TIME WINDOWED DATA STRUCTURES

FARAH CHANCHARY

A thesis submitted to the Faculty of Graduate and Post Doctoral Affairs
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

Carleton University
Ottawa, Ontario, Canada, 2019

ABSTRACT

We study *time windowed data structures*, where a *timestamped* input sequence can be preprocessed into a data structure based on a given predicate $\mathcal{P}$, so that for a query time interval specified at the query time, it can answer queries based on the inputs whose timestamps lie in the query interval and match $\mathcal{P}$. We refer to these queries as *window queries*. In this thesis, we consider several variations of these window queries for three types of inputs; they are relational event (RE) graphs, geometric objects (e.g., points, line segments, polygons) and set elements. We study three types of query problems, namely *window decision problems*, *window reporting problems* and *window counting problems*.

Considering an RE graph $G = (|V| = n, |E| = m)$ as the input, where each edge in E has a unique positive timestamp, we present data structures to answer the following window problems; (a) decision problems for monotone graph properties, such as disconnectedness and bipartiteness, (b) reporting problems for the minimum spanning tree, the minimum spanning interval, and the graph edit distance for spanning forests, and (c) subgraph counting problems to count the total number of subgraphs that match a given pattern, such as paths of length 2 and 3 (in bipartite graphs), quadrangles and complete subgraphs of some fixed order $\ell \geqslant 3$. We further present a general approach for analyzing various structural parameters of an RE graph slice, such as the *density*, the number of k-stars, the approximation of h-*index*, the *embeddedness*, the *neighborhood overlap* and the number of *influenced vertices*, using the colored range searching data structures.

We also study data structures that can answer window queries for a sequence of geometric objects, such as points, line segments and convex c-gons. We study the *windowed intersection decision problems* for these objects. For a sequence of points in $\mathbb{R}^d$, $d \geqslant 2$, we solve some variations of the maximal layer problem, such as counting total number of points on the maximal layer, k-dominated and k-dominant points for some fixed integer k, and deciding if a given point belongs to a maximal layer L_γ, where $\gamma = 1, 2$ or $\gamma \geqslant 3$. Lastly, we present data structures that report $(1+\epsilon)$-approximations to *window-aggregate queries* for various geometric problems, such as diameter, width, radius of a minimum enclosing ball, volume of the smallest bounding box, and the cost of ℓ-center clustering ($\ell \geqslant 2$), using a constant size coreset. These data structures answer queries in polylogarithmic time and use subquadratic space.

We present two data structures that answer window queries for *stochastic* input sequences. Let $\alpha_1, \ldots, \alpha_c$ be constants, with $0 < \alpha_1 < \alpha_2 < \ldots < \alpha_c < 1$. Let $P = P_1 \cup P_2 \cup \ldots \cup P_c$ be a set of n points in $\mathbb{R}^d$, for some fixed d. For $r = 1, 2, \ldots, c$, let all points in P_r be colored by the rth color. Furthermore, for $r = 1, 2, \ldots, c$, each point in P_r is associated with probability α_r. Given a query interval we find the point that has the highest probability to be on the maximal layer in the query interval. In the second problem, S is an $m \times n$ matrix in which each entry is a real number in $[0, 1]$. We present a data structure so that for any query window it can decide if there exists an element in each row of the query window with probability at least τ, with $0 < \tau \leqslant 1$.

ACKNOWLEDGMENTS

I would not have been able to write this thesis without the help and support of many people.

I wish to express sincere gratitude to my supervisors Anil Maheshwari and Michiel Smid. I am grateful for their invaluable guidance, constant support and advice through out this research work.

I sincerely thank the members of my thesis defense committee for their insightful comments. I am specially grateful to Stephane Durocher for referring to the binary rank-select data structure that can improve some of the existing bounds, and to Jean-Lou De Carufel for pointing out many minor details that certainly improved the overall readability of my thesis.

I am ever grateful to my family members, my parents and my friends for their unconditional support and constant encouragement for me, specially during these years.

And finally, I would like to thank all members and students of the Computational Geometry lab at Carleton University for making the lab an awesome place to work and learn.

I THANK YOU ALL.

CONTENTS

PREFACE

This thesis is in "integrated article format" in which each chapter is based on published papers, conference proceedings, or papers awaiting publication.

- Chapter 2 considers the window query problems in the relational event graphs. This chapter is a combination of the results that have been presented in the 10th International Workshop on Algorithms and Computation [2] and the results that have been published in the Journal of Graph Algorithms and Applications [3]. This is a joint work with Anil Maheshwari.
- Chapter 3 presents the window query problems using colored range searching data structures. This chapter is a combination of the results that have been presented in the 3rd International Conference on Algorithms and Discrete Applied Mathematics [6] and the results that have been accepted for publication in the Journal of Discrete Applied Mathematics [5]. This is a joint work with Anil Maheshwari and Michiel Smid.
- Chapter 4 presents various window query problems for geometric objects (points, line segments and polygons) and window-aggregate queries using coresets. This chapter is a combination of results that have been presented in the 4th International Conference on Algorithms and Discrete Applied Mathematics [7] and the results that have been submitted in the Journal of Discrete Applied Mathematics [4]. This is a joint work with Anil Maheshwari and Michiel Smid.
- Chapter 5 presents window data structures for stochastic input sequences, such as points in $\mathbb{R}^d$, for a fixed $d \geqslant 1$, and a set of elements. This chapter presents the results that have been submitted to the International Journal of Computational Geometry & Applications [1]. This is a joint work with Paz Carmi, Anil Maheshwari and Michiel Smid.

BIBLIOGRAPHY

[1] Carmi, P., Chanchary, F., Maheshwari, A., Smid, M.: The most likely object to be seen through a window. Submitted to International Journal of Computational Geometry & Applications (2019)

[2] Chanchary, F., Maheshwari, A.: Counting subgraphs in relational event graphs. In: WALCOM: Algorithms and Computation - 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9627, pp. 194–206. Springer (2016), https://doi.org/10.1007/978-3-319-30139-6_16

[3] Chanchary, F., Maheshwari, A.: Time windowed data structures for graphs. Journal of Graph Algorithms and Applications 23(2), 191–226 (2019)

[4] Chanchary, F., Maheshwari, A., Smid, M.: Window queries for problems on intersecting objects, maximal points and approximation using core sets. Submitted to Journal of Discrete Applied Mathematics (2018)

[5] Chanchary, F., Maheshwari, A., Smid, M.: Querying relational event graphs using colored range searching data structures. Accepted for publication in Journal of Discrete Applied Mathematics (2019)

[6] Chanchary, F., Maheshwari, A., Smid, M.H.M.: Querying relational event graphs using colored range searching data structures. In: Algorithms and Discrete Applied Mathematics - Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10156, pp. 83–95. Springer (2017), https://doi.org/10.1007/978-3-319-53007-9_8

[7] Chanchary, F., Maheshwari, A., Smid, M.H.M.: 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. Lecture Notes in Computer Science, vol. 10743, pp. 199–213. Springer (2018), https://doi.org/10.1007/978-3-319-74180-2_17

1 INTRODUCTION

In this section we provide a summary of the thesis. First we describe some graph theoretic definitions followed by the definitions of different types of window query problems that we present in later chapters. We also provide brief descriptions of the main techniques that are used to obtain our results.

1.1 GRAPH THEORETIC DEFINITIONS AND WINDOW QUERIES

A *relational event (RE) graph* $G = (V, E = (e_1, e_2, \ldots, e_m))$ is an undirected graph with a fixed set V of n vertices and a sequence of edges E between pairs of vertices, where each edge has a unique positive timestamp [7]. Without loss of generality, we assume that $t(e_1) < t(e_2) < \ldots < t(e_m)$, where $t(e_i)$ is the timestamp of the edge e_i. A *graph slice* for the interval $[i, j]$ is defined as $G_{i,j} = (V, E_{i,j} = \{e_i, e_{i+1}, \ldots, e_j\})$. See Figure 1.1 for an example.

Arboricity $a(G)$ of a graph $G = (V, E)$ having $m = |E|$ edges and $n = |V|$ vertices is the minimum number of edge-disjoint spanning forests into which G can be partitioned [22]. For a connected graph G, $a(G) = O(\sqrt{m})$ [18].

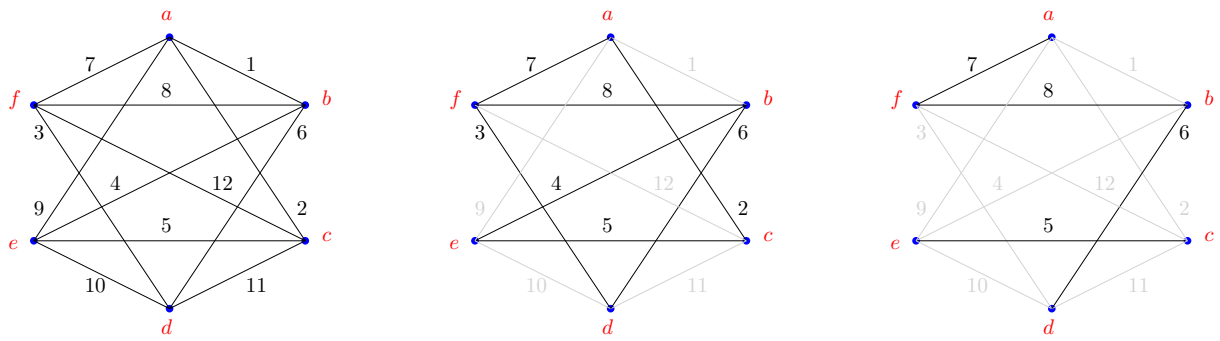


Figure 1.1: (a) An RE graph G with twelve edges. Timestamp of each edge is mentioned as an integer value. (b) A connected graph slice $G_{2,8}$. (c) A disconnected graph slice $G_{5,8}$.

In this thesis we always consider a sequence $S = (s_1, s_2, \ldots, s_n)$ of n time stamped objects as the input, where the objects can be edges of an RE graph, geometric objects such as points, line segments and polygons, or elements of a set such that the input s_k in S has the timestamp k.

In a *window query* we are given a sequence S of n input objects and a predicate $\mathcal{P}$. We want to preprocess the sequence of input objects into a suitable data structure such that given a query interval $q = [i, j]$, with $1 \leqslant i < j \leqslant n$ (for RE graphs it is $1 \leqslant i < j \leqslant m$), it can efficiently answer the query based on the input elements that lie in the query interval - denoted by $S_{i,j}$ and matches $\mathcal{P}$.

In this thesis we examine the following three types of window queries.

1. The *window decision query* is to report whether a given predicate $\mathcal{P}$ is satisfied within a query interval.
2. The *window reporting query* returns the output within a query interval that matches a given predicate $\mathcal{P}$.
3. The *window counting query* returns the size of the output, that is the total number of elements within a query interval that match a given predicate $\mathcal{P}$.

1.2 PROBLEM DEFINITIONS

This section provides the definitions of the window problems that we consider in the thesis. For graph problems the input is an RE graph $G = (|V| = n, |E| = m)$. For geometric problems the input is a sequence of n geometric objects, such as points, line segments or convex polygons. We denote the query interval as $[i, j]$, with $1 \leqslant i \leqslant j \leqslant m$ for all graph problems and $1 \leqslant i \leqslant j \leqslant n$ for all geometric problems.

1.2.1 *Graph Problems*

1. Decision problem for monotone graph properties:

   DEFINITION 1.1. *A graph property $\mathcal{P}$ is monotone if every graph slice $G_{i,j} = (V, E_{i,j} = \{e_i, e_{i+1}, \ldots, e_j\})$ of an RE graph G with property $\mathcal{P}$ also has property $\mathcal{P}$.*

   An RE graph G and a monotone graph property $\mathcal{P}$ are given. For any query interval $[i, j]$ decide whether the graph slice $G_{i,j} = (V, E_{i,j})$ satisfies $\mathcal{P}$.

2. Report minimum spanning tree:

   DEFINITION 1.2. *A minimum spanning tree (MST) is a subset of the edges of a connected, edge-weighted graph that connects all the vertices without creating any cycles and has the minimum possible total edge weight.*

   Suppose a weighted RE graph G is given where each edge of G has a positive real weight. Let the weight of the MST of G be denoted by ω^*. For a query interval $[i, j]$, report whether there exists some MST $T = (V, E')$ of G such that $E' \subseteq \{e_i, \cdots, e_j\}$ with weight ω^*.

3. Report minimum spanning interval:

   DEFINITION 1.3. *The spanning interval is defined as the time difference $t(e_q) - t(e_p)$ such that there exists an MST $T = (V, E' \subseteq \{e_p, \cdots, e_q\})$ of G.*

   Suppose a weighted RE graph G is given where each edge of G has a positive real weight. Let the weight of the MST of G be denoted by ω^*. For a query interval $[i, j]$ report the minimum spanning interval $t(e_q) - t(e_p)$ such that there is an MST of G in $G_{i,j}$ with weight ω^* and $i \leqslant p \leqslant q \leqslant j$.

4. Report graph edit distance:

   DEFINITION 1.4. *Given a set of graph edit operations (insertion of graph edges), the graph edit distance* $\mathcal{GED}(G, H)$ *between a source graph* G *and a target graph* H *is defined as,*

   $$\mathcal{GED}(G, H) = \min\{c(S) | S \textit{ is a sequence of operations transforming } G \textit{ into } H\}.$$

   In this definition, $S = (s_1, s_2, \ldots, s_k)$ is a sequence of operations that transforms G into H. The cost of a sequence $S = (s_1, s_2, \ldots, s_k)$ is given by $c(S) = \sum_{i=1}^{k} c(s_i)$, where $c(s_i)$ is the cost of the operation s_i. The goal of the graph edit distance is to find the minimum cost of the operations that makes the transformation possible. We consider *edge deletion* as the only permitted graph edit operation to solve our problem. We assume that for unweighted graphs each edit operation has a unit cost.

   An RE graph G is given. For a query interval $[i, j]$ report the $\mathcal{GED}(G_{i,j}, H)$ where $H = (V, E')$ is a spanning forest of $G_{i,j}$ and $E' \subseteq \{e_i, e_{i+1}, \ldots, e_j\}$.

5. Counting subgraphs:

   DEFINITION 1.5. *A graph* $G' = (V', E')$ *is a subgraph of graph* $G = (V, E)$ *if* $V' \subseteq V$ *and* $E' \subseteq E$.

   An RE graph G is given. For a query interval $[i, j]$ report the number of subgraphs of some fixed patterns that exists in $G_{i,j}$. We present data structures for the following patterns: 2-paths (paths of length two in general graphs), 3-paths (paths of length three in bipartite graphs), all complete subgraphs of size 3 and more, and quadrangles or simple cycles of length 4 (C_4).

   A set of quadrangles can be implicitly represented by a tuple $(y, z, \{a_1, a_2, a_3, ...\})$ in $O(a(G)m)$ time and space [18], where y and z are vertices on two opposite sides of all quadrangles of this set and each vertex $v \in \{a_1, a_2, a_3, ..\}$ shares edges with both y and z. Within this setting, any two vertices from $\{a_1, a_2, a_3, ..\}$ together with y and z represent a quadrangle. Figure 1.2 illustrates an example.

6. Report graph density:

   DEFINITION 1.6. *The density of an undirected simple graph* $G = (V, E)$ *is defined as* $D(G) = \frac{|E|}{\binom{|V|}{2}}$ *[36].*

   An RE graph G is given. In this setting, a graph slice is defined as $G_{i,j} = (V_{i,j}, E' = \{e_i \cup e_{i+1} \cup \cdots \cup e_j\})$, where $V_{i,j}$ is the set of vertices incident on edges of E'. For a query interval $[i, j]$ report the density of $G_{i,j} = \frac{|E_{i,j}|}{\binom{|V_{i,j}|}{2}}$.

7. Report embeddedness:

   DEFINITION 1.7. *Embeddedness of an edge* (u, v)*, denoted as* $emb(u, v)$*, in a network is the number of common neighbors the two endpoints* u *and* v *have, i.e.,* $emb(u, v) = |N(u) \cap N(v)|$*, where* $N(u)$ *is the number of neighbors of vertex* u *that does not include* u*. [20]*

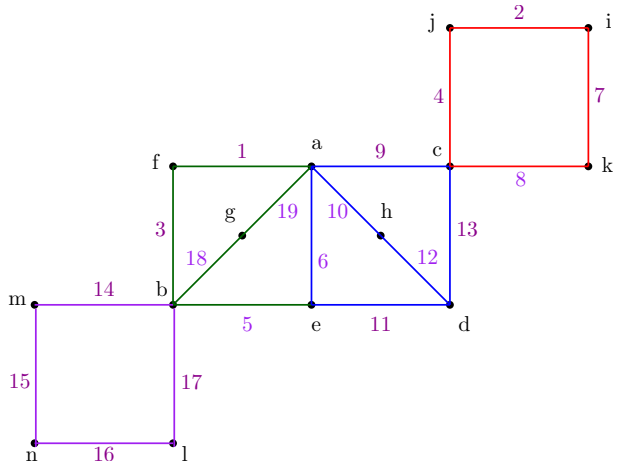


Figure 1.2: The integer numbers on the graphs edges are their timestamps. The implicit representations of the quadrangles in G are as follows $(a, b, \{e, f, g\}), (c, i, \{j, k\}), (c, e, \{a, d, h\}), (b, n, \{\ell, m\})$.

An RE graph G is given. For a query interval $[i, j]$ report the embeddedness of all edges in $G_{i,j}$ with positive embeddedness.

8. Report neighborhood overlap:

DEFINITION 1.8. *Neighborhood overlap of an edge* (u, v)*, denoted as* $\mathrm{NOver}(u, v)$*, is the ratio of the number of vertices who are neighbors of both* u *and* v*, and the number of vertices who are neighbors of only one of them [20].*

$$\mathrm{NOver}(u, v) = \frac{\mathrm{emb}(u, v)}{|N(u) \cup N(v)| - \mathrm{emb}(u, v) - 2}$$

An RE graph G is given. For a query interval $[i, j]$ report the neighborhood overlap of $G_{i,j}$.

DEFINITION 1.9. *Suppose* $G = (V = A \cup B, E)$ *is a bipartite graph. Neighborhood overlap of a pair of vertices* $u, v \in A$ *of* G *is defined as the following ratio [20].*

$$\mathrm{NOverB}(u, v) = \frac{\mathrm{emb}(u, v)}{|N(u) \cup N(v)| - \mathrm{emb}(u, v)}$$

The neighborhood overlap of an entire graph G is defined as the average of the neighborhood overlap values of all the edges of G [33], i.e.,

$$\mathrm{NOver}(G) = \frac{1}{|E|} \sum_{k=1}^{|E|} \mathrm{NOver}(e_k).$$

An (bipartite) RE graph G is given. For a query interval $[i, j]$ report the neighborhood overlap of $G_{i,j}$.

9. Report k-stars:

DEFINITION 1.10. *A k-star is defined to be a complete bipartite graph $K_{1,k}$, i.e., a tree with one internal node and k leaves.*

An RE graph G and an integer k, with $1 \leqslant k \leqslant n$ are given. For a query interval $[i, j]$ report the total number of k-stars in $G_{i,j}$.

10. Report h-index:

DEFINITION 1.11. *The h-index is the largest number h such that the graph contains h vertices of degree at least h. For any graph with m edges, $h = O(\sqrt{m})$ [7].*

An RE graph G is given. For a query interval $[i, j]$ report an $(1+\epsilon)$-approximation of the h-index of $G_{i,j}$, where ϵ is a small value > 0 .

11. Report influenced vertices: Analysis of the *diffusion phenomena* in graphs is a widely studied research area in many communities including computer science, social sciences and epidemiology [30, 31, 32]. The spread of information in any social network requires some influence from a set of designated agents (vertices) and depends on the connectivity of the network. We solve the problem of counting and reporting the influenced vertices in an RE graph slice using the following model.

    Suppose an RE graph $G = (V, E)$ with a fixed set of influential vertices $V' \subseteq V$ and a positive integer r are given. Let $f : V \to N$ be a function assigning thresholds values to vertices such that,

    a) $f(v) = 0$ if v is an influential vertex

    b) $1 \leqslant f(v) \leqslant d(v)$ otherwise, where $d(v)$ is the degree of $v \in V$.

    A vertex can be influenced only if it is on a path of influence. For a pair of vertices u and v, and a positive integer r, a path $\pi = (u = v_1, v_2, v_3, \cdots, v_k, v_{k+1} = v)$ is a path of influence with parameter r, if the following holds:

    a) u is an influential vertex and v is a non-influential vertex

    b) $v_2, v_3, \cdots, v_k$ are influenced vertices

    c) $t(v_i, v_{i+1}) < t(v_{i+1}, v_{i+2})$

    d) $t(v_k, v_{k+1}) - t(v_1, v_2) \leqslant r$.

    DEFINITION 1.12. *A vertex v is influenced with respect to r if either v is an influential vertex, i.e., $v \in V'$, or v is a non-influential vertex ($v \in V \setminus V'$) and there are at least $f(v)$ edge-disjoint paths of influence with parameter r by which v can be reached from some influential vertices.*

    See Figure 3.1 for an example. Here vertices v_1, v_2, v_4 and v_6 are influenced, while vertices v_3 and v_5 are not.

    For a query interval $[i, j]$ report the total number of influenced vertices in $G_{i,j}$.

