume of the smallest bounding box, and the cost of clustering (e.g., k-center, k-rectilinear) of points in P. In this section we present results for answering *window-aggregate* queries for some standard geometric problems on a sequence $P = (p_1, p_2, \ldots, p_n)$ of n points in $\mathbb{R}^d$. More specifically, given a query window $[i, j]$ with $1 \leq i < j \leq n$ we want to find $\mu(P_{i,j})$.

Although the examples of function $\mu(\cdot)$ mentioned above are all well-studied geometric problems in general, exact solutions to these problems are expensive even for lower dimensions. See [15] for known exact and approximate results of these problems. As a result, faster techniques are explored to find approximate solutions to these problems that are $(1 + \epsilon)$ factor within the optimal solutions, where $\epsilon > 0$ is an input parameter. Several studies [16, 9, 17, 15] showed that many of these geometric problems can be linearly approximated by computing and using constant-size *coresets* (see Definition 4).

Definition 4. *Let P be a set of points in $\mathbb{R}^d$ and let $\epsilon > 0$ be a real number. A subset $P' \subseteq P$ is called an ϵ-coreset of P with respect to μ if $\mu(P') \geq (1 - \epsilon) \cdot \mu(P)$ [9].*

More specifically, Agarwal et al. [9] showed that coresets for some of these problems can be computed based on the coresets of the *extent measure* of the point set. The *extent* of a point set P along a given direction is the width of the minimum slab that is orthogonal to the direction and that encloses P.

Definition 5. *The extent measure of a point set P with respect to a point $x \in \mathbb{R}^d$ is defined as $w(P, x) = \max_{p,q \in P} (p - q) \cdot x$.*

The coreset based on the extent measure is the subset S of P such that the extent of S is at least $(1 - \epsilon)$ times the extent of P along every direction [9]. These coresets resemble the *approximate convex hulls* of the point set. As a result, problems that depend on the extent measure or the convex hulls such as width, minimum-radius enclosing circle and minimum-volume bounding box etc. can be approximated using these coresets. Let $f_d(\epsilon)$ be the smallest integer such that for any set P of points in $\mathbb{R}^d$ and any real number $\epsilon > 0$ an ϵ-coreset

$C(P, \epsilon)$ has size at most $f_d(\epsilon)$ (note that the size of the coreset depends on d but not on the size of P). We restate Lemma 23.3 from [18] here.

Lemma 10. *[18, Lemma 23.3]) Consider $P_1' \subseteq P_1 \subseteq \mathbb{R}^d$ and $P_2' \subseteq P_2 \subseteq \mathbb{R}^d$, where P_1' (respectively P_2') is an ϵ-coreset of P_1 (respectively P_2). Then $P_1' \cup P_2'$ is an ϵ-coreset for $P_1 \cup P_2$.*

Following this lemma, all coresets that can be computed based on the extent measures of a point set are called *decomposable* coresets. We also define the decomposable coreset function as follows.

Definition 6. *The function C is a decomposable coreset function if the following holds for any finite set P of points in $\mathbb{R}^d$, any $\epsilon > 0$, and any partition of P into two sets P_1 and P_2: Given only the coresets $C(P_1, \epsilon)$ and $C(P_2, \epsilon)$ of P_1 and P_2, respectively, we can compute the ϵ-coreset $C(P, \epsilon)$ of P in $O(f_d(\epsilon))$ time [19].*

Some approximation results using coresets are also available where the coresets of the points do not depend on the coresets of the extent or the convex hull of the points, e.g., ℓ-center clustering problem, where $\ell \geq 2$ is an input parameter [16].

In the following sections we present techniques that, given a query interval $[i, j]$, with $1 \leq i < j \leq n$, can efficiently find an approximation for $\mu(P_{i,j})$ using coresets.