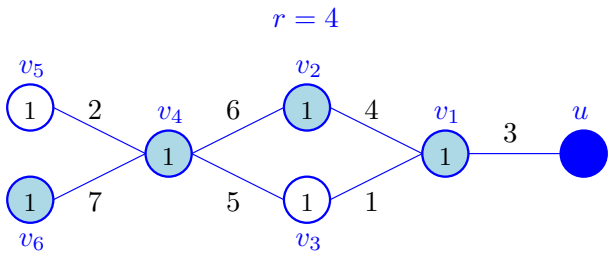


Figure 1.3: A path of influence $\pi = (u, v_1, v_2, v_4, v_6)$ starting from an influential vertex u (shaded dark blue) to vertices v_1, v_2, v_4 and v_6 (shaded light blue) given $r = 4$. The value of $f(v) = 1$ for all $v_k \in V$, with $k = 1, \ldots, 6$. Vertices v_3 and v_5 (opaque) are not influenced by u since there is no path with increasing timestamps from u to these vertices.

1.2.2 *Geometric Problems*

1. Intersection decision problem: A sequence $S = (s_1, \ldots, s_n)$ of n geometric objects (such as, line segments, triangles and convex polygons) is given. For a query interval $[i, j]$ report whether there is any intersection between objects in $S_{i,j}$.

2. Report maximal points:

   DEFINITION 1.13. *For a point $p = (p_x, p_y) \in P$, we define $NE(p)$ to be the set of points in P that lie in the North-East quadrant of p, i.e., $NE(p) = \{q = (q_x, q_y) \in P : q_x > p_x$ and $q_y > p_y\}$ and $SW(p)$ to be the set of points in P that lie in the South-West quadrant of p, i.e., $SW(p) = \{q = (q_x, q_y) \in \mathbb{R}^2 : q_x < p_x$ and $q_y < p_y\}$. A point p is dominated by q if $q \in NE(p)$. A point $p \in P$ is a maximal point if $NE(p) \cap P = \emptyset$.*

   DEFINITION 1.14. *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 $p[k] \leqslant p'[k]$ for $1 \leqslant k \leqslant d$ and $p \neq p'$. For $\gamma > 1$, the γ'th maximal layer L_γ is the set of maximal points in $P - \cup_{l=1}^{\gamma-1} L_l$ [19].*

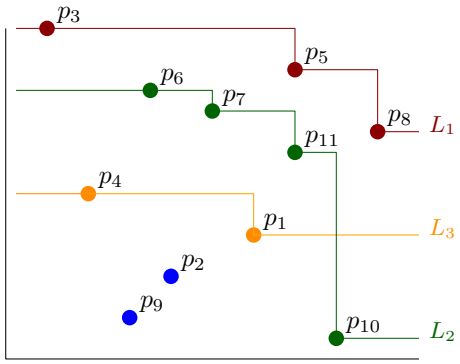


Figure 1.4: Given a sequence of points $(p_1, \ldots, p_{11}) \in \mathbb{R}^2$ this example illustrates maximal layers L_1, L_2, and L_3.

   DEFINITION 1.15. *Given a sequence of n points in $\mathbb{R}^d$ and an integer k, with $1 \leqslant k \leqslant n$, k-dominated points are defined to be the points that are dominated by at least k other points and*

k-dominant points are all maximal points such that each of these points dominates at least k other points.

Consider a sequence $P = (p_1, p_2, \ldots, p_n)$ of n points in $\mathbb{R}^d$, where $d \geqslant 2$. Given a query interval $[i, j]$ solve the following problems.

a) count the number of maximal points in $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$,

b) given an integer k, with $i \leqslant k \leqslant j$, decide if point p_k is on the maximal layer L_γ of $P_{i,j}$, where $\gamma = 1$, or 2, or $\geqslant 3$,

c) for a fixed integer $k \geqslant 1$, report all points in the query interval that are dominated by at least k points of $P_{i,j}$ (i.e., k-*dominated points*),

d) for a fixed integer $k \geqslant 1$, 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*).

3. Window aggregated geometric problem: Let $P = (p_1, p_2, \ldots, p_n)$ be a sequence of n points in $\mathbb{R}^d$, where $d \geqslant 1$ is a fixed dimension.

DEFINITION 1.16. *An aggregate function performs a computation on a set of values and returns a single value.*

Examples of aggregate function are minimum, maximum, summation and average of a set of values.

DEFINITION 1.17. *Let μ be a fixed aggregate function that takes a set of points P in $\mathbb{R}^d$ and assigns a real number $\mu(P) \geqslant 0$ 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.

DEFINITION 1.18. *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 $\mu(P') \geqslant (1 - \epsilon) \cdot \mu(P)$ [4].*

DEFINITION 1.19. *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 $O(f(\epsilon))$ time [34].*

For a query interval $[i, j]$, with $1 \leqslant i < j \leqslant n$, we compute a $(1 + \epsilon)$-coreset of size $f(\epsilon)$ in time $O(f(\epsilon) \log n)$ for the geometric problems that admit some decomposable coresets. For example, the following geometric problems admit decomposable coresets.

Diameter: the maximum distance over all pairs of points in P. *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. *Smallest enclosing disc*: the disc with the minimum radius that contains all the points in P. *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. *Minimum-width annulus*: the minimum width over 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.

4. Report the cost of ℓ-center clustering. Given a query interval $q = [i, j]$, with $1 \leqslant i < j \leqslant n$, and an integer $\ell \geqslant 2$, we report the cost of the ℓ-center clustering of the points in $P_{i,j}$. Suppose $C = \{c_1, c_2, \ldots, c_\ell\}$ is the ℓ-center clustering of $P_{i,j}$. We consider the following cost functions, denoted as $\Phi(C)$, of an ℓ-center clustering; (i) the maximum radius of the minimum enclosing balls of the clusters: $\min \Phi^{max}(C) = \max_{c_\ell \in C} \mathrm{radius}(c_\ell)$, and (ii) the minimum summation of the radius of all balls of the clusters: $\min \Phi^{sum}(C) = \sum_{c_\ell \in C} \mathrm{radius}(c_\ell)$.

1.2.3 *Stochastic Input Sequences*

In this section we deal with input elements that exist with some inherent uncertainty. Various models of uncertainty have been devised to categorize and study these types of elements. We briefly mention the models and some of the relevant studies here.

- *Stochastic points*, where each point p_k has a fixed location which only exists with a probability α_k, with $0 < \alpha_k \leqslant 1$. This model has been studied to solve the minimum spanning tree problem [26], the closest pair and the post office problem [27], the colored dominance problem [38], and the most-likely skyline problem [5].

- *Uncertain points* $P = (p_1, p_2, \ldots, p_n)$, where each point p_k's location is described by a probability distribution α_k. The distribution can be a parametric function such as a uniform distribution or a Gaussian distribution, or may be a non-parametric function such as a histogram. This model has been studied to give randomized approximation algorithm for calculating the distribution of any statistics on uncertain data [25], for computing the probability of each point lying on the skyline [2], and for computing the probabilistic skylines [35].

- *Indecisive points* (also known as *multipoint* model), where each point can take one of a finite number of locations. A simplified model might let each point have exactly r possible locations, that is, each uncertain point p_k is at one of $\{p_{k,1}, \ldots, p_{k,r}\}$ locations and each location is equally likely with probability $1/r$. In this case, each location is assigned a weight $w_{k,\ell}$ as the probability that p_k is at $p_{k,\ell}$ where $\sum_{\ell=1}^{r} w_{k,\ell} = 1$ for all $k = 1, \ldots, n$. This model has been studied for computing the distribution of answers for LP-type problems [25], and for computing the most likely convex hull [3, 37].

- *Imprecise points*, where each point's location is not known precisely but it is restricted to a region. Possible regions can be uncertain intervals (in 1-dimension) or some geometric regions in 2 or higher dimensions. This model has been used to compute the Delaunay triangulations of a set of imprecise points by several authors [23, 28].

In this thesis, we present data structures to solve the following window query problems for stochastic inputs.

1. Report the most likely maximal point (MLMP): Let α_1 and α_2 be constants with $0 < \alpha_1 < \alpha_2 < 1$. A sequence of blue and red stochastic points $P = (p_1, \ldots, p_n)$ in $\mathbb{R}^d$ is given, where

every blue point is associated with probability α_1 and every red point is associated with probability α_2 and $d \geqslant 1$. A point $p = (x_1, \ldots, x_d)$ in P is on the maximal layer of P if there is no other point $q = (x'_1, \ldots, x'_d)$ in P such that $x'_1 > x_1, x'_2 > x_2, \ldots$ and $x'_d > x_d$. Let Z be a random subset of P given by including each blue point of P in Z independently with probability α_1 and each red point of P in Z independently with probability α_2. For a query interval $[i, j]$, with $1 \leqslant i \leqslant j \leqslant n$, we report the point that has the highest probability to be on the *maximal layer* of the queried points $(p_i, \ldots, p_j)$. Later we generalize this problem to c-colored MLMP problem, where $c \geqslant 2$ is a constant.

2. Report the most likely common element (MLCE): Let $\mathcal{U} = \{1, 2, \ldots, n\}$ be the universe. Let $S_1, S_1, \ldots, S_m$ be a sequence of random subsets of $\mathcal{U}$ such that for $p = 1, \ldots, m$ and $i = 1, \ldots, n$, element i is added to S_p with probability α_{pi} (independently of other choices). Let τ be a fixed real number with $0 < \tau \leqslant 1$. For query indices p, q, i and j, with $1 \leqslant p \leqslant q \leqslant m$ and $1 \leqslant i \leqslant j \leqslant n$, decide whether there exists an element k with $i \leqslant k \leqslant j$ such that $\Pr(k \in \cap_{r=p}^{q} S_r) \geqslant \tau$.

1.3 PREVIOUS WORK

A relational event (RE) graph is generally used to represent the communication events between pairs of entities in an underlying network that occurred at some specific times. In recent studies, RE graphs have been used to model social networks for various computational analysis and are also named *dyadic event data* [10] and *contact sequences* [24]. The problem of finding whether an MST exists in a queried graph slice can be directly applied in the design of any type of connected networks in a query time interval, such as, telecommunication, transportation, computer networks or electrical grids. The problem of graph edit distance for spanning forests provides a cost effective measure (i.e., the number of operations required) to maintain the connectivity within the connected components of a network. We find that the applications of subgraph counting data structures are specifically pertinent to some useful social network analyses, for example *clustering coefficient*, which is also known as *network transitivity* and is an important model for extracting community structure from social networks.

We also answer window queries to compute (a) the *density* of a graph, which is one of the basic indicators for measuring graph structure by evaluating how close the graph is to a complete graph, (b) center vertices of k-stars (also known as *hubs*) that are the basis of many tasks, for example web search and epidemic outbreak detection, (c) embeddedness of an edge of the graph that represents the trustworthiness of its neighbors, and the confidence level in the integrity of the transactions that take place between two members connected by this edge, and (d) the *diffusion phenomena* in graphs that has been studied to analyze the cascading failures in physical infrastructure networks, information cascades in social or economic systems, spreads of infectious diseases or ideas, fashions and behavior among people.

The problems of *maximal points* and *maximal layers* simulates the problems in geographic information systems (GIS) and various multi-criteria decision making models in business. We study *stochastic* data sets that originate from real-world data acquisition methods (for example, reading data through sensor networking), various forms of scientific measurements, and subsequent data management techniques (such as cleaning and integrating data sets). These techniques lead to

massive data generation with some inherent uncertainty. The study of finding the *most likely common element* can be applied to identify duplicate elements (e.g., computer files) after a network partition or a backup service query.

Bannister et al. [7] were the first to consider RE graphs for preprocessing into data structures that can answer time windowed queries using timestamped events. They presented data structures that can count number of connected components, number of components containing cycles, number of vertices having degrees equal to some predefined value, and number of influenced vertices on a time-increasing paths [7]. They showed techniques to reduce time windowed problems into a matroid rank problem so that ranks of the query graph slices can be answered by a dominance counting query. Hence they obtained sub-logarithmic query times for time windowed problems. However, the required preprocessing time and space for all problems presented in [7] (see Table 1.1) are the time and space requirements for the reduction only, and do not consider those of the dominance counting data structures.

Table 1.1: Summary of previous results of window queries for relational event graph $G = (V, E)$. Here $n = |V|$, $m = |E|$, $\mathcal{W} = j - i + 1$ is the width of the query window $[i, j]$, r (respectively, f) is the number of the past (respectively, future) neighbors of an edge and h is the h-index (the largest number h such that a graph G contains at least h vertices of degree at least h).

Problem	Preprocessing time	Space	Query time	Reference
Number of (reciprocated) edges, Vertices with k degree,	$O(m+n)$	$O(m+n)$	$O(\log \mathcal{W}/\log\log m)$	[7]
Reachable vertices, Connected components, tree components	$O(m \log n)$	$O(m+n)$	$O(\log \mathcal{W}/\log\log m)$	[7]
Edges with bounded number of neighbors	$O((r+f)m)$	$O(m+n)$	$O(\log \mathcal{W}/\log\log m)$	[7]
Triad closure	$O(hm)$	$O(m+n)$	$O(\log \mathcal{W}/\log\log m)$	[7]

A similar time window model for geometric objects was first studied in [6]. In this paper the authors presented results for reporting the convex hull of points in the plane and the skyline and proximity relations of point sets in $\mathbb{R}^d$. They used a hierarchical decomposition in time and constructed range trees on a given sorted sequence of temporal points to answer the windowed queries related to the convex hulls and proximity relations. For the problems related to the convex hull the data structure can answer queries in polylogarithmic time. For the nearest neighbor queries and for the construction of proximity graphs, the authors provided $(1+\epsilon)$ approximation of the solutions, where ϵ is a small constant. However, their skyline queries use a different preprocessing technique based on the rectangle stabbing data structures. Subsequently more results on window queries have been presented by Bokal et al. [9] and Chan and Pratt [12]. In both these papers the authors mainly focused on answering decision problems for some hereditary properties, such as the convex hull area decision problem (in 2-dimension), the diameter decision problem (in 2-dimension and 3-dimension), the width decision problem (in 2-dimension) and the orthog-

onal segment intersection detection problem. In [9] Bokal et al. 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 $1 \leqslant i \leqslant n$, 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 [12] improved preprocessing times for diameter decision problems and convex hull area decision problems presented in [9]. The authors showed the improved techniques to solve the diameter decision problem in 2-dimension and in 3-dimension, 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 as *FIFO updates* and used this technique to solve the 2-dimensional convex hull area decision problem and the 2-dimensional width decision problem. Table 1.2 summarizes previous results of window queries for geometric problems.

Table 1.2: 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, W is the size of the query window and α is the inverse Ackermann function.

Problems	**Preprocessing time**	**Query time**	**Reference**
Convex hull reporting	$O(n \log n)$	$O(h \log^2 W)$	[6]
Gift wrapping, line stabbing, tangent queries	$O(n \log n)$	$O(\log^2 W)$	[6]
Linear prog., line decision queries	$O(n \log n)$	$O(\log W)$	[6]
Skyline reporting	$O(n^{1+\epsilon})$	$O(w)$	[6]
Spherical range reporting	$O(n \log n)$	$O(\log W + w)$	[6]
Approx. nearest neighbor	$O(n \log n)$	$O(\log W)$	[6]
Diameter decision problem (2D)	$O(n \log^2 n)$	$O(1)$	[9]
	$O(n \log n)$	$O(1)$	[12]
Diameter decision problem (3D)	$O(n \log^2 n)$	$O(1)$	[12]
Convex hull area decision problems	$O(n \log n \log\log n)$	$O(1)$	[9]
	$O(n \alpha(n) \log n)$	$O(1)$	[12]
Monotone paths	$O(n)$	$O(1)$	[9]
Orthogonal segment intersection detection	$O(n \log n \log\log n)$	$O(1)$	[12]
Width decision problem (2D)	$O(n \log^8 n)$	$O(1)$	[12]

1.4 OUR RESULTS

In this section, for convenience, we present our results using *(space, query time)* format denoted by *space-time*, where *space* is the total space (size) required by our data structures and *query time* is the

time required to answer a given window query. In later chapters we also analyze our algorithms for each window query problem and compute the preprocessing time.

1.4.1 *Graph Problems*

In [14, Chapter 2] we consider an RE graph $G = (|V| = n, |E| = m)$ and study three types of window query problems. For all these problems we assume a query has to be answered for a given interval $[i, j]$, with $1 \leqslant i \leqslant j \leqslant m$. The first problem is the *decision problem for monotone graph properties* such as *disconnectedness* and *bipartiteness*. We show that G can be preprocessed to answer whether $G_{i,j}$ is disconnected or bipartite using $(O(m + n \log n), O(\log n))$ space-time. Secondly, we study the window reporting problems for reporting the *minimum spanning tree* using $(O(m \log^{\epsilon} n), O(\log \log n))$ space-time, the minimum spanning interval using $(O(m \log n), O(\log n))$ space-time and the *graph edit distance for forest* using $(O(m \log n), O(\log n + |X|))$ space-time, where $|X|$ is the size of the output. The third window query problem we solve is the *subgraph counting problem*. We show that G can be preprocessed so that the total number of 2-paths and 3-paths in $G_{i,j}$ can be counted using $(O(n), O(m + n))$ and $(O(m + n), O(m + n))$ space-time, respectively; the total number of complete subgraphs (cliques) of some fixed order $\ell \geqslant 3$ in $G_{i,j}$ can be counted using $(O(m + n + \mathcal{K}), O(\log \mathcal{W} + \log \log \mathcal{K}))$ space-time, where $\mathcal{K}$ is the number of complete subgraphs in G_{ij} and $\mathcal{W} = j - i + 1$, and the total number of quadrangles (C_4) in $G_{i,j}$ can be counted using $(O(\alpha(G)m \log n), O(\gamma \log n + w))$ space-time, where $\alpha(G)$ is the arboricity of graph G, $\gamma \leqslant \min\{\binom{n}{2}, \alpha(G)m\}$ and w is the number of quadrangles [13].

In [15, Chapter 3] we consider answering several window queries in an RE graph G by reducing the reporting problems to the colored range searching problems. In particular, we present data structures that can report for a graph slice $G_{i,j}$, the number of vertices it has and its *density* using $(O(m + n), O(\log n))$ space-time, the degrees of all the vertices using $(O(m + n), O(\log^2 n + w))$ space-time, the number of k-stars (hubs) using $(O(m + n), O(\log n + w))$ space-time, the approximation of the h-*index* using $(O(m + n), O(\log^2 n + h \log n))$ space-time, the *embeddedness* using $(O(\alpha(G)m), O(\log^2 n + t \log n))$ space-time, the *neighborhood overlap* using $(O(mn), O(\log^2 n + (t + s) \log n))$ space-time, and the number of influenced vertices using $(O(m \log n), O(\log n + w))$ space-time, where w is the size of the outputs. These results are summarized in Table 1.3.

1.4.2 *Geometric Problems*

In [16, Chapter 4] we study window queries for geometric problems. A sequence S of n geometric objects (such as points, line segments, triangles and convex c-gons) can be preprocessed so that window query can be answered for the intersection decision problem using $(O(n), O(\log n))$ space-time. Consider a sequence of points P in $\mathbb{R}^d$. We show that several counting and decision problems on maximal layers and maximal points can be answered in $(O(n \operatorname{polylog} n), O(\operatorname{polylog} n))$ space-time. The window counting problems include counting the maximal points, k-dominated points and k-dominant points in $P_{i,j}$. The decision problem includes whether a given point p_k, with $1 \leqslant i \leqslant k \leqslant j \leqslant n$, is on the γ-th maximal layer of $P_{i,j}$, where $\gamma = 1, 2$ or $\geqslant 3$.

Lastly we present techniques to approximate window-aggregate queries for $(1 + \epsilon)$-approximations for various geometric problems such as diameter, width, radius of a minimum enclosing ball,

Table 1.3: Summary of our results on the relational event graph $G = (V, E)$. Here $n = |V|$, $m = |E|$, $\mathcal{W} = j - i + 1$ is the width of the query window $[i, j]$, h is the h-index (the largest number h such that the graph contains at least h vertices of degree at least h), z is the size of the output, s is the number of edges having neighboring edges, t is the number of edges that are contained in some triangles, p is the number of vertex pairs having some common neighbors, k is the number of vertices having some neighbors in $G_{i,j}$, ϵ is a small positive constant, X and Y are set of output edges, $a(G) = O(\sqrt{m})$ is the arboricity of G (the minimum number of edge-disjoint spanning forests into which G can be partitioned), $\gamma \leqslant \min\{\binom{n}{2}, a(G)m\}$, ℓ is the order of a complete subgraph, w is the number of a specific subgraph (i.e., 2-path, 3-path (in bipartite graphs), quadrangle, or complete subgraph) in the query interval, and $\mathcal{K}$ is the total number of the same subgraph in G.