4.1. Geometric Problems using Decomposable Coresets

In this section we want to compute $\mu(P_{i,j})$ for a query interval $[i, j]$ using decomposable coresets that are available for computing μ. We follow a similar technique used by Nekrich and Smid [19] for computing range-aggregate queries. We divide the sequence of points in P in $\mathbb{R}^d$ into a sequence of $n/f_d(\epsilon)$ blocks $S = (S_1, S_2, \ldots S_{n/f_d(\epsilon)})$ such that each block S_k contains $f_d(\epsilon)$ points of P, and the overall sequence of points in blocks in S from left to right gives the exact sequence of P. We build a range tree T on points in blocks of S from left to right such that the k-th leaf of T contains the $f_d(\epsilon)$ points belonging to block S_k, for $1 \leq k \leq n/f_d(\epsilon)$. Each leaf node contains the coreset of $f_d(\epsilon)$ points. As we move to the upper levels of T, coresets of the child nodes can be combined to compute the coreset of size $O(\log(n/f_d(\epsilon)) \cdot f_d(\epsilon))$ for each canonical node v of T. However, we can reduce the size of each coreset to $O(f_d(\epsilon))$ as follows. Recall that these coresets are based on the extent measures of some subsets of points. We assume that there is a set of $O(1/\epsilon^{d-1})$ directions in $\mathbb{R}^d$ such that the angle between any two directions is $O(\epsilon)$. So the coreset stored at each canonical node contains the farthest points in every $O(1/\epsilon^{d-1})$ directions. Then the size of the coreset stored at each canonical node is $f_d(\epsilon) = O(1/\epsilon^{d-1})$. When $d = 2$, we obtain $f_2(\epsilon) = 2\pi/\epsilon$. During a query, the coreset of each canonical node can be computed by comparing the $O(f_d(\epsilon))$ points of its child nodes in $O(f_d(\epsilon))$ time. The total number of canonical nodes in T is $O(n/f_d(\epsilon))$. Therefore, the total space required for T is $O(n/f_d(\epsilon) \cdot \log(n/f_d(\epsilon)) \cdot f_d(\epsilon)) = O(n \log n)$.

For each query $q = [i, j]$, let the leaf nodes S_x and S_y of T contain the points p_i and p_j, respectively. Each of S_x and S_y contains at most $f_d(\epsilon)$ points. Then coresets for the rest of the points in $p_{i+1}, \ldots, p_{j-1}$ can be found by combining the coresets of at most $O(\log(n/f_d(\epsilon))$ canonical nodes of T. Therefore, the coreset $C(P_{i,j}, \epsilon)$ of size $f_d(\epsilon)$ can be computed in $O(\log(n/f_d(\epsilon)) \cdot f_d(\epsilon)) = O(f_d(\epsilon) \log n)$ time.

Theorem 13. *Let $P = (p_1, p_2, \ldots, p_n)$ be a sequence of n points in $\mathbb{R}^d$, where d is a fixed dimension. P can be preprocessed into a data structure of size $O(n \log n)$ such that for a query interval $[i, j]$ with $1 \leq i < j \leq n$, we can compute a $(1+\epsilon)$-coreset of size $f_d(\epsilon)$ in time $O(f_d(\epsilon) \log n)$ for geometric problems that admit some decomposable coresets.*

Applications: Theorem 13 can be used to compute $\mu(P_{i,j})$ for any queried sequence of points if $\mu(\cdot)$ admits a decomposable coreset of size $f_d(\epsilon)$. In particular, suppose we have a function μ that can be computed in $O(n^\lambda)$ time and μ admits a coreset $C(P, \epsilon)$ of size $f_d(\epsilon)$ such that $\mu(P') \geq (1-\epsilon)\mu(P)$. Then by Theorem 13 we compute $C(P_{i,j}, \epsilon)$ of size $f_d(\epsilon)$ and run an exact algorithm for computing $\mu(C(P_{i,j}, \epsilon))$. For example, since the exact diameter of a point set can be computed in quadratic time (i.e., $\lambda = 2$) in any fixed dimension, given a query interval $[i, j]$ a $(1+\epsilon)$-approximation to the diameter problem of $P_{i,j}$ can be computed in $O((f_2(\epsilon)^2 + f - 2(\epsilon) \log n)$ time. Table 3 shows running times for computing $(1 + \epsilon)$-approximations to $\mu(\cdot)$ for some well-known geometric problems, which we briefly discuss below.

Diameter: the maximum distance over all pairs of points in P. In $d = 2$ and 3, an $O(n \log n)$ time algorithm is available (randomized for $d = 3$ [20]). For $d > 4$, it can be trivially solved in quadratic time. *Width*: the minimum width over all slabs that enclose P, where a slab of width w refers to a region between two parallel hyperplanes of distance w. In $d = 3$, an $O(n^2)$ time algorithm is available [21]. *Smallest enclosing disc*: the disc with the minimum radius that contains all the points in P. An expected linear time algorithm is available [22]. *Smallest enclosing cylinder*: the minimum radius over all cylinders that enclose P, where a cylinder of radius z refers to the region of all points of distance z from a line. In $d = 3$, for a small constant $\delta > 0$, a $O(n^{3+\delta})$ time algorithm is known [23]; *Minimum-width annulus*: the minimum width over any all annuli that enclose P, where an annulus (also called a spherical shell) of width $|z-y|$ is a region between two concentric spheres of radii y and z. For $d = 2$, $O(n^{3/2+\delta})$ time randomized algorithms are available [24].

4.2. ℓ-center Clustering using Coresets

Let P be a sequence of n points $(p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$. The geometric ℓ-center clustering problem partitions P into ℓ subsets such that a certain *cost function* of the clustering is minimized. We define *windowed geometric ℓ-center clustering problem* as follows. Given a query interval $q = [i, j]$, with $1 \leq i < j \leq n$, and an integer $\ell \geq 2$, report the cost of the ℓ-clustering of the points in $P_{i,j} = (p_i, \ldots, p_j)$. Suppose $C = \{c_1, c_2, \ldots, c_\ell\}$ is the ℓ-center clustering of $P_{i,j}$. We consider the following cost functions $\Phi(C)$ of an ℓ-center clustering; (i) the maximum radius of the minimum enclosing balls of the clusters: $\min \Phi^{\max}(C) = \max_{c_\ell \in C} radius(c_\ell)$, and (ii) the minimum summation of the radius of all balls

Table 3: Window-aggregate query time for $(1+\epsilon)$-approximation for some geometric problems. Here n is the total number of input points, $f_d(\epsilon)$ is the size of the coreset of the queried points in dimension $d \geq 2$, and $\delta > 0$ is a small constant.

Problems	**Query time**
Diameter	$O(f_d(\epsilon)^2 + f_d(\epsilon)\log n)$
Width (in $\mathbb{R}^3$)	$O(f_3(\epsilon)^2 + f_3(\epsilon)\log n)$
Volume of smallest bounding box (in $\mathbb{R}^3$)	$O(f_3(\epsilon)^3 + f_3(\epsilon)\log n)$
Smallest enclosing disc	$O(f_d(\epsilon)\log n)$
Smallest enclosing cylinder (in $\mathbb{R}^3$)	$O(f_3(\epsilon)^{3+\delta} + f_3(\epsilon)\log n)$
Minimum-width annulus (in $\mathbb{R}^2$)	$O(f_2(\epsilon)^{3/2+\delta} + f_2(\epsilon)\log n)$

of the clusters: $\min \Phi^{\text{sum}}(C) = \sum_{c_\ell \in C} radius(c_\ell)$.

Abrahamsen et al. [16] presented a coreset-based algorithm that computes a $(1+\epsilon)$-approximation to the range-clustering query for ℓ-center clustering by using an ϵ-coreset of the input points. To be more specific, given a set P of n points in $\mathbb{R}^d$, for any query range Q, a fixed integer $\ell \geq 2$ and a parameter $\epsilon > 0$, they report an ϵ-coreset P' of $P \cap Q$. Then any standard clustering algorithm can generate the cost of an ℓ-clustering C for the points in $P \cap Q$ using points from P' such that $\Phi(C) \leq (1+\epsilon) \cdot \Phi(C_{opt})$, where Φ is the cost of the clustering and C_{opt} is an optimal clustering for $P \cap Q$. For the sake of completeness, we first briefly state their algorithm that returns an ϵ-coreset P' of $P_Q = P \cap Q$ [16] of size $O(\ell(f(\ell)/(c\epsilon))^d)$ and then we show how this data structure can be used to compute coresets for ℓ-center clustering in the window query setting.