Problem	Space	Query time	Reference
Monotone properties			
Bipartiteness	$O(m + n \log n)$	$O(\log n)$	[14, 2.3.2]
Disconnectedness	$O(m + n \log n)$	$O(\log n)$	[14, 2.3.3]
MST	$O(m \log n)$	$O(\log n)$	[14, 2.4.1]
Min spanning interval	$O(m \log^{\epsilon} n)$	$O(\log \log n)$	[14, 2.4.2]
$\mathcal{GED}$ (Forest)	$O(m \log n)$	$O(\log n + \lvert X\rvert)$	[14, 2.4.3]
2-paths	$O(m)$	$O(n)$	[14, 2.5.1]
	$O(m + n + \mathcal{K})$	$O(\log \mathcal{W} / \log \log \mathcal{K})$	[14, 2.5.1]
3-paths	$O(m + n)$	$O(m + n)$	[14, 2.5.1]
	$O(m + n + \mathcal{K})$	$O(\log \mathcal{W} / \log \log \mathcal{K})$	[14, 2.5.1]
Complete Subgraphs (fixed $\ell \geqslant 3$)	$O(m + n + \mathcal{K})$	$O(\log \mathcal{W} / \log \log \mathcal{K})$	[14, 2.5.2]
Complete Subgraphs (all $\ell \geqslant 3$)	$O(m + n + \mathcal{K} \log \mathcal{K} / \log \log \mathcal{K})$	$O((\log \mathcal{K} / \log \log \mathcal{K})^2)$	[14, 2.5.3]
Quadrangles	$O(a(G)m \log n)$	$O(\gamma \log n + w)$	[14, 2.5.4]
Number of vertices,			
Graph density	$O(m + n)$	$O(\log n)$	[15, 3.4.1]
Degrees of vertices	$O(m + n)$	$O(\log^2 n + z)$	[15, 3.4.1]
k-stars	$O(m + n)$	$O(\log n + z)$	[15, 3.4.2]
h-index (approx.)	$O(n \log n)$	$O(\log^2 n + h \log n)$	[15, 3.4.2]
Embeddedness	$O(a(G)m)$	$O(\log^2 n + t \log n)$	[15, 3.4.3]
$\mathrm{NOver}(G_{i,j})$	$O(mn \log n)$	$O(\log^2 n + (t + s) \log n)$	[15, 3.4.3]
$\mathrm{NOver}(G_{i,j})$-bipartite	$O(a(G)m)$	$O(\log^2 n + p \log n + k)$	[15, 3.4.4]
Influenced vertices	$O(m \log n)$	$O(\log n + z)$	[15, 3.4.5]

and volume of the smallest bounding box using $(O(n \log n), O(f(\epsilon) \log n))$ space-time, where $\epsilon > 0$ is a small constant. Given two parameters $\ell \geqslant 2$ and $\epsilon > 0$, we compute the cost of ℓ-center clustering $(\ell \geqslant 2)$ for a sequence of points using coresets of size $O(\ell(f(\ell)/(c\epsilon))^2)$ using $(O(n \log n), O(\ell((f(\ell)/(c\epsilon)) \log n) + \ell(f(\ell)/(c\epsilon))^2 + \mathcal{W}))$ space-time, where $\mathcal{W}$ is the width of the query window and c is a positive constant. The results are summarized in Table 1.4.

Table 1.4: Summary of our results on geometric inputs. n is the number of input objects, $\gamma \geqslant 2$ 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, $\ell \geqslant 2$ is an input parameter, $\epsilon > 0$ and $c > 0$ are small constants.

Problems	Space	Query	Reference
Intersection Decision			
Segment, Bichromatic segment, Triangle, c-gon (c is constant)	$O(n)$	$O(\log n)$	[16, 4.2]
Points on Maximal Layers			
p_k on maximal layer L_1	$O(n \log^{d-2} n)$	$O(1)$	[16, 4.3.1]
p_k on max layer $L_\gamma, \gamma = 2, \geqslant 3$	$O(n \log^{d+1} n)$	$O(\log^{d+1} n)$	[16, 4.3.3]
Count maximal points: $d = 2, 3$	$O(n \log^2 n)$	$O(\log^2 n)$	[16, 4.3.2]
Count maximal points: $d \geqslant 4$	$O(n \log^{d-2} n)$	$O(\log^2 n)$	[16, 4.3.6]
k-dominated points: $2 \leqslant d \leqslant 3$	$O(kn \log n)$	$O(\log^2 n + kw)$	[16, 4.3.4]
k-dominated points: $d \geqslant 4$	$O(kn \log^{d-2} n)$	$O(\log^2 n + kw)$	[16, 4.3.6]
k-dominant points: $2 \leqslant d \leqslant 5$	$O(kn \log^3 n)$	$O(\log^4 n + kw)$	[16, 4.3.5]
k-dominant points: $d \geqslant 6$	$O(kn \log^{d-2} n)$	$O(\log^4 n + kw)$	[16, 4.3.6]
Approximation using Coresets			
Decomposable coresets	$O(n \log n)$	$O(f(\epsilon) \log n)$	[16, 4.4.1]
ℓ-clustering using coresets	$O(n \log n)$	$O(\ell((f(\ell)/(c\epsilon)) \log n) + \ell(f(\ell)/(c\epsilon))^2 + \mathcal{W})$	[16, 4.4.2]

1.4.3 *Stochastic Input Sequences*

In [11, Chapter 5] we investigate two types of window queries for stochastic input sequences.

Let $\alpha_1, \ldots, \alpha_c$ be constants, with $0 < \alpha_1 < \alpha_2 < \ldots < \alpha_c < 1$. Let $P = P_1 \cup P_2 \cup \ldots \cup P_c$ be a set of n points in $\mathbb{R}^d$, for some fixed d. For $r = 1, 2, \ldots, c$, let all points in P_r be colored by the rth color. Furthermore, for $r = 1, 2, \ldots, c$, each point in P_r is associated with probability α_r. Given a query interval $[i, j]$ we can find the point that has the highest probability to be on the maximal layer (most likely maximal point (MLMP)) in $O((n \log n), O(1))$ space-time. We also consider the case where a sequence of stochastic points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^d$ is given such that for $k = 1, 2, \ldots, n$, point p_k exists with probability γ_k, with $0 \leqslant \gamma_k \leqslant 1$, independently of the other points. Given a query interval $[i, j]$, with $i \leqslant j$, and an additional integer $t \in [i, j]$, we can report the probability that the point p_t lies on the maximal layer in $P_{i,j}$ using $(O(n \log^d n), O(\log^d n))$ space-time.

In the second part of this chapter we consider the *most likely common element* problem. Let $\mathcal{U} = \{1, 2, \ldots, n\}$ be the universe. Let $S_1, S_1, \ldots, S_m$ be a sequence of random subsets of $\mathcal{U}$ such that for $p = 1, \ldots, m$ and $i = 1, \ldots, n$, element i is added to S_p with probability α_{pi} (independently of other choices). for a fixed real number τ with $0 < \tau \leqslant 1$ and for query indices p, q, i and

j, with $1 \leqslant p \leqslant q \leqslant m$ and $1 \leqslant i \leqslant j \leqslant n$ we can decide whether there exists an element k with $i \leqslant k \leqslant j$ such that $\Pr(k \in \cap_{r=p}^{q} S_r) \geqslant \tau$ in $(O(mn), O(1))$ space-time and we can report all such elements in $(O(mn), O(\log n + w))$ space-time, where w is the size of the output. We also consider a special case where the elements of $\mathcal{U}$ exist with probability in $\{0, 1\}$ and $\tau = 1$. We solve this problem in $(O(N), O(\log N + w))$ space-time, where N is the total number of elements in all subsets that exist with probability 1 and w is the size of the output.

1.5 OVERVIEW OF OUR APPROACH

We briefly describe the techniques we use to achieve these results. Details are presented in the corresponding chapters.

1.5.1 *Window Problems for Graphs*

In [14, Chapter 2] and [15, Chapter 3] we solve several graph problems by considering RE graphs as the input of the window queries.

Monotone Graph Properties. The first problem we solve in [14, Chapter 2] is a decision problem to verify whether a query graph slice satisfies some given monotone graph properties. A graph property is monotone if it is closed under the removal of edges, such as disconnectedness and bipartiteness.

We present an algorithm to reduce the time windowed decision problems under monotone graph properties $\mathcal{P}$ to the standard range searching problems using dynamic data structures that maintain $\mathcal{P}$. We preprocess edges of G by using a *sliding window* technique and some dynamic data structures that maintain a given monotone graph property $\mathcal{P}$ and require $S(n)$ space, $U(n)$ update time and $Q(n)$ query time. We assume that the data structures accept update operations such as edge insertions and deletions and allow queries that test whether the current graph satisfies $\mathcal{P}$. We preprocess edges of $G = (V, E = \{e_1, e_2, \ldots, e_m\})$ using $\mathcal{D}$ to find all the maximal subsequences of edges that satisfy $\mathcal{P}$ in $O(m \cdot (U(n) + Q(n)))$ time (see Figure 1.5). For each maximal subsequence of edges $e_a, e_{a+1}, \ldots, e_b$ that satisfies $\mathcal{P}$, we store a point $p = (a, b) \in \mathbb{R}^2$.

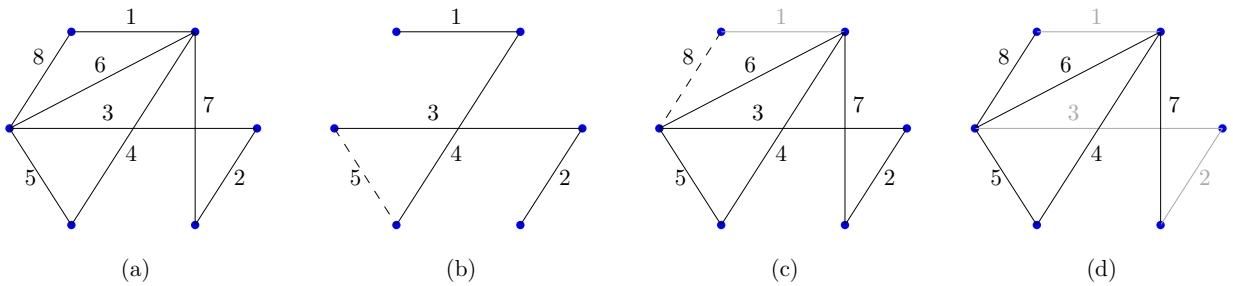


Figure 1.5: (a) An RE graph G with eight edges. Timestamp of each edge is mentioned as an integer value. G has three maximal subsequences of edges that satisfy the disconnectedness monotone property; they are (b) $e_1, \ldots, e_4$ (e_5 connects G), (c) $e_2, \ldots, e_7$ (e_8 connects G), and (d) $e_4, \ldots, e_8$ (preprocessing ends).

We show how to use a 2-dimensional priority search tree (PST), see [29], on the new point set obtained in the preprocessing stage to answer the window query for the monotone property using a grounded query rectangle on this PST in $O(\log n)$ time.

In Chapter 2 we present different variations of the sliding window technique to preprocess the edges of G to reduce the window problems into range search problems and answer window queries for minimum spanning trees, minimum spanning interval and graph edit distance for the forest target class.

Subgraph Counting.

Counting 2-paths. Let an RE graph $G = (V, E)$ with n vertices and m edges be the input. We maintain a set of n lists, one for each vertex $v_k \in V$, where $1 \leqslant k \leqslant n$ and the list L_{v_k} stores timestamps of v_k's incident edges, sorted in increasing order. We show that by using *fractional cascading* data structure, see [17], G can be preprocessed into a data structure in $O(n+m)$ time using $O(n+m)$ space so that total number of 2-paths in the queried subgraph can be counted in $O(n)$ time.

Counting 3-paths in a bipartite RE graph. The input is an RE graph $G = (V = \{A \cup B\}, E = \{(u, v) : u \in A, v \in B\})$. We build two cascading structures for the vertices in A and B that are similar to the data structure described above with few additional information using an array and some pointers. We show that this data structure can be built in $O(n+m)$ time using $O(n+m)$ space so that all 3-paths in the query interval can be counted in $O(n+m)$ time.

Counting all complete subgraphs. We use the edge-scanning algorithm of Chiba and Nishizeki [18] to preprocess the edges of G. During preprocessing we find all the intervals $[low, high]$ that represent the timespans of all the complete subgraphs of a given order ℓ in G. We convert each interval $[low, high]$ into a point $(high, low)$ in $\mathbb{R}^2$. We use dominance counting data structure on the set of all 2-dimensional points $(high, low)$ and show that the number of complete subgraphs of the fixed order ℓ can be counted in $O(\log \mathcal{W} / \log\log \mathcal{K})$ time and the preprocessing requires $O(a(G)^{\ell-2}m)$ time using $O(m+n+\mathcal{K})$ space, where $\mathcal{W}$ is the width of the query window and $\mathcal{K}$ is the number of complete subgraphs of a fixed order ℓ ($\geqslant 3$) in G.

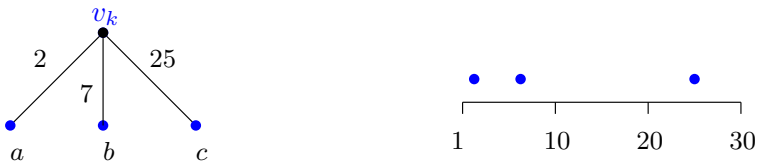
Counting quadrangles (cycles of length 4). We show that using the technique of the implicit representation of a set of quadrangles we build a window data structure based on 2-dimensional range-trees and interval trees to count all quadrangles in a query subgraph. Our technique improves the naive $O(n^4)$ preprocessing time and space to $O(a(G)m\log n)$ time and space so queries can be answered in $O(\gamma \log n + w)$ time, where $\gamma \leqslant \min\{\binom{n}{2}, a(G)m\}$ and w is the number of quadrangles.

Window Queries using Colored Range Searching Data Structures. We present the general approach to solve window problems in RE graphs using colored range searching data structures. Colored range searching problems are variations of the standard range searching problems, where a set of colored objects (e.g., points, lines or rectangles) is given. In the generic instance of the range searching problems, a set of objects S is to be preprocessed into a data structure

so that given a query range q, it can efficiently answer counting or reporting queries based on the intersection of q with the elements in S. In the colored version, the query should efficiently report those colors that have at least one object intersecting the query range.

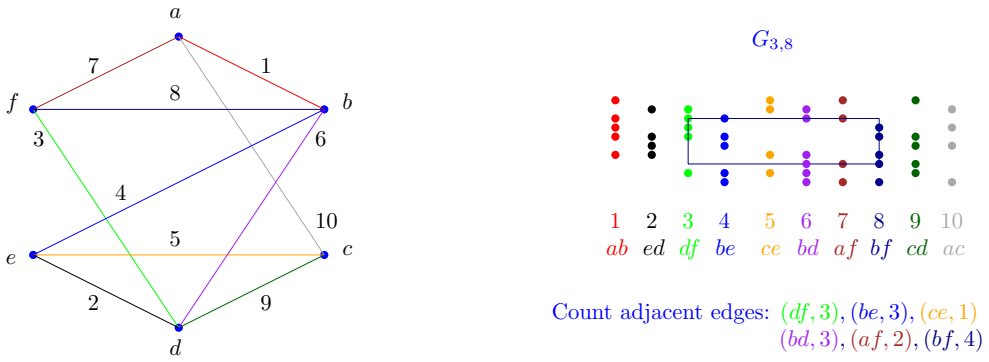
Suppose an RE graph $G = (V, E)$ is given and we want to answer queries about structural parameters of G, such as density, embeddedness, neighborhood overlap etc. To solve each problem, we will define a set of colors $C = \{c_1, c_2, \cdots, c_p\}$, where each color $c_k \in C$ is encoded as an integer in the range $[1, p]$ for some integer p. We process each edge $e = (u, v)$ of G according to the timestamps of the edges in increasing order and scan either each adjacent edge of e or each neighboring vertex of both u and v.

Depending on the problem in hand, our algorithm associates C either to the set of vertices V (i.e., $|C| = n$) or to the set of edges E (i.e., $|C| = m$). When each vertex $v_k \in V$ is associated with a color c_k, the algorithm scans each neighbor N_i of v_k, where $1 \leqslant i \leqslant d(v_k)$, generates a c_k colored point p, and assigns the timestamp $t(v_k, N_i)$ as p's coordinate value (see Figure 1.6(a)). Similarly, when each edge $e_k \in E$ is associated with a color c_k, the algorithm scans each adjacent edge M_i of e_k, and generates a c_k colored point p with the timestamp of M_i assigned as p's coordinate value (see Figure 1.7(b)). This general approach is extended when we report influenced vertices or preprocess bipartite graphs.



(a) (b)

Figure 1.6: (a) A vertex v_k and its three neighbors a, b and c. Since the timestamp $t(v_k, a)$ of the edge (v_k, a) is 2, (v_k, a) is shown as a c_k-colored point on the line with x-coordinate value = 2. (b) The set of all c_k colored points on the line.



(a) (b)

Figure 1.7: (a) Relational event graph G, and (b) Number of adjacent edges in the graph slice $G_{3,8}$.

We summarize the components of our general approach as follows. To solve any problem with our model, we need to specify the following components.

1. The set of colors C and the corresponding graph element (i.e., vertices, edges);
2. The representation of points in $\mathbb{R}^d$, where $d \geqslant 1$;
3. Appropriate colored range searching data structure to answer queries.

In Chapter 3 we present data structures to answer window queries on the graph density, degrees of vertices, embeddedness, neighborhood overlap, k-stars, h-index and influenced vertices.

1.5.2 *Window Problems for Geometric Objects*

In [16, Chapter 4] we present data structures to answer window queries on geometric problems.

Intersecting Objects. We define *valid pairs* and show the technique to reduce the windowed intersection decision problem into a range query problem, given that there exists a data structure that can find all valid pairs. A *valid pair* of indices (α, β), with $1 \leqslant \alpha < \beta \leqslant n$, is defined as follows: For each $1 \leqslant \alpha \leqslant n$, 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 $\beta = \infty$. See Figure 1.8 for an illustration.

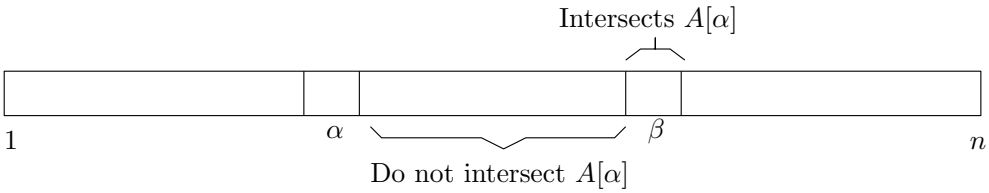


Figure 1.8: A valid pair (α, β).

For each valid pair (α, β), we store a point $(\alpha, \beta) \in \mathbb{R}^2$ using a priority search tree (PST) data structure [29]. 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. We also provide the data structure of size $O(M(n) \log(n))$ in $O(P(n) \log n)$ time so that all valid pairs can be found in $O(n \cdot Q(n) \log n)$ time, where $M(n)$ space, $P(n)$ preprocessing time and $Q(n)$ query time are respectively the space, preprocessing time and query time of a data structure $DS(X)$ that can find whether a query object q intersects any member of a set of n geometric objects X.

Maximal Layers and Maximal Points. Suppose $P = (p_1, p_2, \ldots, p_n)$ is a sequence of n points in $\mathbb{R}^2$. We present data structures to answer various window queries such as counting the total number of maximal points, k-dominants points or k-dominated points or deciding if a given point lies on the maximal layer in the query window. We show that for any k, with $1 \leqslant k \leqslant n$, we can find values for the following two functions (see Figure 1.9)

$$\alpha(k) = \min\{i : i > k \text{ and } p_i \in NE(p_k)\} \text{ and } \beta(k) = \max\{i : i < k \text{ and } p_i \in NE(p_k)\}$$

in $O(n \log n)$ time using $O(n)$ space. Our basic approach for solving all window problems in this section is by mapping a sequence of d-dimensional points to a sequence of higher dimensional points by integrating the information of $\alpha(\cdot)$, $\beta(\cdot)$ and some other values that we discuss in Chapter 4. Lastly we use range searching techniques to answer the window queries.

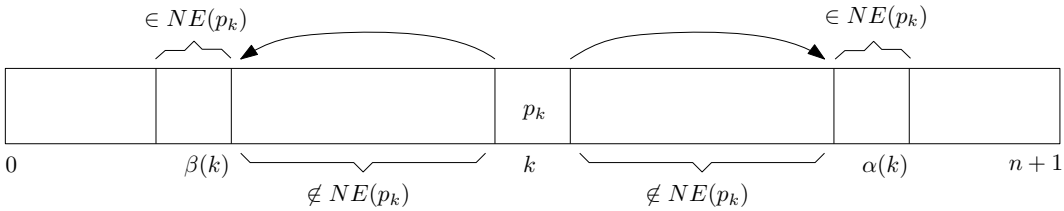


Figure 1.9: $A[\alpha(k)]$ and $A[\beta(k)]$ of a point p_k in array $A[0..n+1]$.

Approximations 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 $\mu(P) \geqslant 0$ 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 $1 \leqslant i < j \leqslant n$ we want to find $\mu(P_{i,j})$.

Our approach: Recall Definition 1.18 of coreset. Agarwal et al. [4] 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 orthogonal to the direction that encloses P.

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 [4]. 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 $f(\epsilon)$ 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 $f(\epsilon)$ (note that the size of the coreset depends on d but not on the size of P). We restate Lemma 23.3 from [21] here.

LEMMA 1.20. *[21, 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 $P_1 \cup P_2$.*

Our basic approach is to divide the sequence of points in P into a sequence of $n/f(\epsilon)$ blocks $S = (S_1, S_2, \ldots S_{n/f(\epsilon)})$ such that each block S_k contains $f(\epsilon)$ 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 $f(\epsilon)$ points belonging to the block S_k, for all $1 \leqslant k \leqslant n/f(\epsilon)$. This approach generates coresets of size $O(\log(n/f(\epsilon)) \cdot f(\epsilon))$ for each canonical node v of T. We show that we can reduce the size of each coreset to $O(f(\epsilon))$ so that the total space required for T becomes $O(n \log n)$.