Algorithm $\mathcal{A}$ from [16]: Build a compressed quad-tree T_P on the point set P. Given a query rectangle Q, search T_P to find the set of canonical squares $\mathcal{B}$ that covers P_Q. All the larger squares in $\mathcal{B}$ are refined into a total of at least $\ell 2^{2d}$ smaller squares. Let LB be the size of the largest square after the refinement; LB is a lower bound on the optimal value of the ℓ-clustering of P_Q. Set a parameter $r = \epsilon \cdot LB$. Repeatedly refine $\mathcal{B}$ until the size of each square is at most $r/\sqrt{d}$. Let $\mathcal{S}$ be the set of all smaller squares after the second refinement step. For each square $S \in \mathcal{S}$, pick a point in $P_Q \cap S$. The collection of these points is the coreset P' of P_Q.

We restate Lemmas 5 and 6 of Abrahamsen et al. [16] in terms of $d = 2$.

Lemma 11. *[16, Lemma 5] The value LB computed by Algorithm $\mathcal{A}$ is a correct lower bound on $Opt_\ell(P_Q)$, and the set R_Q is a weak r-packing for $r = \epsilon \cdot LB/f_2(\ell)$ of size $O(\ell(f_2(\ell)/(c\epsilon))^2)$.*

Lemma 12. *[16, Lemma 6] Algorithm $\mathcal{A}$ runs in $O(\ell(f_2(\ell)/(c\epsilon))^2 + \ell((f_2(\ell)/(c\epsilon)) \log n))$ time.*

Window query data structure: Recall that our input is a sequence $P = (p_1, p_2, \ldots, p_n)$ of n points in $\mathbb{R}^2$. We build a range tree T on the points of P from left to right according to their index in the sequence. Each of the canonical nodes v of T covers a subsequence of the points $P_{q,r} = (p_q, p_{q+1}, \ldots, p_r)$, with $1 \leq q < r \leq n$, in the subtree rooted at v. Algorithm $\mathcal{A}$ for computing coresets from a quad tree can be applied to the clustering problem in the window setting as follows. At each canonical node v of T we store the quad tree that covers $P_{q,r}$, the points stored in the subtree rooted at v. Given a query interval $[i, j]$ we can find $O(\log n)$ canonical nodes containing $O(\log n)$ quad trees that cover the queried subsequence of points $P_{i,j}$. If these $O(\log n)$ quad trees can quickly be combined into a single quadtree such that it covers all points in $P_{i,j}$, then a single coreset of size $O(\ell(f_2(\ell)/(c\epsilon))^2)$ for $P_{i,j}$ can be computed according to Algorithm $\mathcal{A}$. We apply the technique of Bannister et al. [2] to construct a compressed quad-tree for the queried subset of points in time linear in the width of the query window $\mathcal{W} = j - i + 1$. In each of the canonical nodes v of T, we additionally store two structures; (i) the *Morton-order* (or the Z-order) and (ii) a *skip-quadtree* over the points in the subtree of v. The Morton-order is a mapping of the locations of a multidimensional point set to a linear list of numbers [25]. A skip-quadtree is a linear-size variant of the compressed quadtree that is built on a structure similar to the skip lists [26].

During any query $q = [i, j]$, we find two overlapping canonical subsets each of width less than $2\mathcal{W}$ and merge the Z-orders stored in these subsets into one list in $O(\mathcal{W})$ time, where $\mathcal{W} = j - i + 1$ (by Lemma 13 in [2]). According to Chan [27], a compressed quad tree can be computed from the final Z-order in $O(\mathcal{W})$ time. Now we apply Algorithm $\mathcal{A}$ to compute a coreset for points in $P_{i,j}$ for ℓ-center clustering. Now Lemmas 11 and 12 yield the following result.