Our approach for constructing the data structure for the *ℓ-center clustering* using coresets is as follows. We adapt Abrahamsen et al.'s [1] coreset based algorithm that computes a $(1+\epsilon)$-

approximation to the range-clustering query for ℓ-center clustering by using a ϵ-coreset of the input points and constructing a compressed quad-tree for the queried subset of points in time linear to the width of the query window $\mathcal{W} = j - i + 1$ [6]. We build a range tree T on the points of P from left to right according to the increasing order of the sequence and in addition we store (i) the *Morton-order* (or the Z-order) and (ii) a *skip-quadtree* over the points in each of the canonical nodes v of T. We show how to use these data structures and range searching techniques to compute an $(1+\epsilon)$ approximation to the cost of ℓ-center clustering, with $(\ell \geqslant 2)$, using coresets of size $O(\ell(f(\ell)/(c\epsilon))^2)$, where $\epsilon > 0$ is a small number.

1.5.3 *Window Problems for Stochastic Inputs*

In [11, Chapter 5] we study stochastic input sequences in the window query setting to answer a geometric problem to find the c-*colored most likely maximal point (MLMP)* and the *most likely common element (MLCE)*.

Most Likely Maximal Point (MLMP). We consider the 2-colored version here. Let α_1 and α_2 be constants with $0 < \alpha_1 < \alpha_2 < 1$. Let $P = (p_1, \ldots, p_n)$ be a sequence of n points in $\mathbb{R}^d$ with $d \geqslant 1$. Let each point of P be either colored blue or red. Let $B \subseteq P$ be the set of all blue points and let $R \subseteq P$ be the set of all red points. Furthermore, each blue point is associated with probability α_1 and each red point is associated with probability α_2. Note that $P = B \cup R$ and we denote this instance of P as $(B, \alpha_1; R, \alpha_2)$. Let Z be a random subset of P obtained by including each blue point of P in Z independently with probability α_1 and each red point of P in Z independently with probability α_2. The most likely maximal point in the above mentioned instance of P is the point that has the highest probability to be on the maximal layer of P.

Given a query interval $[i, j]$ we define $P_{i,j} = B_{i,j} \cup R_{i,j}$, where $B_{i,j} = B \cap P_{i,j}$ and $R_{i,j} = R \cap P_{i,j}$. We present data structures that can answer window queries for the MLMP problem. We denote this as $\text{MLMP}(B_{i,j}, \alpha_1; R_{i,j}, \alpha_2)$.

Let $D(p; i, j)$ be the number of points in the $\mathcal{D}$-region of a point p in $P_{i,j}$, i.e.,

$$D(p; i, j) = |\mathcal{D}(p) \cap \{p_i, p_{i+1}, \cdots, p_j\}|$$

and

$$\mathcal{D}(p) = \{q = (x'_1, \ldots, x'_d) \text{ of } P_{i,j} : x'_1 > x_1, \ldots, x'_d > x_d\}.$$

We show that there are two cases to consider for answering the MLMP query. Either a blue point that is on the maximal layer of all points in $P_{i,j}$ can be the most likely maximal point in $P_{i,j}$ with probability α_1 or a red point in $R_{i,j}$ with minimum m numbers of blue points to its $\mathcal{D}$-region can be the most likely maximal point in $P_{i,j}$ with probability $\alpha_2(1-\alpha_1)^m$. Thus we report the answer to query Q as $\max\{\alpha_1, \alpha_2(1-\alpha_1)^m\}$. We present solutions for the case where the most likely maximal point is a red point. Observe that if the answer to query Q is a red point with probability $\alpha_2(1-\alpha_1)^m \geqslant \alpha_1$ then $m \leqslant \lfloor \log(\alpha_2/\alpha_1)/\log(1/1-\alpha_1) \rfloor$. Therefore, for a fixed integer λ, with $0 \leqslant \lambda \leqslant n$, we solve query Q': *Given ℓ and r with $1 \leqslant \ell \leqslant r \leqslant n$, is there a red point p in $A[\ell, r]$ for which $D(p; \ell, r) \leqslant \lambda$?*

In Chapter 5 we show how to answer Q' by using the answers of two simpler queries Q'_1 and Q'_2 in $O(1)$ time.

1. Q_1': *For fixed indices* ℓ *and* k *with* $1 \leqslant \ell \leqslant k \leqslant n$, *given an integer* r *with* $r \geqslant k$, *is there a red point* p *in* $A[\ell, \cdots, k]$ *for which* $D(p; \ell, r) \leqslant \lambda$?

2. Q_2': *For fixed indices* k *and* r *with* $1 \leqslant k \leqslant r \leqslant n$, *given an integer* ℓ *with* $\ell \leqslant k$, *is there a red point* p *in* $A[k, \cdots, r]$ *for which* $D(p; \ell, r) \leqslant \lambda$?

Finally to solve Q we build data structures for solving query Q' for $\lambda = 0, 1, \cdots, \beta$, where $\beta = \lfloor \log(\alpha_2/\alpha_1)/\log(1/(1-\alpha_1)) \rfloor$ using $O(n \log n)$ space. Given a query interval $[i, j]$, we query these data structures sequentially and report a red point if it satisfies the condition. Otherwise we report a blue point as the MLMP.

In Chapter 5 we show how to extend this technique to solve the c-colored MLMP problem defined as follows. Let $\alpha_1, \ldots, \alpha_c$ are constants, with $0 < \alpha_1 < \alpha_2 < \ldots < \alpha_c < 1$. Let $P = P_1 \cup P_2 \cup \ldots \cup P_c$ be a set of n points in $\mathbb{R}^d$, for some fixed d. For $r = 1, 2, \ldots, c$, let all points in P_r be colored by the r-th color. Furthermore, for $r = 1, 2, \ldots, c$, each point in P_r is associated with probability α_r. P can be preprocessed into a data structure that can report the most likely maximal point within $P_{i,j}$, with $1 \leqslant i \leqslant j \leqslant n$.

Most Likely Common Element. We consider S to be an $m \times n$ matrix in which each entry $S[p, i]$ is a real number $\alpha_{pi} \in [0, 1]$. We present data structure so that for any query values p, q, i and j, with $1 \leqslant p \leqslant q \leqslant m$ and $1 \leqslant i \leqslant j \leqslant n$, it can decide if there exists some integer k for which $i \leqslant k \leqslant j$ and $\prod_{r=p}^{q} \alpha_{rk} \geqslant \tau$.

Let S' be the matrix of size $m \times n$. For $p = 1, \ldots, m$ and for $i = 1, \ldots, n$, we store $S'[p, i] = r - p + 1$, where $r = \max\{h : \prod_{\ell=p}^{h} \alpha_{\ell i} \geqslant \tau\}$. For each row of S', we build the range maximum data structure [8] so that for any query interval $[p, q]$ and $[i, j]$ it can report the maximum value stored in the $S'[p, i], \ldots, S'[p, j]$ in $O(1)$ time using linear space. To solve the reporting queries, we use mapping techniques so that each entry in S' is mapped to a point in $\mathbb{R}^2$. We further build a priority search tree (PST) (see [29]) on the new set of 2-dimensional points and show the technique of using range searching data structures to answer the MLCE problem in $O(\log n + w)$ time, where w is the size of the output.

1.5.4 *Organization*

The rest of the thesis is organized as follows. Chapter 2 and 3 present data structures to answer window queries on RE graphs. Chapter 4 presents data structures to answer window queries for the sequence of geometric objects. Chapter 5 presents data structures to answer window queries on stochastic inputs.

BIBLIOGRAPHY

[1] Abrahamsen, M., de Berg, M., Buchin, K., Mehr, M., Mehrabi, A.D.: Range-clustering queries. In: 33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017,

Brisbane, Australia. LIPIcs, vol. 77, pp. 5:1–5:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017), https://doi.org/10.4230/LIPIcs.SoCG.2017.5

[2] Afshani, P., Agarwal, P.K., Arge, L., Larsen, K.G., Phillips, J.M.: (approximate) uncertain skylines. Theory Comput. Syst. 52(3), 342–366 (2013), https://doi.org/10.1007/s00224-012-9382-7

[3] Agarwal, P.K., Har-Peled, S., Suri, S., Yildiz, H., Zhang, W.: Convex hulls under uncertainty. Algorithmica 79(2), 340–367 (2017), https://doi.org/10.1007/s00453-016-0195-y

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

[5] Agrawal, A., Li, Y., Xue, J., Janardan, R.: The most-likely skyline problem for stochastic points. In: Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017, July 26-28, 2017, Carleton University, Ottawa, Ontario, Canada. pp. 78–83 (2017)

[6] Bannister, M.J., Devanny, W.E., Goodrich, M.T., Simons, J.A., Trott, L.: 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), http://www.cccg.ca/proceedings/2014/papers/paper02.pdf

[7] Bannister, M.J., DuBois, C., Eppstein, D., Smyth, P.: 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. pp. 856–864 (2013), https://doi.org/10.1137/1.9781611973105.61

[8] Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings. pp. 88–94 (2000), https://doi.org/10.1007/10719839_9

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

[10] Brandes, U., Lerner, J., Snijders, T.A.B.: Networks evolving step by step: Statistical analysis of dyadic event data. In: 2009 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2009, 20-22 July 2009, Athens, Greece. pp. 200–205 (2009), https://doi.org/10.1109/ASONAM.2009.28

[11] Carmi, P., Chanchary, F., Maheshwari, A., Smid, M.: The most likely object to be seen through a window. Submitted to International Journal of Computational Geometry & Applications (2019)

[12] Chan, T.M., Pratt, S.: 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. LIPIcs, vol. 51, pp. 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016), https://doi.org/10.4230/LIPIcs.SoCG.2016.28

[13] Chanchary, F., Maheshwari, A.: Counting subgraphs in relational event graphs. In: WALCOM: Algorithms and Computation - 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9627, pp. 194–206. Springer (2016), https://doi.org/10.1007/978-3-319-30139-6_16

[14] Chanchary, F., Maheshwari, A.: Time windowed data structures for graphs. Journal of Graph Algorithms and Applications 23(2), 191–226 (2019)

[15] Chanchary, F., Maheshwari, A., Smid, M.: Querying relational event graphs using colored range searching data structures. Accepted for publication in Journal of Discrete Applied Mathematics (2019)

[16] Chanchary, F., Maheshwari, A., Smid, M.: Window queries for problems on intersecting objects, maximal points and approximation using core sets. Submitted to Journal of Discrete Applied Mathematics (2018)

[17] Chazelle, B., Guibas, L.J.: Fractional cascading: I. A data structuring technique. Algorithmica 1(2), 133–162 (1986), https://doi.org/10.1007/BF01840440

[18] Chiba, N., Nishizeki, T.: Arboricity and subgraph listing algorithms. SIAM J. Comput. 14(1), 210–223 (1985), https://doi.org/10.1137/0214017

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

[20] Easley, D.A., Kleinberg, J.M.: Networks, Crowds, and Markets - Reasoning About a Highly Connected World. Cambridge University Press (2010), http://www.cambridge.org/gb/knowledge/isbn/item2705443/?site_locale=en_GB

[21] Har-Peled, S.: Geometric approximation algorithms. American Mathematical Soc. (2011)

[22] Harary, F.: Graph theory. Addison-Wesley (1991)

[23] Held, M., Mitchell, J.S.B.: Triangulating input-constrained planar point sets. Inf. Process. Lett. 109(1), 54–56 (2008), https://doi.org/10.1016/j.ipl.2008.09.016

[24] Holme, P.: Network reachability of real-world contact sequences. Physical Review E 71(4), 046119 (2005)

[25] Jørgensen, A., Löffler, M., Phillips, J.M.: Geometric computations on indecisive and uncertain points. CoRR abs/1205.0273 (2012), http://arxiv.org/abs/1205.0273

[26] Kamousi, P., Chan, T.M., Suri, S.: Stochastic minimum spanning trees in euclidean spaces. In: Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011. pp. 65–74. ACM (2011), https://doi.org/10.1145/1998196.1998206

[27] Kamousi, P., Chan, T.M., Suri, S.: Closest pair and the post office problem for stochastic points. Comput. Geom. 47(2), 214–223 (2014), https://doi.org/10.1016/j.comgeo.2012.10.010

[28] Löffler, M., Snoeyink, J.: Delaunay triangulation of imprecise points in linear time after preprocessing. Comput. Geom. 43(3), 234–242 (2010), https://doi.org/10.1016/j.comgeo.2008.12.007

[29] McCreight, E.M.: Priority search trees. SIAM J. Comput. 14(2), 257–276 (1985), https://doi.org/10.1137/0214021

[30] L. M. Bettencourt, A. Cintrón-Arias, D. I. Kaiser, C. Castillo-Chávez, The power of a good idea: Quantitative modeling of the spread of ideas from epidemiological models, Physica A: Statistical Mechanics and its Applications 364 (2006) 513–536.

[31] D. Centola, M. Macy, Complex contagions and the weakness of long ties, American Journal of Sociology 113 (3) (2007) 702–734.

[32] L. Gargano, P. Hell, J. G. Peters, U. Vaccaro, Influence diffusion in social networks under time window constraints, Theor. Comput. Sci. 584 (2015) 53–66. doi:10.1016/j.tcs.2015.02.015.
URL https://doi.org/10.1016/j.tcs.2015.02.015

[33] Meghanathan, N.: A greedy algorithm for neighborhood overlap-based community detection. Algorithms 9(1), 8 (2016), https://doi.org/10.3390/a9010008

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

[35] Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007. pp. 15–26. ACM (2007), http://www.vldb.org/conf/2007/papers/research/p15-pei.pdf

[36] Santoro, N., Quattrociocchi, W., Flocchini, P., Casteigts, A., Amblard, F.: Time-varying graphs and social network analysis: Temporal indicators and metrics. CoRR abs/1102.0629 (2011), http://arxiv.org/abs/1102.0629

[37] Suri, S., Verbeek, K., Yildiz, H.: On the most likely convex hull of uncertain points. In: Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings. pp. 791–802. Springer (2013), https://doi.org/10.1007/978-3-642-40450-4_67

[38] Xue, J., Li, Y.: Colored stochastic dominance problems. CoRR abs/1612.06954 (2016), http://arxiv.org/abs/1612.06954

2

TIME WINDOWED DATA STRUCTURES FOR GRAPHS

We present data structures that can answer *time windowed* queries for a set of timestamped events in a relational event graph. We study the relational event graph as input to solve (a) time windowed decision problems for monotone graph properties, such as disconnectedness and bipartiteness, and (b) time windowed reporting problems such as reporting the minimum spanning tree, the minimum time interval, and the graph edit distance for obtaining spanning forests. We also present results of window queries for counting subgraphs of a given pattern, such as paths of length 2 (in general graphs) and paths of length 3 (in bipartite graphs), quadrangles and complete subgraphs of a fixed order or of all orders $\ell \geqslant 3$ (i.e., cliques of size ℓ). These query results can be used to compute graph parameters that are important for social network analysis, e.g., clustering coefficients, embeddedness and neighborhood overlapping.

This chapter is a combination of results that have been presented in the 10th International Workshop on Algorithms and Computation (WALCOM 2016) [9] and results that have been published in the Journal of Graph Algorithms and Applications [10].

2.1 INTRODUCTION

A *relational event (RE) graph* $G = (V, E = \{e_1, e_2, \ldots, e_m\})$ is an undirected graph with a fixed set V of n vertices and a sequence of edges E between pairs of vertices, where each edge has a unique positive timestamp [3]. Without loss of generality, we assume that $t(e_1) < t(e_2) < \ldots < t(e_m)$, where $t(e_i)$ is the timestamp of the edge e_i. In a *time windowed query* we are given a relational event graph G and a predicate $\mathcal{P}$. We want to preprocess G into a data structure such that given a query time interval $q = [i, j]$ with $1 \leqslant i < j \leqslant n$, it can answer time windowed queries based on the *graph slice* $G_{i,j} = (V, E_{i,j} = \{e_i, e_{i+1}, \ldots, e_j\})$ that matches $\mathcal{P}$.

This paper follows an *event based* approach [2], where each relational event (graph edge) appears at a specific instance in time, such that at some time k there exists at most one event, i.e., the edge e_k. Therefore, when we consider an interval $[i, j]$ in time, we get a set of events $\{e_i, \ldots, e_j\}$ whose timestamps fall in that interval. As mentioned above, given a query time interval our data structures answer queries using only the set of events that exist in this time interval.

In this paper, we present new results for time windowed decision problems on monotone properties of relational graph events, such as disconnectedness and bipartiteness, and problems on minimum spanning trees (MSTs) within a given query interval, e.g., reporting the existence of an MST of G, the minimum spanning interval (i.e., minimum time required to obtain an MST of G), and the graph edit distance ($\mathcal{GED}$) to convert a graph slice into a spanning forest. We also present window data structures for counting all occurrences of subgraphs that match with a given pattern, e.g., paths of length 2 (2-paths), paths of length 3 (3-paths), quadrangles (cycles of length 4) and all complete subgraphs of a given order ℓ, with $\ell \geqslant 3$. The triangle counting problem is fundamental to many graph applications. This problem has been studied in various contexts,

for example as a base case for counting complete subgraphs of given orders [26], in minimum cycle detection problem [24], and as a special case of counting given length cycles [1]. Counting the total number of triangles and quadrangles are also essential for analyzing large networks (such as WWW, social networks, bipartite graphs or two-mode networks) as they are used to compute important network structures, such as clustering coefficients [34, 35] and transitivity coefficients [20]. Complete subgraphs counting problems have applications in combinatorics and network analysis (see e.g., [32, 33]). We also present some applications to show how various graph parameters can be computed using subgraph counting data structures that are particularly useful for social network analysis.

2.1.1 *Previous Work*

A relational event (RE) graph generally represents communication events between pairs of entities in an underlying network that occurred at some specific times. In recent studies, RE graphs have been used to model social networks for various computational analysis and are also named *dyadic event data* [6] and *contact sequences* [23]. Bannister et al. [3] were the first to consider this model for preprocessing into data structures that can answer time windowed queries using timestamped events. They presented data structures that can count number of connected components, number of components containing cycles, number of vertices having degrees equals to some predefined value, and number of influenced vertices on a time-increasing paths [3]. They showed techniques to reduce time windowed problems into a matroid rank problem so that ranks of the query graph slices can be answered by a dominance counting query. Hence they obtained sublogarithmic query times for time windowed problems. However, the required preprocessing time and space for all problems presented in [3] (see Table 2.1) are the time and space requirements for the reduction only, and do not consider those of the dominance counting data structures.

Chanchary et al. [11] presented techniques for building time window data structures for RE graphs by using a colored range searching approach [5, 17, 18]. Their results include reporting several graph parameters that are essential for social network analysis, such as graph density, h-index, k-stars, embeddedness, neighborhood overlap (for both general and bipartite graphs) and the total number of influenced vertices in the queried graph slices (see Table 2.1).

Recently, window queries have been extended towards solving geometric problems such as reporting skyline and proximity relations of point sets [2], finding all maximal subsequences that hold some hereditary property for a set of points [4], solving convex hull area decision problem, diameter and width decision problem [8], intersection decision problems for objects (e.g., line segments, triangles and convex c-gons) and problems related to dominant points and maximal layers [12].

2.1.2 *New Results*

The main contributions of this paper are listed below, and are also summarized in Table 2.1. Let G be a (weighted) relational event graph consisting of m edges and n vertices.

1. *Decision problems on monotone graph properties*: A graph property $\mathcal{P}$ is called monotone if every subgraph of a graph with property $\mathcal{P}$ also has property $\mathcal{P}$. Given a dynamic algorithm

$\mathcal{D}$ that maintains a monotone graph property $\mathcal{P}$ (e.g., disconnectedness and bipartiteness) having update time $U(n)$, query time $Q(n)$, and space $S(n)$, the time windowed decision problem can be reduced to a 2-dimensional range searching query in time $O(m \cdot (U(n) + Q(n)))$ using space $O(S(n) + m)$ that can answer queries in time $O(\log n)$.

2. *Problems on minimum spanning trees*: Given a weighted RE graph G that has an MST with weight ω^*:
   - We can preprocess G into a data structure of size $O(m \log n)$ such that given a query time interval $[i, j]$ we can report whether there exists an MST $T = (V, E')$ of G with weight ω^* such that $E' \subseteq \{e_i, \cdots, e_j\}$ in $O(\log n)$ time. Preprocessing requires $O(m \log^4 n)$ time.
   - We can preprocess G into a data structure of size $O(m \log^\epsilon n)$, where ϵ is a small constant, such that given a query time interval $[i, j]$ we can report the minimum spanning interval (i.e., the smallest time interval $[\alpha, \beta]$ such that $i \leqslant \alpha$, $\beta \leqslant j$ and $\beta - \alpha$ is the smallest time required to obtain an MST of G) in $G_{i,j}$ in $O(\log \log n)$ time. Preprocessing requires $O(m \log^4 n)$ time.
   - We can preprocess G into a data structure of size $O(n + m)$ such that given a query time interval $[i, j]$ we can report the graph edit distance ($\mathcal{GED}$) to convert $G_{i,j}$ into a spanning forest of G in $O(\log n + |X|)$ time, where X is the set of the minimum number of edges such that $G_{i,j} \setminus X$ is a forest. Preprocessing requires $O(m \log n)$ time.
3. *Problems on counting subgraphs*: We can preprocess G into time windowed data structures so that given a query time interval $[i, j]$ we can count the total number of the following subgraphs in $G_{i,j}$.
   - *Paths of length 2 (2-paths)*: Preprocessing takes $O(n + m)$ time using $O(n + m)$ space, and queries can be answered in $O(n)$ time.
   - *Paths of length 3 (3-paths) (in bipartite graphs)*: Preprocessing takes $O(n + m)$ time using $O(n + m)$ space, and queries can be answered in $O(n + m)$ time.
   - *Complete subgraphs of a fixed order $\ell \geqslant 3$*: Preprocessing takes $O(a(G)^{\ell-2}m)$ time and $O(m + n + \mathcal{K})$ space, and queries can be answered in $O(\log \mathcal{W} / \log \log \mathcal{K})$ time, where $\mathcal{W}$ is the width of the query window and $\mathcal{K}$ is the total number of complete subgraphs of a fixed order in G.
   - *Complete subgraphs of all orders $\ell \geqslant 3$*: Preprocessing takes $O(a(G)^{\ell-1}m)$ time and $O(m + n + \mathcal{K} \log \mathcal{K} / \log \log \mathcal{K})$ space, and queries can be answered in $O((\log \mathcal{K} / \log \log \mathcal{K})^2)$ time, where $\mathcal{K}$ is the total number of complete subgraphs in G of orders 3 and more.
   - *Quadrangles*: Preprocessing takes $O(a(G)m \log n)$ time and $O(a(G)m \log n)$ space. Queries can be answered in $O(\gamma \log n + w)$ time, where $\gamma \leqslant \min\{\binom{n}{2}, a(G)m\}$ and w is the number of reported quadrangles.

2.1.3 *Organization*

The rest of this paper is organized as follows. Section 2 provides preliminaries to the paper. Sections 3 and 4 present results of time windowed data structures for problems on monotone

graph properties and on minimum spanning trees, respectively. In section 5 we present window data structures for subgraph counting problems for paths, complete subgraphs and quadrangles. In Section 6 we present some applications of window data structures. Section 7 concludes the paper.

2.2 PRELIMINARIES

2.2.1 *Relational Event Graph*

A relational event (RE) graph G is defined to be a simple graph with a set of n vertices V and a set of m edges (or relational events) $E = \{e_k \mid 1 \leqslant k \leqslant m\}$ between pairs of vertices. We assume that the graph is undirected so the pairs of vertices in the set of edges are unordered. We also assume that each edge or relational event has a unique timestamp. We denote the timestamp of an edge $e_k \in E$ by $t(e_k)$. Without loss of generality, we assume that $t(e_1) < t(e_2) < \ldots < t(e_m)$, and that the timestamps follow the sequence $1, 2, \ldots, m$.

Given a relational event graph G, for a pair of integers $1 \leqslant i < j \leqslant m$, we define the *graph slice* $G_{i,j} = (V, E_{i,j} = \{e_i, e_{i+1}, \ldots, e_j\})$. $N_{i,j}(v)$ denotes the set of neighbors of vertex v in $G_{i,j}$ and $deg_{i,j}(u)$ is the number of edges adjacent to vertex u in $G_{i,j}$. Figure 2.1(a) illustrates an RE graph G with 5 edges. Figures 2.1(b) and 2.1(c) represent, respectively, the graph slices $G_{2,4}$ and $G_{2,5}$.

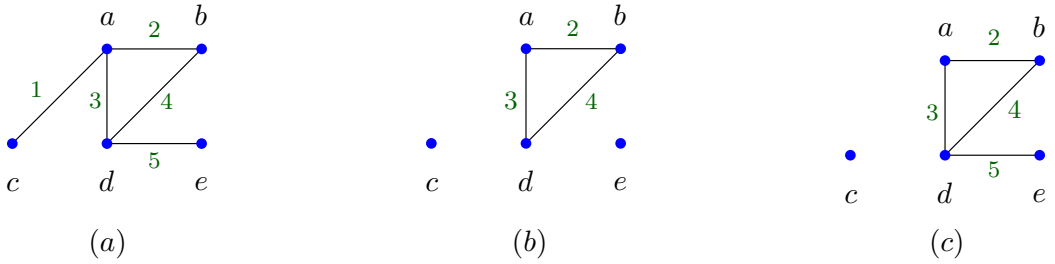


Figure 2.1: (a) An RE graph G with five edges (the integer numbers on the edges are their timestamps), (b) a graph slice $G_{2,4}$ and (c) a graph slice $G_{2,5}$. Examples of open triplets: $\langle bd, de\rangle$, $\langle ad, de\rangle$ in G and $G_{2,5}$; $\langle ca, ab\rangle$, $\langle ca, ad\rangle$ in G. Examples of closed triplets: $\langle ab, bd, da\rangle$ in G, $G_{2,4}$ and $G_{2,5}$.

A *triplet* is a connected subgraph consisting of three distinct vertices that are connected by either three edges (*closed triplet*) or two edges (*open triplet*). Note that any triangle consists of three closed triplets, one centered on each of the vertices. Figure 2.1(a) shows a triangle (a, b, d) consisting of three open triplets with time stamps $\langle 2, 3\rangle$, $\langle 2, 4\rangle$ and $\langle 3, 4\rangle$ where each of these triplets are closed by a third edge with timestamps 4, 3 and 2, respectively.

A *high* (respectively, *low*) event in any RE graph slice $G_{i,j}$ is defined to be the timestamp of an edge e_k, where e_k is the edge with the highest (respectively, the lowest) timestamp among all edges that form a particular subgraph in $G_{i,j}$, such as paths of a fixed length, quadrangles or complete subgraphs. We refer to Figure 2.2 and suppose we are interested in counting how many quadrangles (C_4) are in the query slice $[i, j]$. For a given slice $G_{1,7}$, edges $\{e_1, .., e_7\}$ do not form any C_4, thus no *high* or *low event* occurs in this slice. However, edges e_2, e_4, e_7 and e_8 create a $C_4 = (e_2, e_4, e_7, e_8)$ in $G_{1,8}$. Therefore, edges e_9 and e_{13} become the *low* and the *high event* respectively for this C_4. It is possible that some edges participate in multiple subgraphs and thus

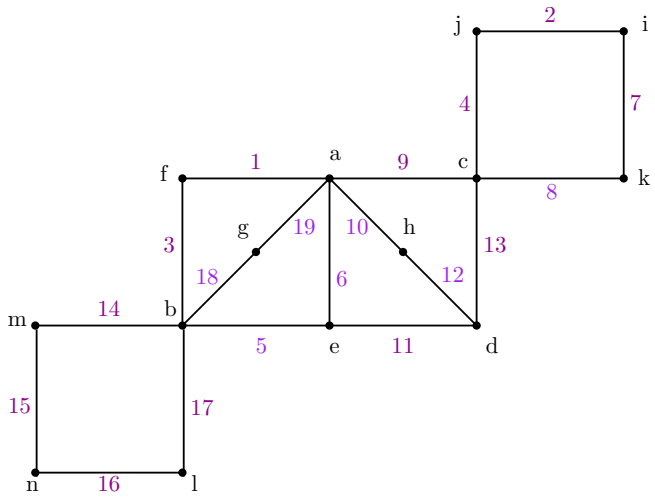


Figure 2.2: An RE graph with 14 vertices and 19 edges. The timestamps are mentioned as the integer numbers on the graphs edges.

become *high* or *low events* for more than one subgraphs in the same slice. There are three C_4's in $G_{6,13}$, $(e_6, e_9, e_{11}, e_{13})$, $(e_6, e_{10}, e_{11}, e_{12})$ and $(e_9, e_{10}, e_{12}, e_{13})$. For two of these subgraphs, e_{13} is the *high event*, though each of them have different *low events*.

Throughout the paper, adjacency linked lists are used to represent an RE graph G. There will be two copies of each edge (u, v) for each endpoints u and v. Each node of the linked list for u stores its neighbour v and the timestamp of the edge (u, v).

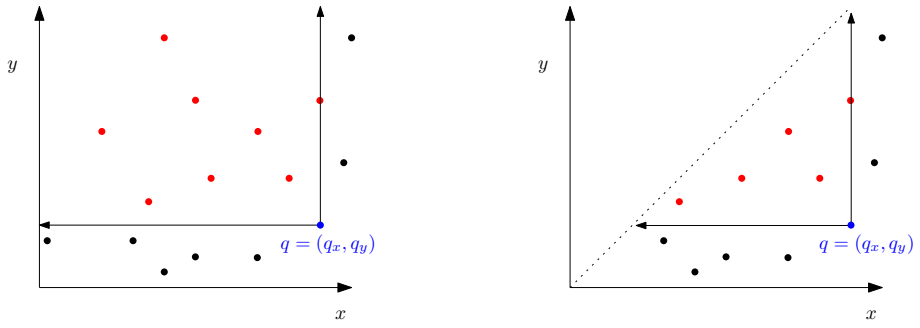
Arboricity $a(G)$ of a graph $G = (V, E)$ having $m = |E|$ edges and $n = |V|$ vertices is the minimum number of edge-disjoint spanning forests into which G can be partitioned [19]. Chiba and Nishizeki [14] gave an upper bound on $a(G)$ for a general graph G as $a(G) \leqslant \lceil (2m + n)^{1/2}/2 \rceil$. Thus, for a connected graph G, $a(G) = O(\sqrt{m})$. We state the following lemma from their paper as this result will be used later in the section for subgraph counting.

LEMMA 2.1. *[14, Lemma 2.1] If a graph* $G = (V, E)$ *has a set of* n *vertices and* m *edges, then* $\sum_{(u,v)\in E} \min\{deg(u), deg(v)\} \leqslant 2a(G)m$, *where* $deg(x)$ *denotes the degree of vertex* x *in* G.

2.2.2 *Geometric Data Structures*

The d-dimensional dominance counting problem for a set S of d-dimensional points is to store S in a data structure such that given a query point q the points in S that are dominated by q can be counted quickly. Let $p = (p_x, p_y)$ and $q = (q_x, q_y)$ be two points in plane. We say q *dominates* p, if $p_x \leqslant q_x$ and $q_y \leqslant p_y$. Note that this is a non-traditional definition of dominance. The standard dominance counting data structure considers dominance relationship to be $q_x \geqslant p_x$ and $q_y \geqslant p_y$, see Lemmas 1 and 2 in [25].

We can use this data structure to query for dominated points by mirroring the coordinates of the points. Given a point set S with n points in plane, the dominance counting query is to determine the total number of points of S dominated by a query point q. See Figure 2.3(a) for an example.



(a) (b)

Figure 2.3: (a) All red points are dominated by a query point q. A red point $p = (p_x, p_y)$ is dominated by $q = (q_x, q_y)$, where $p_x \leqslant q_x$ and $q_y \leqslant p_y$. (b) A special case where all points are below the main diagonal of a grid.

We state the following results from [15] and [25] on dominance counting, range tree, interval tree and priority tree data structures.

THEOREM 2.2. *[25, Theorem 2] Let S be a set of n d-dimensional points, where $d \geqslant 2$ is a constant. Then there exists a data structure for the d-dimensional dominance counting problem using $O(n(\log n/\log\log n)^{d-2})$ space such that queries can be answered in $O((\log n/\log\log n)^{d-1})$ time.*

The following result is from [3] that presents a data structure for window sensitive dominance queries on a set of n points, where the integer coordinates of each point are in the range from 1 to n. Here, it is assumed that the points are below the main diagonal, see Figure 2.3(b).

THEOREM 2.3. *[3, Theorem 9] Let S be a set of $O(n)$ points below the main diagonal of an $n \times n$ grid, then there exists a data structure of size $O(n)$ that can perform dominance queries for which the query point (i, j) is at distance $d = (j - i)\sqrt{2}$ from the main diagonal in time $O(\log d/\log\log n)$ time, with $1 \leqslant i < j \leqslant n$.*

A 2-dimensional *range tree* is a data structure for rectangular range queries, where each query is composed of two 1-dimensional sub-queries respectively on the x-coordinates and on the y-coordinates of the points.

THEOREM 2.4. *[15, Theorem 5.11] Let P be a set of n points in d-dimensional space, with $d \geqslant 2$. A range tree for P uses $O(n \log^{d-1} n)$ space and it can be constructed in $O(n \log^{d-1} n)$ time. With this range tree one can report the points in P that lie in a rectangular query range in $O(\log^{d-1} n + k)$ time, where k is the number of reported points.*

An *interval tree* data structure stores a set of n axis-parallel line segments or intervals on the line.

THEOREM 2.5. *[15, Theorem 10.4] An interval tree for a set of n intervals uses $O(n)$ storage and can be built in $O(n \log n)$ time. Using the interval tree we can report all intervals that contain a query point in $O(\log n + k)$ time, where k is the number of reported intervals.*

A *priority search tree* data structure stores 2-dimensional points and performs range queries that are unbounded on one side.

THEOREM 2.6. *[15, Theorem 10.9] A priority search tree for a set of n points in the plane uses $O(n)$ storage and can be built in $O(n \log n)$ time. This structure can report all points in a query range of the form $(-\infty, q_x] \times [q_y, q'_y])$ in $O(\log n + k)$ time, where k is the number of reported points.*

2.3 MONOTONE GRAPH PROPERTIES

In this section, we present data structures to solve *time windowed decision problems* under some monotone graph property $\mathcal{P}$.

DEFINITION 2.7. *A graph property $\mathcal{P}$ is monotone if every graph slice of a graph with property $\mathcal{P}$ also has property $\mathcal{P}$.*

In other words, a graph property is monotone if it is closed under the removal of edges. For example, every subgraph of a planar graph is planar. Other examples of monotone graph property includes disconnectedness and bipartiteness.

Problem statement: Given an RE graph $G = (V, E)$ and a monotone graph property $\mathcal{P}$, we want to preprocess G so that for any query time interval $[i, j]$ with $1 \leqslant i < j \leqslant m$, we can answer quickly whether the graph slice $G_{i,j} = (V, E_{i,j})$ satisfies $\mathcal{P}$.

2.3.1 *Overview of the Algorithm*

We present a general approach to reduce the time windowed decision problems under monotone graph properties $\mathcal{P}$ to the standard range searching problems using dynamic data structures that maintain $\mathcal{P}$. Suppose, there exists a dynamic data structure $\mathcal{D}$ that maintains some monotone graph property $\mathcal{P}$ and requires $S(n)$ space, $U(n)$ update time and $Q(n)$ query time. We further assume that, $\mathcal{D}$ accepts update operations such as edge insertions and deletions and allows queries that test whether the current graph satisfies $\mathcal{P}$. We preprocess edges of $G = (V, E = \{e_1, e_2, \ldots, e_m\})$ using $\mathcal{D}$ to find all the maximal subsequences of edges that satisfy $\mathcal{P}$. To be more specific, starting with e_1, we insert edges of E according to the increasing timestamps of edges to $\mathcal{D}$ and check whether the subgraph formed by edges added so far satisfies property P. If it does, we continue adding edges to $\mathcal{D}$ until for the first time we find a graph slice $G_{1,b+1}$ that does not satisfy P for some $b \geqslant 1$ (see Figure 2.4).

Now, for this first maximal subsequence of edges $e_1, e_2, \ldots, e_b$ that satisfies $\mathcal{P}$, we store a point $p = (1, b) \in \mathbb{R}^2$. Next, we keep deleting edges from $\mathcal{D}$ starting with e_1 and following the same sequence of edges as they have been inserted until we find some e_a such that $e_a, e_{a+1}, \ldots, e_{b+1}$ satisfies $\mathcal{P}$. To find the next maximal subsequence, we start inserting edges from e_{b+2} and repeat the process. We continue processing edges until we scan all m edges of G. At the end of this process, we will have a set of points S in plane, where each point $(a, b) \in S$ represents a graph slice having a maximal contiguous edge set $\{e_a, e_{a+1}, \ldots, e_b\}$ that satisfies property $\mathcal{P}$. During preprocessing, each edge of G is updated (inserted and deleted) and queried exactly once using $\mathcal{D}$. So the total time required for this preprocessing step is $O(m \cdot (U(n) + Q(n)))$.

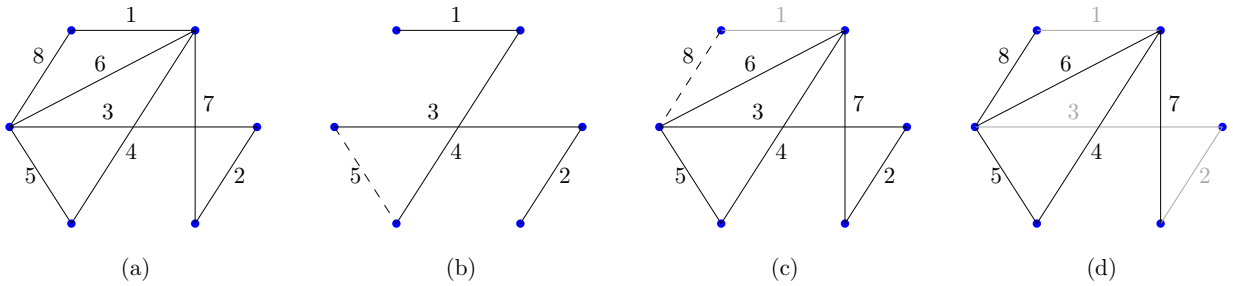


Figure 2.4: (a) An RE graph G with eight edges. Timestamp of each edge is mentioned as an integer value. G has three maximal subsequences of edges that satisfy the disconnectedness monotone property; they are (b) $e_1, \ldots, e_4$ (e_5 connects G), (c) $e_2, \ldots, e_7$ (e_8 connects G), and (d) $e_4, \ldots, e_8$ (preprocessing ends).

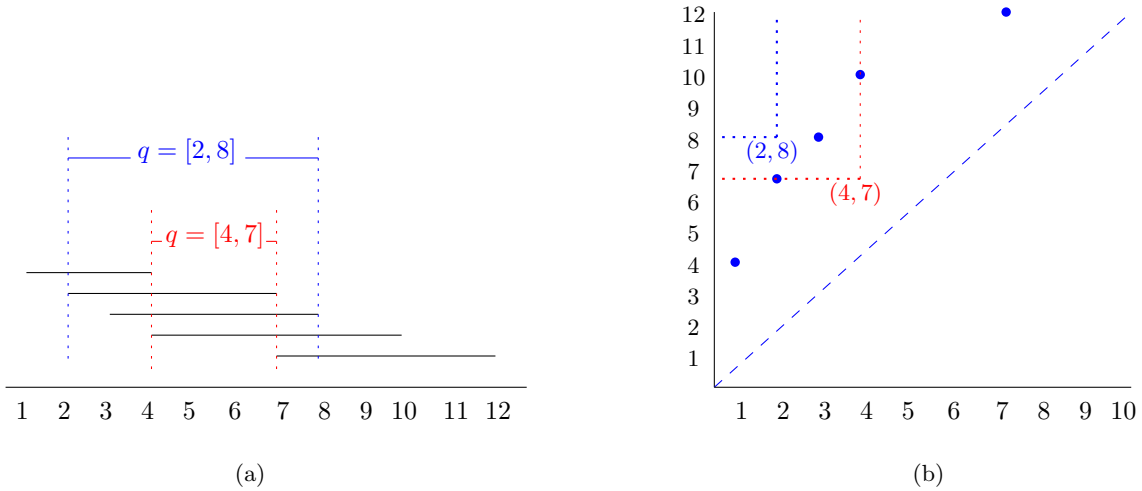


Figure 2.5: (a) An example illustrating a set of time intervals representing graph slices that satisfy some monotone property $\mathcal{P}$, (b) Query: for $q = [2, 8]$ (blue line) there is no point in $(-\infty, 2] \times [8, +\infty)$, hence $G_{2,8}$ does not satisfy $\mathcal{P}$, and for $q = [4, 7]$ (red line) there are points in $(-\infty, 4] \times [7, +\infty)$, hence $G_{4,7}$ satisfies $\mathcal{P}$.

Each point $(a, b) \in S$ obtained from this preprocessing step represents a time interval $[a, b]$, where $b > a$, such that $G_{a,b}$ satisfies $\mathcal{P}$ (see Figure 2.5(a)). Let I_S be the set of all intervals found from S in this way. A query time interval $[i, j]$ satisfies $\mathcal{P}$ if and only if $[i, j]$ is contained in some interval $[a, b] \in I_S$. We define the North-West quadrant of the point (i, j) as $NW(i, j) = (-\infty, i] \times [j, \infty)$. Now we reduce this problem to range emptiness problem as stated in the following lemma.

LEMMA 2.8. *A query time interval* $[i, j]$ *is contained in some interval* $[a, b] \in I_S$ *if and only if* $NW(i, j) \cap S \neq \emptyset$.

Proof. For the '*if*' part, note that all points $(a, b) \in S$ are above the main diagonal as $b > a$. Therefore, when some query interval $[i, j]$ is contained in $[a, b] \in S_I$, it implies that $a \leqslant i < j \leqslant b$ and $(a, b) \in NW(i, j)$ (see Figure 2.5(b)).