Theorem 14. *Let P be a sequence of n points in $\mathbb{R}^2$. Then P can be preprocessed into a data structure of size $O(n \log n)$ such that given a query window $[i, j]$ and two parameters $\ell \geq 2$ and $\epsilon > 0$, a coreset of size $O(\ell(f_2(\ell)/(c\epsilon))^2)$ for the ℓ-center clustering can be computed in $O(\ell((f_2(\ell)/(c\epsilon)) \log n) + \ell(f_2(\ell)/(c\epsilon))^2 + \mathcal{W})$ time, where $\mathcal{W}$ is the width of the query window and c is a positive constant.*

Applications: Given a query window of width $\mathcal{W} = j-i+1$, with $1 \leq i < j \leq n$, we can compute a $(1+\epsilon)$-approximation to the cost of the ℓ-center clustering of points in $P_{i,j}$ within the following bounds (by Corollary 8 in [16]).

- Euclidean ℓ-center clustering with $\ell = 2$: the cost is measured by the maximum Euclidean diameter of the cluster. Queries can be answered in $O((1/\epsilon) \log n + (1/\epsilon^2) \log^2(1/\epsilon) + \mathcal{W})$ expected time.

- Rectilinear ℓ-center clustering with $\ell = 2, 3$: the cost is measured by half of the maximum side length of a minimum enclosing axis-aligned cube of the cluster (in L_∞-metric). Queries can be answered in $O((1/\epsilon) \log n + 1/\epsilon^2 + \mathcal{W})$ time.

- Rectilinear ℓ-center clustering with $\ell = 4, 5$: queries can be answered in $O((1/\epsilon) \log n + (1/\epsilon^2)\, \text{polylog}(1/\epsilon) + \mathcal{W})$ time.

5. Conclusion

In this paper, we present data structures to solve different types of window queries using geometric objects. Our window data structures can answer windowed intersection decision queries for line segments, triangles and convex

c-gons in plane. We also present solutions for windowed queries for counting maximal points, and reporting whether a given point is on the maximal layers L_γ ($\gamma = 1, 2, \geq 3$), all k-dominated points and all k-dominant points in $\mathbb{R}^d$, where $d \geq 2$. Finally we show how to approximate window queries by computing $(1+\epsilon)$-approximations to various geometric problems such as the diameter, width, volume of the smallest bounding box, radius of the smallest enclosing disc or cylinder, minimum-width annulus and ℓ-center clustering ($\ell \geq 2$) using coresets.

Remark: The query time of the windowed intersection decision problem can be improved so that given a query interval $[i, j]$, with $1 \leq i \leq j \leq n$, a data structure of size $O(n)$ can answer a query in $O(1)$ time as follows. Let B be an array of size n. For each $B[\alpha]$, with $1 \leq \alpha \leq n$, we store the minimum index y such that any two objects intersect within the time interval $[\alpha, y]$. Note that during the preprocessing stage we find n valid pairs (α, β) such that $A[\beta]$ is the first object that intersects $A[\alpha]$ and $\beta > \alpha$. We store the set U of these n points (α, β) in an orthogonal range successor data structure of size $n + o(n)$ such that for a query region $Q = [\alpha, +\infty) \times [\alpha, +\infty)$ it reports the point $p = (x, y)$ such that $p \in U \cap Q$ and y is minimum β value among all points in $U \cap Q$ in $O(\log n)$ time [28]. We set $B[\alpha] = y$. For all $\alpha = 1, \ldots, n$, we can find $B[\alpha]$ in $O(n \log n)$ time which is upperbounded by the original preprocessing time (see, for example, Corollary 5). For a given query interval $q = [i, j]$, if $j \geq B[i]$ we report that some objects intersect in q, otherwise all objects in q are disjoint. The improved result is summarized in the following theorem.