Now we prove the '*only if*' part. First we observe that I_S can never contain two subsequences such that one is totally contained in another. See Figure 2.6(a). Our algorithm will always keep

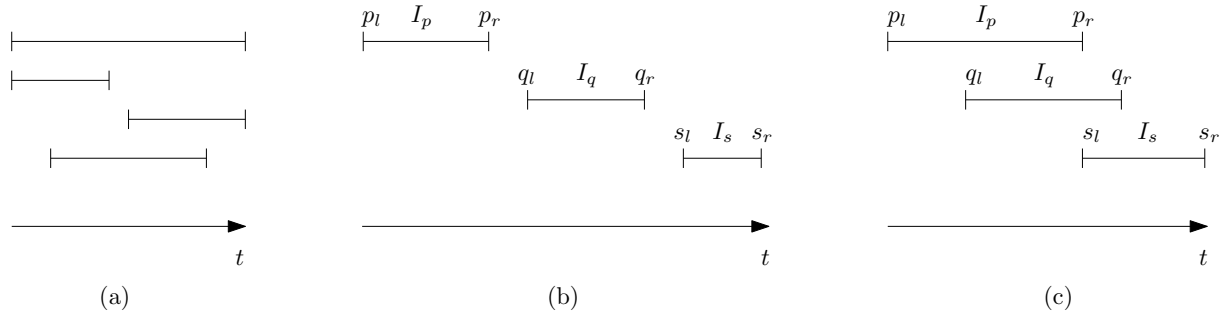


Figure 2.6: (a) Subsequences cannot be contained in other maximal subsequences. The permissible orientations of subsequences can be either (b) Non-overlapping subsequences, or (c) Partially overlapping subsequences.

the subsequence with the maximum length. Moreover, there can be only two possible orderings of all subsequences of I_S where two maximal subsequences will either partially overlap each other or they do not overlap at all, see Figures 2.6(b) and (c).

Suppose our algorithm does not identify a valid maximal subsequence I_q. Two cases have to be verified here.

Case (a): when $I_q = (e_{q_\ell}, \ldots, e_{q_r})$ does not overlap any other subsequences and it is in between two subsequences $I_p = (e_{p_\ell}, \ldots, e_{p_r})$ and $I_r = (e_{r_\ell}, \ldots, e_{r_r})$. That is, the ordering of the timestamps of the edges are $p_r < q_\ell < q_r < r_\ell$. Note that I_p is identified as a valid maximal subsequence because $e_{p_\ell}, \ldots, e_{p_r}$ satisfies some monotone property while $e_{p_\ell}, \ldots, e_{p_{r+1}}$ does not. So our algorithm will keep deleting edges from e_{p_ℓ} and check for satisfiability of the property. In this process all edges from e_{p_ℓ} to $e_{q_{\ell-1}}$ go through the insertion, verification and deletion process. Finally edge e_{q_ℓ} will be verified and if I_q is valid then all edges in the sequence $(e_{q_\ell}, \ldots, e_{q_r})$ must be identified by our algorithm.

Case (b): when I_q partially overlaps I_p and I_r and suppose the ordering of the timestamps of the edges are $q_\ell < p_r < s_\ell < q_r < s_r$. As before our algorithm identifies that $e_{p_\ell}, \ldots, e_{p_r}$ satisfies some monotone property but $e_{p_\ell}, \ldots, e_{p_{r+1}}$ does not. Now if I_q is valid, when edges $e_{p_\ell}, \ldots, e_{q_{\ell-1}}$ are deleted then $e_{q_\ell}, \ldots, e_{p_{r+1}}$ must satisfy the property and do not get deleted. In this step our algorithm must keep adding next edges in the sequence and identifies $e_{q_\ell}, \ldots, e_{q_r}$ as the valid maximal subsequence I_q. □

For any monotone property, this approach generates $O(m)$ maximal subsequences. So for any RE graph G, this preprocessing step produces a point set S, with $|S| \leqslant m$. We build a 2-dimensional priority search tree (PST), see [27], on the point set S. To answer queries of the form '*Given a query time interval* $[i, j]$ *s.t.* $i \leqslant j$, *does the graph slice* $G_{i,j}$ *satisfy property* $\mathcal{P}$*?*', we query this data structure using a grounded query rectangle $q = (-\infty, i] \times [j, +\infty)$. We report that $G_{i,j}$ satisfies $\mathcal{P}$ if the query returns a positive count. Total space required by a 2-dimensional PST is linear. Thus the total space requirement of this approach is $O(S(n) + m)$. A 2-dimensional PST on m points can answer each grounded query in time $O(\log m)$. If G is completely connected then $m = O(n^2)$ and therefore $O(\log m) = O(\log n)$. We will use this assumption through out the paper. We summarize the result in the following theorem.

THEOREM 2.9. *Suppose, $\mathcal{D}$ is a dynamic algorithm that maintains a monotone graph property $\mathcal{P}$ using $S(n)$ space, and requires $U(n)$ time per update and $Q(n)$ time per query. Given an RE graph with n vertices and m edges and an arbitrary query time interval $[i, j]$, we can reduce the time windowed decision problem for reporting whether $G_{i,j}$ satisfies a monotone property $\mathcal{P}$ to a 2-dimensional range searching query in $O(m \cdot (U(n) + Q(n)))$ time using $O(S(n) + m)$ space that can answer queries in $O(\log n)$ time.*

2.3.2 *Bipartiteness*

A graph is bipartite if the set of its vertices can be decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent. We directly use Henzinger and King's [21] dynamic algorithm for maintaining bipartiteness of a graph that supports each update in $O(\log^3 n)$ time and each query can be answered in $O(\log n / \log\log n)$ time. Their data structure uses $S(n) = O(m + n \log n)$ space. Therefore we obtain the following result.

COROLLARY 2.10. *We can preprocess an RE graph G with n vertices and m edges into a data structure of size $O(m + n \log n)$ in $O(m \log^3 n)$ time such that a time windowed bipartiteness decision query can be answered in $O(\log n)$ time.*

2.3.3 *Disconnectedness*

Let $G = (V, E)$ be a graph such that G is not connected. Then observe that for any subgraph $E' \subseteq E$, $G' = (V, E')$ is also disconnected. Thus the property of being disconnected is monotone. Following our algorithm, as described in Section 2.3.1, we first identify all maximal subsequences of edges $e_a, e_{a+1}, \ldots, e_{b-1}, e_b$ such that G is not connected by the edges of $G_{a,b}$. We use the deterministic dynamic connectivity algorithm by Holm et al. [22, Theorem 3] to answer connectivity queries. This dynamic data structure $\mathcal{D}$ uses $S = O(m + n \log n)$ space, amortized $U(n) = O(\log^2 n)$ update time and worst case $Q(n) = O(\log n / \log\log n)$ query time to answer whether two vertices u and v are connected in a given graph G.

However, our time windowed queries ask for the connectivity of the entire graph and not just any two vertices. We describe here how to use the same data structure $\mathcal{D}$ to answer the full connectivity query by executing an additional check. We keep a *count* that stores the number of connected components of G. Initially, *count* is equal to total number of vertices since no edges have been inserted into the data structure so far, and thus each vertex represents one component. Suppose at some time during preprocessing G, we finished processing edge e_{k-1} and now we want to insert an edge $e_k = (u, v)$ into this structure. Also assume that vertices u and v are not connected in $\mathcal{D}$. Since inserting e into $\mathcal{D}$ will connect two previously disconnected components containing vertices u and v, respectively, into one component, the total number of components maintained by $\mathcal{D}$ will be reduced by one. So *count* is decreased by 1. By a similar argument, every time an edge e is deleted from $\mathcal{D}$ the total number of components maintained by this structure will be increased by one. We update *count* accordingly during processing each edge. Thus, we know that a graph slice is connected only when *count* equals 1. Updating *count* takes $O(1)$ time per edge insertion and deletion.

In total, we can find at most $O(m)$ maximal subsequences of edges that satisfy the connectivity property, or equivalently, $O(m)$ points to be stored in the dominance counting structure. So, our windowed data structure for disconnectedness requires $O(m + n \log n)$ space and $O(m(\log^2 n + \log n / \log\log n)) = O(m \log^2 n)$ time for preprocessing. Using Theorem 2.9, we obtain the following result for the windowed query for disconnectedness.

COROLLARY 2.11. *We can preprocess an RE graph G with n vertices and m edges into a data structure of size $O(m + n \log n)$ in $O(m \log^2 n)$ time such that a time windowed decision query for disconnectedness can be answered in $O(\log n)$ time.*

2.4 PROBLEMS ON MINIMUM SPANNING TREES

In this section we solve three window query problems using the minimum spanning tree (MST) of a weighted RE graph. In particular, we first show an application of dynamic algorithm to construct a data structure to report whether an MST exists in the query time interval. Next we report the minimum spanning interval of an MST for any query time interval. Then, we show how to report the graph edit distance to transform a query graph slice into a spanning forest.

2.4.1 *Weight of the MST*

Suppose a weighted RE graph $G = (V, E)$ is given where each edge of G has a positive numerical weight. Let the weight of the MST of G be denoted by ω^*. For a query time interval $[i, j]$, we want to report whether there exists some MST $T = (V, E')$ such that $E' \subseteq \{e_i, \cdots, e_j\}$ with weight ω^*.

We utilize two dynamic data structures to preprocess the edges of G. The first dynamic structure (due to Holm et al. [22]) is used to maintain the minimum spanning forest (MSF) of G using the edges inserted so far. We call this data structure $\mathcal{D}_1$. We require a second dynamic structure to verify whether the MSF maintained by $\mathcal{D}_1$ is connected, i.e., it is also the MST of G. We use the dynamic connectivity algorithm that we have used to solve the time windowed problem for disconnectedness (see Section 2.3.3) as our second data structure, namely $\mathcal{D}_2$.

Note that $\mathcal{D}_1$ maintains the MSF of G. We augment $\mathcal{D}_1$ so that every time an edge e is inserted into $\mathcal{D}_1$, it reports which edge has been added to the MSF and which edge (if any) has been deleted from the MSF. After each update, $\mathcal{D}_1$ also updates the total weight of the current MSF. We have the following cases to consider.

1. The newly inserted edge e does not create any cycle with the existing tree edges and therefore is added to the current MSF. $\mathcal{D}_1$ reports that e has been added to the MSF.

2. The newly inserted edge e creates a cycle with the existing tree edges. If the weight of e is less than that of any other edge of the cycle then e replaces the edge with the highest weight in that cycle. Then $\mathcal{D}_1$ reports that e has been added to the MSF and the edge with the highest weight in the corresponding cycle has been deleted from MSF.

3. Otherwise, the MSF does not change and $\mathcal{D}_1$ does not report anything.

These simple augmentations can be done without making any changes to the preprocessing and space time bounds of the original data structure.

Now we discuss the preprocessing. For $k = 1$ to m, we insert edge e_k into $\mathcal{D}_1$ according to their increasing timestamps and check whether e_k becomes a new tree edge in $\mathcal{D}_1$. We insert e_k into $\mathcal{D}_2$ in two cases; (a) when e_k becomes a new tree edge and (b) when e_k replaces an existing tree edge. Every time a new tree edge e_k is inserted into $\mathcal{D}_2$ we check the connectivity of the current subgraph using $\mathcal{D}_2$. Every time a new tree edge e_k replaces an old tree edge $e_{k'}$ in $\mathcal{D}_1$, we insert e_k into $\mathcal{D}_2$ and delete $e_{k'}$ from $\mathcal{D}_2$. We keep repeating this process until for the first time we find a subsequence of edges $e_1, \ldots, e_q$ that has the MST of G with weight ω^*. Observe that in case where $q > n-1$, some of the edges in this subsequence with higher weights have been replaced by some other lighter edges from the same subsequence. We start deleting from edge e_1 and check whether $e_2, \ldots, e_q$ still holds the MST of weight ω^*. We keep deleting edges according to the same sequence of edge insertion until we find the minimal subsequence $I = e_p, \ldots, e_q$ where the MST exists. We store a point $(p, q) \in \mathbb{R}^2$ marking that there exists an MST of G in the interval $[p, q]$. Now we delete edge e_p, insert edge e_{q+1} and repeat the whole process until we finish scanning all edges of G. At the end of the preprocessing we have a set of points $S' \in \mathbb{R}^2$, where each point in S' represents a minimal subsequence in which an MST with weight ω^* exists. Let $I_{S'}$ be the set of all intervals obtained by this process. Now the following lemma holds.

LEMMA 2.12. *A query time interval* $[i, j]$ *contains some interval* $[p, q] \in I_{S'}$ *if and only if* $SE(i, j) \cap S' \neq \emptyset$, *where* $SE(i, j) = [i, +\infty) \times (-\infty, j]$ *is the South-East quadrant of the point* (p, q).

The proof is similar to the one presented for Lemma 2.8 and hence omitted.

THEOREM 2.13. *Suppose a weighted RE graph* G *is given with* m *edges and* n *vertices, and let* G *has an MST of weight* ω^*. *Given a query time slice* $[i, j]$, *the problem of reporting whether* $G_{i,j}$ *contains an MST of* G *with weight* ω^* *can be reduced to a 2-dimensional range searching query in* $O(m \log^4 n)$ *time using* $O(m \log n)$ *space. Queries can be answered in* $O(\log n)$ *time.*

Proof. We observe that no two intervals in $I_{S'}$ can have same start time. Therefore, there can be at most one interval starting from any time k with $1 \leqslant k \leqslant n$. So the number of MSTs that can possibly be generated over m edge updates is at most $O(m)$. Holm et al.'s algorithm maintains an MSF using amortized $O(\log^4 n)$ time per update and $O(m \log n)$ space [22, Theorem 8]. As mentioned before, the dynamic connectivity algorithm by the same authors provides $O(\log^2 n)$ update time using $O(m + n \log n)$ space. Similar to the data structure of Section 2.3.1 we build a 2-dimensional PST on $O(m)$ points using linear space and query using a grounded query rectangle $[i, +\infty) \times (-\infty, j]$ in $O(\log n)$ time. We report there exists an MST of G in $G_{i,j}$ if the query returns a positive count. □

2.4.2 *Minimum Spanning Interval*

We define a *spanning interval* as the time difference $t(e_q) - t(e_p)$ such that there exists an MST $T = (V, E' \subseteq \{e_p, \cdots, e_q\})$ of G. The motivation for solving this problem is the fact that given a query time interval $[i, j]$ multiple MSTs of G can exist in the graph slice $G_{i,j}$, and we are interested to find a minimum spanning interval that contains an MST. To solve this problem we change the data structure described in Section 2.4.1 as follows. The length of the minimal subsequence

$e_p, \ldots, e_q$ that holds an MST is $q - p + 1$. So now we consider each point $a = (p, q)$ as a weighted point and initialize the weight of each point to $w(a) = q - p + 1$. Therefore the problem of finding the minimum spanning interval within $G_{i,j}$ can be reduced to the orthogonal range minimum problem on $O(m)$ points. According to Chan et al. [7], orthogonal range minimum problem can be solved in $O(\log \log n)$ time using $O(m \log^{\epsilon} n)$ space, where ϵ is a small positive constant. We summarize the result here.

THEOREM 2.14. *Given an RE graph* G *with* m *edges and* n *vertices, and a query time slice* $[i, j]$*, the problem of reporting the minimum spanning interval in* $G_{i,j}$ *can be reduced to the orthogonal range minimum problem in* $O(m \log^4 n)$ *time using* $O(m \log^{\epsilon} n)$ *space. Queries can be answered in* $O(\log \log n)$ *time.*

2.4.3 *Graph Edit Distance for Target Class Forest*

DEFINITION 2.15. *Given a set of graph edit operations (insertion or deletion of graph edges), the graph edit distance* $\mathcal{GED}(G, H)$ *between a source graph* G *and a target graph* H *is defined as* $\mathcal{GED}(G, H) = \min\{c(S) | S$ *is a sequence of operations transforming* G *into* $H\}$.

In this definition, $S = (s_1, s_2, \ldots, s_k)$ is a sequence of operations that transforms G into H. The cost of a sequence $S = (s_1, s_2, \ldots, s_k)$ is given by $c(S) = \sum_{i=1}^{k} c(s_i)$, where $c(s_i)$ is the cost of the operation s_i. The goal of the graph edit distance is to find the minimum cost of the operations that makes the transformation possible. We consider *edge deletion* as the only permitted graph edit operation to solve our problem. We assume that for unweighted graphs, each edit operation has a unit cost. In this section, we want to solve the following problem.

Given an RE graph $G = (V, E)$ and a query time interval $[i, j]$, we want to compute the $\mathcal{GED}(G_{i,j}, H)$ where $H = (V, E')$ is a spanning forest of $G_{i,j}$ and $E' \subseteq \{e_i, e_{i+1}, \ldots, e_j\}$. This is equivalent to saying that we want to find a set of edges X such that $G_{i,j} \setminus X$ is a forest and $|X|$ is minimum.

First we present the following lemma.

LEMMA 2.16. *The total cost* $\mathcal{C}$ *of* $\mathcal{GED}(G_{i,j}, H)$ *is equivalent to the number of chordless cycles in* $G_{i,j}$*, where* H *is a spanning forest of* $G_{i,j}$.

Proof. Let C be a simple cycle without any chords. Observe that by deleting any edge of C, we obtain a spanning forest (tree) of C. Also note that graph edit distance must report the minimum cost of operations. Since one edge deletion is required for every chordless cycle and every edge deletion operation has unit cost, the total cost $\mathcal{C}$ of $\mathcal{GED}(G_{i,j}, H)$ is the same as the total number of edge deletions in $G_{i,j}$. Therefore, reporting $\mathcal{GED}(G_{i,j}, H)$ is equal to the number of chordless cycles in $G_{i,j}$. Therefore the lemma holds. □

Algorithm: We maintain a link-cut tree T, see [31], to store the vertices of G. A link-cut tree allows edge insertions and deletions in amortized time $O(\log n)$. This data structure also supports standard aggregate functions (e.g., max, min, sum or increment) over all the edges from a vertex v to the root of T in time $O(\log n)$ [31]. We store the timestamp of an edge as its weight so that

we can query T to find the edge with the minimum timestamp that is on the path from v to the root.

For $k = 1$ to m, we process each edge e_k in increasing order of the timestamp and add e_k to T unless it creates a cycle with the existing edges of T. Suppose $e_k = (u_k, v_k)$ is an edge with u_k and v_k as its two end points, that creates a cycle in T. Then there must be an existing path from u_k to v_k in T. We query T to find the edge e_ℓ with the minimum timestamp $t(e_\ell)$ on this path. We store a point $(k, \ell) \in \mathbb{R}^2$ with label ℓ. We delete e_ℓ from T and insert e_k to T. We repeat these steps until we finish processing all edges in E. At the end of this process we obtain a set of labelled points P with $|P| = O(m)$ in $\mathbb{R}^2$. Figure 2.7 illustrates an example of our algorithm for computing $\mathcal{GED}$ for forests.

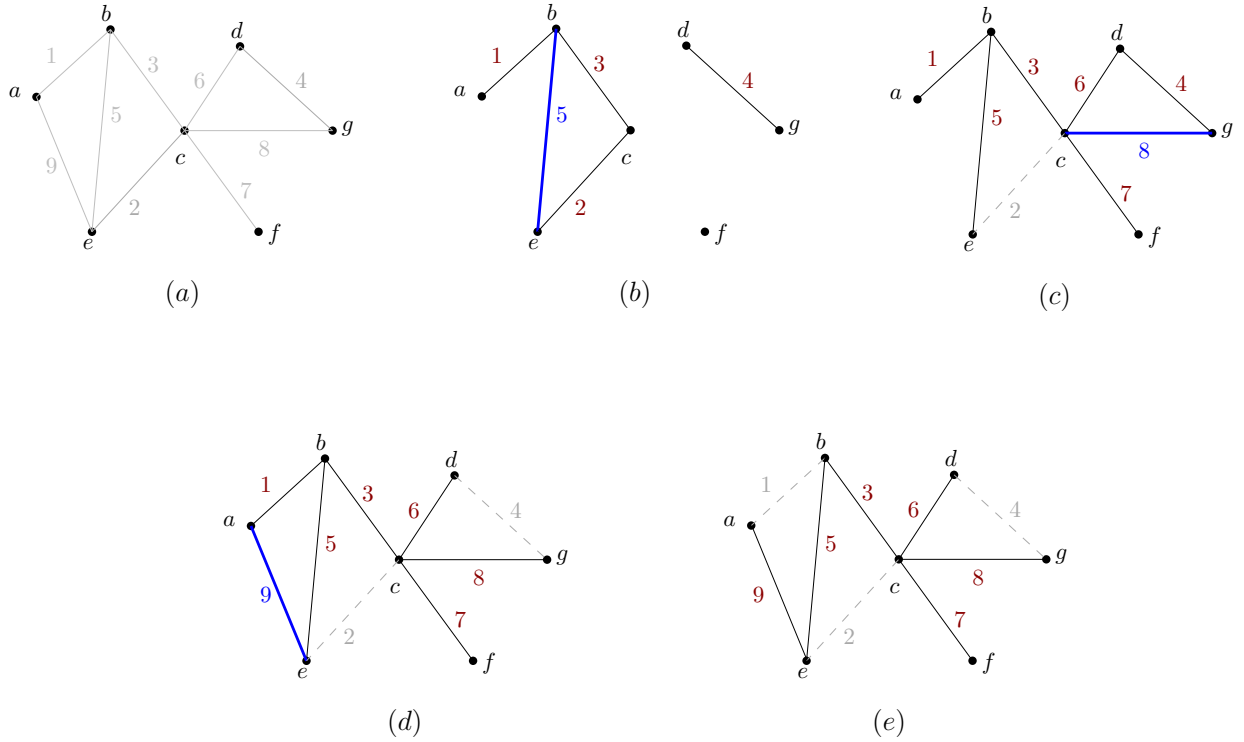


Figure 2.7: (a) An RE graph G with seven vertices stored in a link-cut tree T. Edges are not inserted yet. (b) Starting with e_1 edges are inserted to T according to the increasing timestamps until e_5 creates a cycle $C_1 = \langle e_2, e_3, e_5 \rangle$ (timestamp of each edge is mentioned as an integer value). (c) The edge with the lowest timestamp in C_1 is e_2 and a point $(5, 2) \in \mathbb{R}^2$ with label 2 is stored. Edge e_2 is deleted and $e_6, \ldots, e_8$ are inserted to T. (d) Edge e_8 creates another cycle $C_2 = \langle e_4, e_6, e_8 \rangle$. Similarly the edge with the lowest timestamp e_4 is deleted from T and a point $(8, 4) \in \mathbb{R}^2$ with label 4 is stored. Edge e_9 is inserted to T. (e) e_9 creates cycle $C_3 = \langle e_1, e_5, e_9 \rangle$. Again the edge with the lowest timestamp e_1 in C_3 is deleted from T and a point $(9, 1) \in \mathbb{R}^2$ with label 1 is stored. (e) $\mathcal{GED}(G_{1,9}, H) = 3$, where H is a spanning forest of G. Reported edges are 1, 2 and 4. Bold blue edges create cycles, solid black edges are part of H and dashed grey edges are deleted from $G_{1,9}$.

LEMMA 2.17. *The number of points in* $P \cap ([i, j] \times [i, j])$ *is equal to the* $\mathcal{GED}$ *of* $G_{i,j}$.

Proof. Note that every point $(k, \ell) \in \mathbb{R}^2$ that represents a chordless cycle in the interval $[\ell, k]$ lies below the main diagonal. The number of points $|X| = P \cap ([i, j] \times [i, j])$ gives us the total number of chordless cycles that exist in time interval $[i, j]$. By Lemma 2.16, the number of points in $|X|$ is equal to the $\mathcal{GED}$ of $G_{i,j}$. □

THEOREM 2.18. *Given an RE graph G with m edges and n vertices, and a query time slice $[i, j]$, the problem of reporting the $\mathcal{GED}$ for target class forest can be reduced to the range searching problem in $O(m \log n)$ time using $O(m + n)$ space. Queries can be answered in $O(\log n + |X|)$ time using $O(m \log n)$ space, where X is the set of edges such that $G_{i,j} \setminus X$ is a forest and $|X|$ is minimum.*

Proof. The preprocessing step ensures that when an edge e_k creates a cycle such that e_ℓ is an edge in that cycle with the smallest timestamp, the interval $[\ell, k]$ contains exactly one chordless cycle. According to Lemma 2.16 this chordless cycle in $G_{\ell,k}$ must contribute a unit cost to the graph edit distance. So a point $(k, \ell) \in \mathbb{R}^2$ with label ℓ represents that a cycle exists in time interval $[\ell, k]$ and e_ℓ is the edge that must be deleted to maintain the forest. It is also necessary to delete e_ℓ physically from T to ensure that the next cycle found by the scanning process is chordless. Labels are required only if we want to report the edges. Otherwise if we want to report the value of $\mathcal{GED}$ only (i.e., the total number of edges to be deleted) then we can ignore labels.

Every edge will be inserted and deleted from T at most once in this process. Queries are also made at most once for each edge. Therefore, total preprocessing time is $O(m \log n)$ using $O(n + m)$ space [31]. As mentioned above P has $O(m)$ labelled points in $\mathbb{R}^2$. P can be stored using a standard range search tree that can be built in $O(m \log n)$ time using $O(m \log n)$ space [15].

For a given query time interval $[i, j]$, we count the number of points $|X| = P \cap q$ that intersect with the query rectangle $q = [i, j] \times [i, j]$ in $O(\log n)$ time. By Lemma 2.17 we report $|X|$ as the $\mathcal{GED}$ for converting $G_{i,j}$ to a forest, i.e., $|X|$ is the number of edges needed to be deleted from $G_{i,j}$ such that the resulting graph slice becomes a forest. If the query is to report the set of edges that are needed to be deleted, we can answer in $O(\log n + |X|)$ time to report the labels of the points (i.e., timestamps of the deleted edges), where X is the set of deleted edges. □

2.5 PROBLEMS ON COUNTING SUBGRAPHS

In this section we consider the problems of counting subgraphs of some fixed patterns in a queried graph slice $G_{i,j}$. The patterns for which we present data structures are 2-paths (paths of length two in general graphs), 3-paths (paths of length three in bipartite graphs), all complete subgraphs of size 3 and more, and quadrangles (C_4) or simple cycles of length 4.

2.5.1 *Counting 2-paths and 3-paths (in bipartite graphs)*

First, we consider the case of counting the number of 2-paths. Let $G = (V, E)$ be an RE graph with n vertices and m edges. We maintain a set of n lists, one for each vertex $v_k \in V$, where $1 \leqslant k \leqslant n$. The list L_{v_k} stores timestamps of v_k's incident edges, sorted in increasing order. The length of this list is $deg(v_k)$, where $deg(v_k)$ is the degree of v_k in G. Therefore, the total space over all the lists is at most $\sum_{v_k \in V} deg(v_k) = 2m = O(m)$.

Next we analyze the complexity of computing all 2-paths in G. For a query time slice $[i, j]$, suppose, i' and j' are the indices of the elements in L_{v_k} such that $L_{v_k}[i']$ is the smallest element $\geqslant i$, and $L_{v_k}[j']$ is the largest element $\leqslant j$. Then the total number of 2-paths centering v_k in $G_{i,j}$ will be $\binom{j'-i'+1}{2}$. For each vertex v_k, the number of 2-paths where v_k is the center vertex can be computed by performing a binary search in the list L_{v_k} to locate i' and j' as discussed above. This requires $O(\log deg(v_k))$ time. Thus we can report all 2-paths in $G_{i,j}$ in $O(\sum_{k=1}^{n} \log deg(v_k))$ $= O(n \log n)$ time. Using *fractional cascading* data structure, see [13], we can improve the query time to $O(n)$ time without incurring any increase in space as follows. Fractional cascading data structure is generally used to solve *iterated search* problem, where many search problems can be solved by first solving a subproblem whose size is a constant fraction of the original problem size and then using this solution to obtain the solution of the original problem. We provide a brief description of how this data structure can be built and used in our case. We define $Y_{v_n} = L_{v_n}$. Suppose we take a sample of size $\lfloor n/2 \rfloor$ from Y_{v_n} by selecting every alternate elements. Then elements from this sample list are merged with $L_{v_{n-1}}$ to obtain $Y_{v_{n-1}}$. For each element k, with $1 \leqslant k \leqslant n$, in $Y_{v_{n-1}}$ we use two pointers that points respectively to the smallest integer p such that $Y_{v_n}[p] \geqslant k$ and to the first element of $Y_{v_{n-1}}$ that comes from $L_{v_{n-1}}$ and appears after k. We repeat this process of using $L_{v_{n-2}}$ and $Y_{v_{n-1}}$ to obtain $Y_{v_{n-2}}$ until we obtain Y_{v_1}. Total space required is $O(n)$. Now the iterative query can be answered by first using a binary search in some L_{v_k} with keys i' and j', where $1 \leqslant k \leqslant n$. For the subsequent steps we can find each pair of $L_{v_{k+1}}[i'']$ and $L_{v_{k+1}}[j'']$ from the information of $L_{v_k}[i']$ and $L_{v_k}[j']$ in constant time. Therefore queries can be answered in $O(n + \log n) = O(n)$ time.

Now we present data structure for counting all 3-paths in a bipartite RE graph $G = (V = \{A \cup B\}, E = \{(u, v) : u \in A, v \in B\})$. We maintain two cascading structures. The first structure $\mathcal{D}_1$ maintains all 2-paths centering each vertex $u_k \in A$ with $1 \leqslant k \leqslant |A|$. $\mathcal{D}_2$ is a similar structure for each vertex $v_\ell \in B$ with $1 \leqslant \ell \leqslant |B|$. We also maintain an array, namely $count[1..|A|]$, that initially stores zero in each $count[k]$ with $1 \leqslant k \leqslant |A|$. We update this array during the query with the following information. For query interval $[i, j]$, each $count[k]$ will store $deg_{i,j}(u_k)$, i.e., the number of edges adjacent to u_k in $G_{i,j}$, for all $u_k \in A$, see Figure 2.8.

The data structures $\mathcal{D}_1$ and $\mathcal{D}_2$ are adjacency linked lists. In $\mathcal{D}_1$, every node on u_k's list contains two fields, a vertex v and the timestamp $t(u_k, v)$ such that $(u_k, v) \in E$, $u_k \in A$ and $v \in B$ (see Figure 2.8(b)). In $\mathcal{D}_2$'s structure, every node on v_ℓ's list also has two similar fields and an extra pointer field that points to $count[k]$, if the edge (v_ℓ, u_k) appears in $G_{i,j}$ (see Figure 2.8(c)). Total preprocessing takes $O(m + n)$ time.

The query is processed in two steps. Given a query time interval $[i, j]$, we first perform a binary search on $\mathcal{D}_1$ using key values i and j. This process is same as counting the number of 2-paths. For every $u_k \in A$, with $1 \leqslant k \leqslant |A|$, we store the number of edges adjacent to u_k within time interval $[i, j]$ in position $count[k]$. This step takes $O(n)$ time using fractional cascading.

Next, we do a similar binary search on $\mathcal{D}_2$, and similar to the process described above for 2-paths, let i' and j' be the indices of the elements in L_{v_ℓ} such that $L_{v_\ell}[i']$ is the smallest element $\geqslant L_{v_\ell}[i]$, and $L_{v_\ell}[j']$ is the largest element $\leqslant L_{v_\ell}[j]$. This time, we walk along each element from $L_{v_\ell}[i']$ to $L_{v_\ell}[j']$ for every $v_\ell \in B$, and follow the pointer of each $u_k \in N(v_\ell)$ to reach $count[k]$.

The number of 3-paths in $G_{i,j}$ of the form $\langle a, v_\ell, u_k, b \rangle$, such that $v_\ell \in B$ is a fixed vertex and $u_k \in A$ is a fixed neighbour of v_ℓ, are equal to $(deg_{i,j}(v_\ell) - 1) \times (count[k] - 1)$, given that

$deg_{i,j}(v_\ell) > 1$ and $count[k] > 1$. Otherwise no 3-path exists of this form. So, the total number of 3-paths passing through all the neighbours $u_k \in N(v_\ell)$ of a fixed v_ℓ in $G_{i,j}$ will be

$$(deg_{i,j}(v_\ell) - 1) \times \sum_{\forall k : u_k \in N_{i,j}(v_\ell)} (count[k] - 1),$$

where $N_{i,j}(v_\ell)$ denotes neighbors of v_ℓ in $[i, j]$. Therefore, the count of all 3-paths in $G_{i,j}$ will be

$$\sum_{\ell=1}^{|B|} \Big((deg_{i,j}(v_\ell) - 1) \times \sum_{\forall k : u_k \in N_{i,j}(v_\ell)} (count[k] - 1) \Big).$$

We can find the number of adjacent edges of all $u_k \in A$ with $i \leqslant k \leqslant j$, and of all $v_\ell \in B$ with $i \leqslant \ell \leqslant j$, in $O(n)$ time using fractional cascading on $\mathcal{D}_1$ and $\mathcal{D}_2$. Since scanning neighbours of all v_ℓ, with $1 \leqslant \ell \leqslant |B|$, requires at most $O(m)$ time, total query time to find all 3-paths in $G_{i,j}$ is $O(n + m)$. We have used an array of size at most n, and two copies of the similar data structure that has been used for counting the number of 2-paths. Thus the total space requirement is $O(m + n)$.

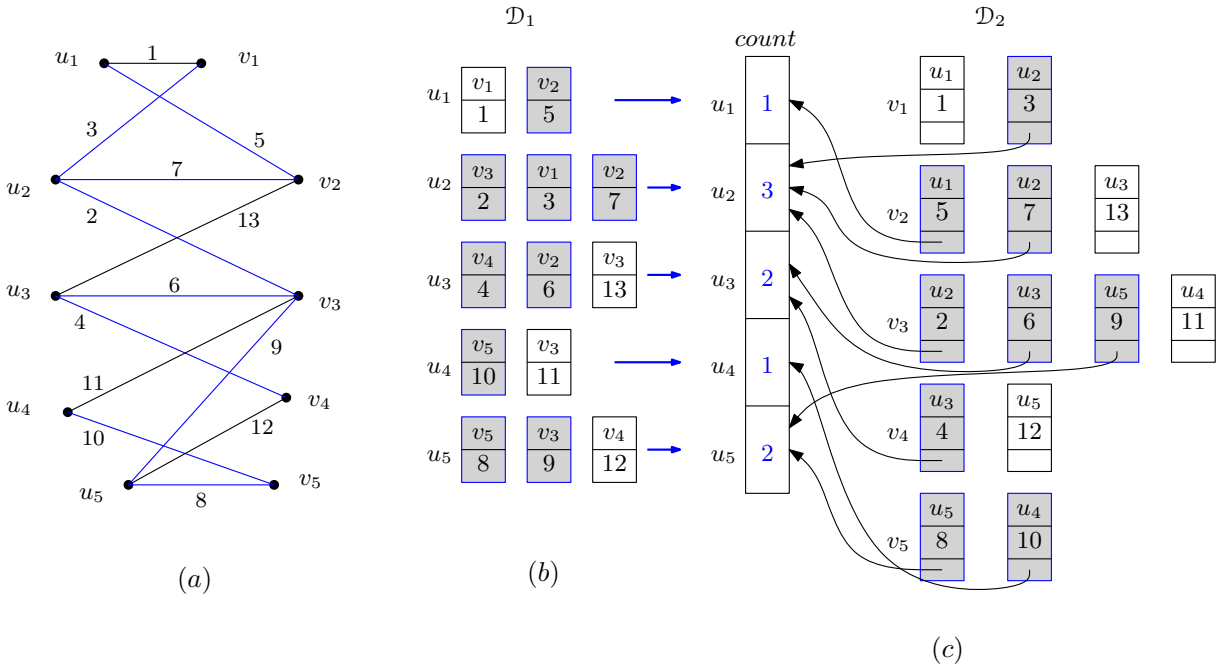


Figure 2.8: (a) A bipartite RE graph G, (b) Fractional cascading structure $\mathcal{D}_1$, (c) Fractional cascading structure $\mathcal{D}_2$. An example illustrating how $\mathcal{D}_1$ and $\mathcal{D}_2$ are queried with time interval [2,10].

THEOREM 2.19. *(a) Let $G = (V, E)$ be an RE graph with n vertices and m edges. G can be preprocessed into a data structure in $O(n + m)$ time using $O(n + m)$ space such that the total number of 2-paths in $G_{i,j}$ can be counted in $O(n)$ time. (b) Let $G = (V, E)$ be a bipartite RE graph with n vertices and m edges. G can be preprocessed into a data structure in $O(n + m)$ time using $O(n + m)$ space such that the total number of 3-paths in $G_{i,j}$ can be counted in $O(n + m)$ time.*

2.5.2 *Counting Complete Subgraphs of a Fixed Order $\ell \geqslant 3$*

In this section we present the general overview of the technique to count all complete subgraphs of order $\ell \geqslant 3$ in an RE graph $G = (V, E)$. The *order* of G is the number of vertices in G. A complete subgraph K_ℓ of G is a subgraph induced by a subset of the vertices V such that every two distinct vertices are adjacent in K_ℓ, where ℓ is the order of K_ℓ.

Preprocessing step: We modify the edge-scanning algorithm of Chiba and Nishizeki [14] that was initially presented to report all complete subgraphs of a fixed order $\ell \geqslant 3$ in a graph as follows. Vertices of G are sorted in non-increasing order of their degrees and without loss of generality, let $deg(v_1) \geqslant deg(v_2) \geqslant \cdots \geqslant deg(v_n)$, where $deg(v)$ is the degree of a vertex v in graph G. Starting with vertex v_1 the algorithm marks the subgraph induced by the neighbours $N(v_1)$ of v_1, and finds all instances of a specified complete subgraph that contains v_1. For each of these instances, we compute the *high* and the *low* values by comparing the timestamps of participating edges. Recall that $high$ (or low) is the highest (or the lowest) timestamp of all the edges involved in a subgraph. The interval $[low, high]$ represents the timespan of that subgraph in G. Each subgraph is then represented by a point $(high, low)$ in $\mathbb{R}^2$. Once all subgraphs containing v_1 are stored as a set of points, v_1 is deleted from G to avoid duplication and the process continues with the next vertex in the sequence. At the end of the process a set of point $P \in \mathbb{R}^2$ representing all complete subgraphs of order ℓ is obtained.

In [14], the algorithm finds a complete subgraph K_ℓ containing a vertex v by detecting $K_{\ell-1}$ in the subgraph induced by the neighbours of v. We modify their algorithm (see modified Algorithms 1 and 2) as follows. An initially empty global stack S is used that now stores $(\ell - k)$ elements of the form $(v_i, min_{v_i}, max_{v_i})$ at the recursion level $(\ell - k)$. All vertices v_i stored in S are pairwise adjacent in G.

Algorithm 2 (procedure $CS(k, G_k)$) finds all complete subgraph of order k in G_k. Each of these complete subgraphs forms a K_ℓ together with $(\ell - k)$ vertices stored in S. When procedure $CS(k, G_k)$ is executed, at a recursive call of depth $(\ell - k)$, we compute low and $high$ events (denoted with min_{v_i} and max_{v_i}, respectively) that are respectively the minimum and the maximum timestamps within the complete subgraph consisting vertices $(v_1 \ldots v_i)$. When k becomes 2, for each edge (x, y) left in G_2, we list vertices $\{x, y\} \cup S$ that form K_ℓ. At this stage, we use the timestamps of edges formed by connecting x and y with each vertex $z \in S$ to update the *low* and the *high* events of K_ℓ. A point $(high, low)$ is stored in $\mathbb{R}^2$ that represents a K_ℓ in the time interval $[low, high]$. The graph representation requires linear space using adjacency lists data structure. The linear space implementation of counting complete subgraphs of fixed order using the same data structure is available in [14]. Thus total space requirement becomes $O(m + n + \mathcal{K})$, where $\mathcal{K}$ is the number of complete subgraphs of the fixed order ℓ. We obtain the following results.

THEOREM 2.20. *Given an RE graph $G = (V, E)$ with m edges and $\mathcal{K}$ complete subgraphs of a fixed order ℓ ($\geqslant 3$), the problem of determining the number of complete subgraphs of order ℓ in the query time interval $[i, j]$ can be reduced to dominance counting in $O(a(G)^{\ell-2}m)$ time using $O(m + n + \mathcal{K})$ space. The query takes $O(\log \mathcal{W} / \log \log \mathcal{K})$ time to count the total number of complete subgraphs of order ℓ in $G_{i,j}$, where $\mathcal{W}$ is the width of the query window.*

Proof. We follow the proof technique used in [14]. Let $n = |V|$ and $m = |E|$. The function PreprocessCS(ℓ, G) recursively calls procedure CS(k, G_k) with $k = \ell$ and $G_k = G$. Let $T(k, m, n)$ be the time required by procedure CS(k, G_k) to find all K_k's in G_k.

When $k = 2$: for each edge in G_2 we update *low* and *high* values of K_k by comparing timestamps of edges connecting at most ℓ vertices. This time is upper bounded by $O(m + n)$. So, $T(2, m, n) = O(m + n)$.

Next, for $k \geqslant 3$: in the i'th iteration of the **for** loop (lines 4 to 16), we find the subgraph G_{k-1} induced by the neighbours of the vertex with the current highest degree v_i in $O\big(d_k(v_i) + \sum_{u \in N(v_i)} d_k(u)\big)$ time (line 5). Line 13 is a recursive call that requires $T\big(k-1, (\sum_{u \in N(v_i)} d_k(u))/2, d_k(v_i)\big)$ time to find and store all complete subgraphs containing v_i. All updates of min_{v_i} and max_{v_i} values with respect to the timestamps of edges connecting v_i with $(\ell - k)$ vertices stored in stack S, is again upper bounded by the time required to compute G_{k-1} in line 5. All stack operations (lines 12 and 14) require constant time.

Thus, the total time required for each v_i is,

$$O\Big(d_k(v_i) + \sum_{u \in N(v_i)} d_k(u)\Big) + O(1) + T\Big(k-1, \big(\sum_{u \in N(v_i)} d_k(u)\big)/2, d_k(v_i)\Big)$$

which is the same as was obtained in [14]. Then following the analysis of [14, Theorem 3] we obtain $T(k, m, n) = O(a(G_k)^{k-2}m + n)$ with $k \geqslant 3$. Note that we do not explicitly report the complete subgraphs. Therefore, when $k = \ell$ finding all K_ℓ's in G requires at most $O(a(G)^{\ell-2}m)$ time.

Points in $P = p_1, \ldots, p_{\mathcal{K}}$ can be stored in a window sensitive dominance data structure $\mathcal{D}$ of linear size, see [3]. Given a query interval $q = [i, j]$ we query $\mathcal{D}$ using $(-\infty, j] \times [i, +\infty)$. Since $high > low$ for every point $(high, low) \in P$, all points of P lie below the main diagonal, (see Figure 2.3(b)). This condition is required by the window sensitive dominance structure $\mathcal{D}$. Thus $\mathcal{D}$ can report the number of points dominated by our query in $O(\log \mathcal{W} / \log\log \mathcal{K})$ time, where $\mathcal{W} = j - i + 1$ is the width of the query interval and $\mathcal{K}$ is the total number of complete subgraphs of order ℓ. □

Algorithm 1: PreprocessCS(ℓ, G)

Input : Relational event graph G and the order of complete subgraph ℓ.
Output: A set of points P in $\mathbb{R}^2$.

1 Set stack S $\leftarrow \emptyset$.
2 Set mt $\leftarrow$ maximum timestamp in G.
3 Set min $\leftarrow$ mt + 1, and max $\leftarrow -1$.
4 Let $G_\ell = G$.
5 Call CS(ℓ, G_ℓ).