Theorem 15. *Given a sequence S of n geometric objects, we can preprocess S into a data structure of size $O(n)$ in time $O(P(n) \log n + n \cdot Q(n) \log n)$ so that it can answer windowed intersection decision queries in $O(1)$ time. The space used during preprocessing is $O(M(n) \log n)$.*

References

[1] F. Chanchary, A. Maheshwari, M. H. M. Smid, Window queries for problems on intersecting objects and maximal points, in: Algorithms and Discrete Applied Mathematics - 4th International Conference, CALDAM 2018, Guwahati, India, February 15-17, 2018, Proceedings, Vol. 10743 of Lecture Notes in Computer Science, Springer, 2018, pp. 199–213. doi:10.1007/978-3-319-74180-2_17.
URL https://doi.org/10.1007/978-3-319-74180-2_17

[2] M. J. Bannister, W. E. Devanny, M. T. Goodrich, J. A. Simons, L. Trott, Windows into geometric events: Data structures for time-windowed querying of temporal point sets, in: Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Halifax, Nova Scotia, Canada, 2014, 2014.
URL http://www.cccg.ca/proceedings/2014/papers/paper02.pdf

[3] M. J. Bannister, C. DuBois, D. Eppstein, P. Smyth, Windows into relational events: Data structures for contiguous subsequences of edges, in: Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013, 2013, pp. 856–864. doi:10.1137/1.9781611973105.61.
URL https://doi.org/10.1137/1.9781611973105.61

[4] D. Bokal, S. Cabello, D. Eppstein, Finding all maximal subsequences with hereditary properties, in: 31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands, Vol. 34 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015, pp. 240–254. doi:10.4230/LIPIcs.SOCG.2015.240.
URL https://doi.org/10.4230/LIPIcs.SOCG.2015.240

[5] T. M. Chan, S. Pratt, Two approaches to building time-windowed geometric data structures, in: 32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA, Vol. 51

of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 28:1–28:15. doi:10.4230/LIPIcs.SoCG.2016.28.
URL https://doi.org/10.4230/LIPIcs.SoCG.2016.28

[6] F. Chanchary, A. Maheshwari, Time windowed data structures for graphs, J. Graph Algorithms Appl. 23 (2) (2019) 191–226. doi:10.7155/jgaa.00489.
URL https://doi.org/10.7155/jgaa.00489

[7] F. Chanchary, A. Maheshwari, M. H. M. Smid, Querying relational event graphs using colored range searching data structures, Discret. Appl. Math. 286 (2020) 51–61. doi:10.1016/j.dam.2019.03.006.
URL https://doi.org/10.1016/j.dam.2019.03.006

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 3rd Edition, MIT Press, 2009.
URL http://mitpress.mit.edu/books/introduction-algorithms

[9] P. K. Agarwal, S. Har-Peled, K. R. Varadarajan, Approximating extent measures of points, J. ACM 51 (4) (2004) 606–635. doi:10.1145/1008731.1008736.
URL https://doi.org/10.1145/1008731.1008736

[10] K. Mouratidis, S. Bakiras, D. Papadias, Continuous monitoring of top-k queries over sliding windows, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006, ACM, 2006, pp. 635–646. doi:10.1145/1142473.1142544.
URL https://doi.org/10.1145/1142473.1142544

[11] T. M. Chan, Optimal partition trees, Discrete & Computational Geometry 47 (4) (2012) 661–690. doi:10.1007/s00454-012-9410-z.
URL https://doi.org/10.1007/s00454-012-9410-z

[12] E. M. McCreight, Priority search trees, SIAM J. Comput. 14 (2) (1985)

257–276. doi:10.1137/0214021.
URL https://doi.org/10.1137/0214021

[13] C. W. Mortensen, Fully dynamic orthogonal range reporting on RAM, SIAM J. Comput. 35 (6) (2006) 1494–1525. doi:10.1137/S0097539703436722.
URL https://doi.org/10.1137/S0097539703436722

[14] M. de Berg, O. Cheong, M. J. van Kreveld, M. H. Overmars, Computational geometry: algorithms and applications, 3rd Edition, Springer, 2008.
URL http://www.worldcat.org/oclc/227584184

[15] T. M. Chan, Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus, Int. J. Comput. Geometry Appl. 12 (1-2) (2002) 67–85. doi:10.1142/S0218195902000748.
URL https://doi.org/10.1142/S0218195902000748

[16] M. Abrahamsen, M. de Berg, K. Buchin, M. Mehr, A. D. Mehrabi, Range-clustering queries, in: 33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia, Vol. 77 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, pp. 5:1–5:16. doi:10.4230/LIPIcs.SoCG.2017.5.
URL https://doi.org/10.4230/LIPIcs.SoCG.2017.5

[17] P. K. Agarwal, S. Har-Peled, K. R. Varadarajan, Geometric approximation via coresets, Combinatorial and Computational Geometry 52 (2005) 1–30.

[18] S. Har-Peled, Geometric approximation algorithms, American Mathematical Soc., 2011.

[19] Y. Nekrich, M. H. M. Smid, Approximating range-aggregate queries using coresets, in: Proceedings of the 22nd Annual Canadian Conference on Computational Geometry, Winnipeg, Manitoba, Canada, August 9-11, 2010, 2010, pp. 253–256.
URL http://cccg.ca/proceedings/2010/paper67.pdf

[20] K. L. Clarkson, Applications of random sampling in computational geometry, II, in: Proceedings of the Fourth Annual Symposium on Computational Geometry, Urbana-Champaign, IL, USA, June 6-8, 1988, ACM, 1988, pp. 1–11. `doi:10.1145/73393.73394`.
URL `https://doi.org/10.1145/73393.73394`

[21] M. E. Houle, G. T. Toussaint, Computing the width of a set, IEEE Trans. Pattern Anal. Mach. Intell. 10 (5) (1988) 761–765. `doi:10.1109/34.6790`.
URL `https://doi.org/10.1109/34.6790`

[22] E. Welzl, Smallest enclosing disks (balls and ellipsoids), in: H. A. Maurer (Ed.), New Results and New Trends in Computer Science, Graz, Austria, June 20-21, 1991, Proceedings, Vol. 555 of Lecture Notes in Computer Science, Springer, 1991, pp. 359–370. `doi:10.1007/BFb0038202`.
URL `https://doi.org/10.1007/BFb0038202`

[23] P. K. Agarwal, B. Aronov, M. Sharir, Line transversals of balls and smallest enclosing cylinders in three dimensions, Discrete & Computational Geometry 21 (3) (1999) 373–388. `doi:10.1007/PL00009427`.
URL `https://doi.org/10.1007/PL00009427`

[24] P. K. Agarwal, M. Sharir, Efficient randomized algorithms for some geometric. optimization problems, Discrete & Computational Geometry 16 (4) (1996) 317–337. `doi:10.1007/BF02712871`.
URL `https://doi.org/10.1007/BF02712871`

[25] G. M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, International Business Machines Company New York, 1966.

[26] D. Eppstein, M. T. Goodrich, J. Z. Sun, The skip quadtree: a simple dynamic data structure for multidimensional data, in: Proceedings of the 21st ACM Symposium on Computational Geometry, Pisa, Italy, June 6-8, 2005, 2005, pp. 296–305. `doi:10.1145/1064092.1064138`.
URL `https://doi.org/10.1145/1064092.1064138`

[27] T. M. Chan, Well-separated pair decomposition in linear time?, Inf. Process. Lett. 107 (5) (2008) 138–141. doi:10.1016/j.ipl.2008.02.008. URL https://doi.org/10.1016/j.ipl.2008.02.008

[28] V. Mäkinen, G. Navarro, Rank and select revisited and extended, Theor. Comput. Sci. 387 (3) (2007) 332–347. doi:10.1016/j.tcs.2007.07.013. URL https://doi.org/10.1016/j.tcs.2007.07.013