Algorithm 2: CS(k, G_k)

Input : Graph slice G_k and parameter k.
Output: A set of points P in $\mathbb{R}^2$.

if $k > 2$ **then**
  Set $j \leftarrow$ order of G_k.
  Sort vertices $v_1, v_2, \ldots, v_j$ in non-increasing order of degrees. Without loss of generality, let $d(v_1) \geqslant d(v_2) \geqslant \cdots \geqslant d(v_j)$.
  for $i = 1$ *to* j **do**
    Let $G_{k-1} \subseteq G_k$ be the subgraph induced by the neighbours of v_i.
    if *stack* $S \neq$ NIL **then**
      Update $\min_{v_i}$ and $\max_{v_i}$ with respect to $t(v_i, z)$ where $z \in S$.
      else
        Set $(\min_{v_i} \leftarrow \min)$ and $(\max_{v_i} \leftarrow \max)$.
      end
    end
    Add $(v_i, \min_{v_i}, \max_{v_i})$ to stack S.
    Call CS($k-1$, G_{k-1}).
    Delete $(v_i, \min_{v_i}, \max_{v_i})$ from stack S.
    Delete v_i from G_k and without loss of generality, let G_k be the resulting graph.
  end
end
else
  if $k = 2$ **then**
    for *each edge* (x, y) *of* G_2 **do**
      Set $(low \leftarrow \min_{v_i})$ and $(high \leftarrow \max_{v_i})$.
      Update low and $high$ with $t(x, y)$.
      Update low and $high$ with respect to $t(x, z)$ and $t(y, z)$ where $z \in S$.
    end
  end
end

2.5.3 *Counting All Complete Subgraphs of Orders $\ell \geqslant 3$*

We extend our algorithm for computing all complete subgraphs of orders between 3 to ℓ. That is, if we are given an order ℓ, we can find all complete subgraphs K_k in $G_{i,j}$, where $3 \leqslant k \leqslant \ell$. For each order $3 \leqslant k \leqslant \ell$, the previous algorithm of a fixed order k can be run once in total $O\big(m \sum_{k=3}^{\ell} a(G)^{k-2}\big) = O(a(G)^{\ell-1} m)$ time to obtain corresponding timespans $[low, high]$ of all complete subgraphs. To represent complete subgraphs of all orders, we modify our algorithm so that, for each complete subgraph of order k, we store a point $p = (a, b, k)$ in $\mathbb{R}^3$ where $a = high$, $b = low$ and $k = order$ in any standard dominance counting data structure, for example see [25]. This results in the following theorem.

THEOREM 2.21. *Given an RE graph* G *with* m *edges and an integer* $\ell \geqslant 3$ *, the problem of determining the total number of complete subgraphs* K_k *of all orders* $3 \leqslant k \leqslant \ell$ *in the query interval* $[i, j]$ *can be reduced to dominance counting in* $O(\alpha(G)^{\ell-1}m)$ *time using* $O(m + n + \mathcal{K}\log\mathcal{K}/\log\log\mathcal{K})$ *space, where* $\mathcal{K}$ *is the total number of complete subgraphs in* G*. The query takes* $O((\log\mathcal{K}/\log\log\mathcal{K})^2)$ *time to report the total number of complete subgraphs in* $G_{i,j}$*.*

Remarks. We can also reduce the problem of finding all 2-paths and 3-paths in $G_{i,j}$ to dominance counting as follows. For each vertex $u \in V$, we find its neighbours $v \in N(u)$, and successively find all neighbours of v, denoted as $w \in N(v)$. For each 2-path starting at u, that is $\langle uv, vw\rangle$, we store a point $(high, low)$ in $\mathbb{R}^2$, where $high$ and low are respectively the maximum and minimum timestamps between $t(u, v)$ and $t(v, w)$. During this search, we will also find a path $\langle wv, vu\rangle$ starting from vertex w while exploring neighbours of w, which is the same path as $\langle uv, vw\rangle$. Since each 2-path will be identified exactly twice during the preprocessing, we keep only one point per 2-path to avoid duplication. The time taken to find all 2-paths in G will be upper bounded by $\alpha(G)m$, which is the time needed to identify all triangles in G (see Theorem 2.20). Now, we can count the number of 2-paths in $G_{i,j}$ using our dominance counting structure.

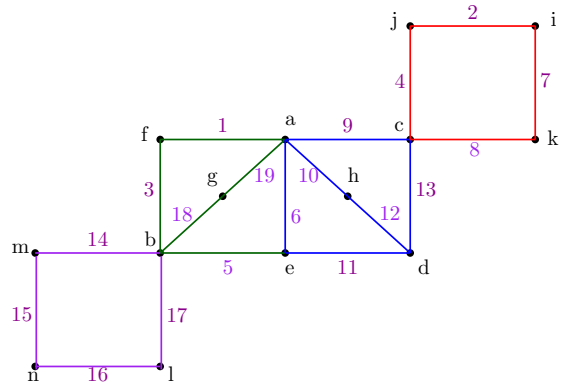
Let $G = (V = \{A \cup B\}, E = \{(u, v) : u \in A, v \in B\})$ be a bipartite RE graph. A complete bipartite graph can have at most $O(n^4)$ 3-paths. Each vertex $u \in A$ in a 3-path shares edges with two neighbours v_1 and v_2 such that $v_1, v_2 \in B$, and vice versa. Thus, reducing the problem of counting all 3-paths in G to the problem of dominance counting requires an exhaustive search for all alternating edge adjacency between the vertices of A and the vertices of B. To find a 3-path $\langle u_1, v_1, u_2, v_2\rangle$, we start from each vertex $u_1 \in A$, and find its neighbour $v_1 \in B$. Successively, we find v_1's neighbour $u_2 \in A$, and u_2's neighbour $v_2 \in B$. For every new edge added to the path, we update $(high, low)$ so that each 3-path can be stored as a point in $\mathbb{R}^2$. Thus, preprocessing all 3-paths in G requires $O(n^4)$ time in the worst case.

COROLLARY 2.22. *Given an RE graph* G *with* m *edges, the problems of counting 2-paths can be reduced to window sensitive dominance counting in* $O(\alpha(G)m)$ *time. Given a bipartite RE graph* $G = (V = A \cup B, E)$ *with* n *vertices, the problems of counting 3-paths can be reduced to window sensitive dominance counting in* $O(n^4)$ *time. Each query can be answered in* $O(\log\mathcal{W}/\log\log\mathcal{K})$ *time, where* $\mathcal{W}$ *is the width of the query window and* $\mathcal{K}$ *is the number of paths in and* G*.*

2.5.4 *Counting Quadrangles*

In this section we present an algorithm for counting quadrangles, i.e., cycles of length 4, in an RE graph G. An edge searching algorithm is presented in [14] where a set of quadrangles can be implicitly represented by a tuple $(y, z, \{a_1, a_2, a_3, ...\})$ in $O(\alpha(G)m)$ time and space, where y and z are vertices on two opposite sides of all quadrangles of this set and each vertex $v \in \{a_1, a_2, a_3, ..\}$ shares edges with both y and z. Within this setting, any two vertices from $\{a_1, a_2, a_3, ..\}$ together with y and z represent a quadrangle.

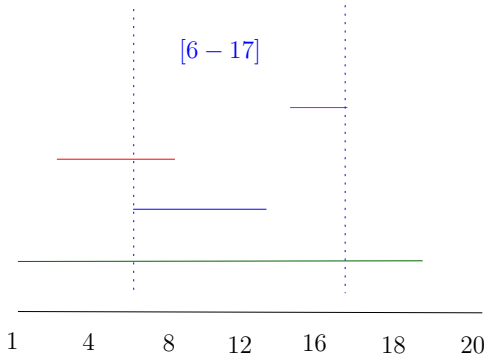
Figure 2.9(c) illustrates a relational event graph with eight quadrangles. The search algorithm represents these quadrangles using four tuples, $(a, b, \{e, f, g\})$, $(a, d, \{c, e, h\})$, $(b, n, \{m, \ell\})$ and $(e, i, \{j, k\})$. We can see that the first tuple $(a, b, \{e, f, g\})$ contains three quadrangles (a, e, b, f), (a, f, b, g) and (a, e, b, g).



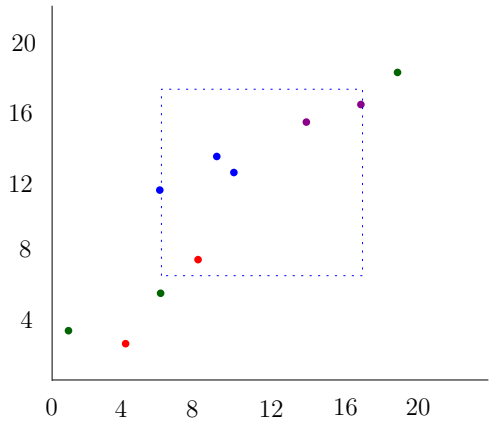
(a)

Tuple	$(a, b, \{e, f, g\})$	$(a, d, \{c, e, h\})$	$(b, n, \{m, l\})$	$(e, i, \{j, k\})$
Interval (x_i)	$[1-19]$	$[6-13]$	$[14-17]$	$[2-8]$
Pointset (P)	$(1,3)$	$(9,13)$	$(14,15)$	$(4,2)$
	$(6,5)$	$(8,12)$	$(17,16)$	$(8,7)$
	$(19,18)$	$(6,11)$		

(b)



(c)



(d)

Figure 2.9: Counting quadrangles in an RE graph. The integer numbers on the graphs edges are their timestamps. (a) Quadrangles in G are highlighted. (b) Tuple representation for each of the quadrangles, and their corresponding time intervals and point sets in $\mathbb{R}^2$. (c) Querying the interval tree for the valid intervals in $q = [6-17]$. (d) Querying the range tree for the valid quadrangles.

We can use the dominance counting data structure to count the total number of quadrangles using a similar algorithm presented in Section 2.5.2 by explicitly representing all quadrangles and computing the corresponding $high$ and low values for each quadrangle. This requires $O(n^2)$ time and space as there can potentially be $O(n)$ vertices in a tuple $(y, z, \{a_1, a_2, a_3, ..., a_{n-2}\})$ such that each vertex in $\{a_1, a_2, a_3, .., a_{n-2}\}$ shares edges with both y and z. To improve the overall space requirement we now present an output sensitive range searching data structure for counting quadrangles in G. We still use the tuple representation of quadrangles discussed above and modify the algorithm as follows (see Algorithm 3).

We need the following additional data structures.

- For each tuple $(y, z, \{a_1, a_2, a_3, ...\})$, we use two linked lists E and E' to store edges adjacent to y and z respectively; that is, $E = \{(y, a_1), (y, a_2), (y, a_3), ..\}$ and $E' = \{(z, a_1), (z, a_2), (z, a_3), ..\}$. Each node in E points to a node in E' that contains the edge having one common endpoint, i.e., (y, a_1) points to (z, a_1), (y, a_2) points to (z, a_2), and so on.
- For each tuple, we use a 2-dimensional range tree to store a set of points $\{(t(e_1), t(e'_1)), (t(e_2), t(e'_2)), ..., (t(e_p), t(e'_p))\}$, where $t(e_i)$ is the timestamp of the edge e_i, and $e_i \in E$, $e'_i \in E'$.
- We store the timespan of each tuple using an interval $x = [min(t(e_1), t(e'_1)), max(t(e_p), t(e'_p))]$. For the given graph G, we use an interval tree to store the set of horizontal intervals $I = \{x_1, x_2, \cdots\}$, where each interval represents the timespan of a tuple.

The quadrangle counting algorithm consists of a *preprocessing* step (Algorithm 3) and a *query* step (Algorithm 4). We describe these steps below.

***Preprocessing* Step:** The preprocessing step takes an RE graph G consisting of n vertices and m edges as input. G is processed by starting with the vertex having the highest degree. So, the vertices are first sorted in non-increasing order of their degrees and without loss of generality, let $d(v_1) \geqslant d(v_2) \geqslant \cdots \geqslant d(v_n)$. Following our search process, once all quadrangles containing v_i are identified correctly, where $1 \leqslant i \leqslant n$, v_i is deleted from G to avoid duplication and the loop continues with the next vertex in the sequence. We first describe two major components of this procedure.

a) Finding Quadrangles: For each vertex $y \in V$, we apply the following technique to find all quadrangles containing y. For each vertex $z \in V$ at distance 2 from y, we find all the vertices that are adjacent to both y and z. We store these vertices in a set $U[z]$. Thus, the tuple $(y, z, U[z])$ represents the set of quadrangles, where every quadrangle has vertices y and z as two opposite corner points.

b) Finding Intervals: Recall that we want to count the number of quadrangles within a query time slice $[i, j]$, but all we have is the representation of a set of quadrangles as tuples. So, for each such tuple we mark its timespan and maintain some geometric data structures so that we can answer the query.

First, we store each tuple $(y, z, U[z] = \{a_1, a_2, a_3, \ldots\})$ using two linked lists denoted by $E = \{(y, a_1), (y, a_2), (y, a_3), ..\}$ and $E' = \{(z, a_1), (z, a_2), (z, a_3), ..\}$ according to the order of the vertices in $\{a_1, a_2, a_3, ..\}$. We represent each tuple with a point set $P = \{(t(y, a_1), t(z, a_1)), (t(y, a_2), t(z, a_2)), ..\}$, where $t(y, a_i)$ is the timestamp of the edge (y, a_i) for all $i = 1, 2, ..$. We store P using a 2-dimensional range tree.

Next, we compute the timespan of each tuple and store it as an interval using an interval tree. We sort linked lists E and E' according to the timestamps of their edges in non-decreasing order. Without loss of generality, let $E = (e_1, e_2, \ldots, e_p)$ and $E' = (e'_1, e'_2, \ldots, e'_p)$, where $t(e_1) \leqslant t(e_2) \leqslant \ldots \leqslant t(e_p)$ and $t(e'_1) \leqslant t(e'_2) \leqslant \ldots \leqslant t(e'_p)$. Now we can create an interval segment $x = [\min(t(e_1), t(e'_1)), \max(t(e_p), t(e'_p))]$ for each tuple to mark its timespan. Following this technique, we compute segments for all tuples and store them in an interval tree $\mathcal{T}$.

Algorithm 3: PreprocessQuad(G)

Input : A relational event graph $G = (V, E)$.
Output: 2D range trees for each tuple of G and an interval tree for G.

Sort the vertices according to their degrees in non-increasing order. Without loss of generality, let $d(v_1) \geqslant d(v_2) \geqslant \cdots \geqslant d(v_n)$.
Create an empty interval tree $I = \emptyset$.
for *each vertex* $v \in V$ **do**
  Set $U[v] \leftarrow \emptyset$.
end
for *each vertex* v_i **do**
  Set $y \leftarrow v_i$.
  for *each vertex* u *adjacent to* y **do**
    for *each vertex* $z \neq y$ *adjacent to* u **do**
      Set $U[z] \leftarrow U[z] \cup \{u\}$.
    end
    for *each vertex* z *with* $|U[z]| \geqslant 2$ **do**
      Let $U[z]$ stores vertices $\{a_1, a_2, \cdots, a_p\}$.
      Store $(y, z, \{a_1, a_2, \cdots, a_p\})$ using linked lists $E = \{(y, a_1), (y, a_2), \cdots, (y, a_p)\}$ and $E' = \{(z, a_1), (z, a_2), \cdots, (z, a_p)\}$.
      Let $e_1 = (y, a_1), e_2 = (y, a_2), \cdots, e_p = (y, a_p)$.
      Let $e'_1 = (z, a_1), e'_2 = (z, a_2), \cdots, e'_p = (z, a_p)$.
      Create a 2D range tree with point set $P = \{(t(e_1), t(e'_1)), (t(e_2), t(e'_2)), \cdots, (t(e_p), t(e'_p))\}$.
      Sort $E = (e_1, e_2, \cdots, e_p)$ and $E' = (e'_1, e'_2, \cdots, e'_p)$ according to increasing order of timestamps. Without loss of generality, let $t(e_1) < t(e_2) < \cdots < t(e_p)$ and $t(e'_1) < t(e'_2) < \cdots < t(e'_p)$.
      Create an interval segment $x = [\min(t(e_1), t(e'_1)), \max(t(e_p), t(e'_p))]$.
      Insert x into interval tree $\mathcal{T}$.
    end
    for *each vertex* z *with* $U[z] \neq \emptyset$ **do**
      Set $U[z] \leftarrow \emptyset$.
    end
  end
  Delete y from G and let G be the resulting graph.
end

Algorithm 4: QueryQuad($\mathcal{T}$, $[i, j]$)

Input : Interval tree $\mathcal{T}$, query interval $[i, j]$.
Output: Total number of quadrangles #Quad within the given query interval.

```
1 Query T using [i, j] horizontal segment and find interval set I' within the query range.
2 Set #Quad ← 0.
3 for each segment x ∈ I' do
4 |   Query 2D range tree with query rectangle [i, i]x[j, j] to find valid set S of edges.
5 |   if |S| ⩾ 2 then
6 |   |   Set #Quad ← (|S| choose 2) + #Quad.
7 |   end
8 end
```

For an illustration, see the example presented in Figure 2.9(a). It shows an RE graph G with four tuples, each tuple is highlighted with a different color. We first store the tuple $(a, b, \{e, f, g\})$ using linked lists $E = (ae, af, ag)$ and $E' = (be, bf, bg)$. Then we create a point set $P = \{(6, 5), (1, 3), (19, 18)\}$ and store it in a two-dimensional range tree. Next, we sort edges in $E = (af, ae, ag)$ and $E' = (bf, be, bg)$ according to their timestamps in increasing order. Thus, we store an interval segment $[\min\{1, 3\}, \max\{19, 18\}] = [1, 19]$ for this tuple in the interval tree. Interval segments and point sets for all tuples are shown in Figure 2.9(b).

***Query* Step:** We query the interval tree $\mathcal{T}$ with a horizontal query segment $[i, j]$ and obtain a set of valid interval segments $I' \subseteq I$. We consider each segment $x_i \in I'$ valid, if it intersects with the query slice indicating that x_i might contain quadrangles that exist within the query slice. To obtain the exact number of quadrangles (#Quad), we need to know which of these valid segments contain at least two sets of paired edges $\{(y, v'), (z, v')\}$ and $\{(y, v''), (z, v'')\}$ such that it makes a quadrangle (y, v', z, v''). Recall that for each segment we have already stored a set of points P that contain information of these paired edges. So for each valid segment, we perform a 2-D rectangular range query on P using a query rectangle with four corner points $(i, i), (i, j), (j, j)$ and (j, i). It returns a set of edges S. If $|S| \geqslant 2$ we add $\binom{|S|}{2}$ to *#Quad*.

Continuing with the same example, Figure 2.9(c) shows all the valid segments after querying an interval tree with query point $q = [6, 17]$. Finally a rectangular query shows that tuples $(a, d, \{c, e, h\})$ and $(b, n, \{m, \ell\})$ have sets of edges that fall within the query range (Figure 2.9(d)). Therefore, we obtain $\binom{3}{2} + \binom{2}{2} = 4$ quadrangles (#Quad) within the query time interval [6,17].

Preprocessing Analysis. In lines 1-5, sorting vertices with respect to their degrees and setting up the lists for all vertices take $O(m + n)$ time. The outer **for** loop (starting from line 6) identifies all quadrangles containing v_i by traversing its neighbours in $O(\sum_{v_i \in V} O(d(v_i) + \sum_{u \in N(v_i)} d(u))$ time, where $N(v_i)$ is the set of neighbours of v_i. So, except for the time needed to construct the range trees and the interval tree, total time required is $O(m+n) + \sum_{v_i \in V} O(d(v_i) + \sum_{u \in N(v_i)} d(u))$. This can be bounded by $O(a(G)m)$ by Lemma 2.1.

Since we can have at most $a(G)m$ tuples in G, there can be at most $a(G)m$ intervals to store in the interval tree. This is a weak upper bound on the number of intervals. For example, for

any complete graph $a(G) = \lceil n/2 \rceil$, whereas for planar graphs $a(G) = O(1)$. Since, each interval considers unique pair of vertices on opposite sides of a quadrangle, there can be at most $\binom{n}{2}$ quadrangles in G. So the number of intervals can be bounded as $\alpha = \min\{a(G)m, \binom{n}{2}\} \leqslant a(G)m$. We maintain 2-dimensional range trees for every tuple identified in G (line 16). We observe that, the total number of points that can be stored in these range trees can be $O(\sum_{v_i \in V} O(d(v_i) + \sum_{u \in N(v_i)} d(u))) = O(a(G)m)$. Therefore, by Theorem 2.4, total time required to build these range trees is $O(a(G)m \log(a(G)m)) = O(a(G)m \log n)$.

Using similar reasoning, in lines 18-19, the interval tree for α intervals can be created in time, $O(\alpha \log \alpha) = O(\alpha \log n)$ (by Theorem 2.5). Therefore, the total time required by Algorithm 3 is $O\big(a(G)m + a(G)m \log n + \alpha \log n\big) = O(a(G)m \log n)$.

Query Analysis. Suppose given a query time interval $q = [i, j]$, we find I' as the set of all valid interval segments that intersects with q (line 1 of Algorithm 4). Let $\gamma = |I'|$, where $\gamma \leqslant \min\{\binom{n}{2}, a(G)m\}$. Reporting all γ segments from the interval tree requires $O(\log(\min\{\binom{n}{2}, a(G)m\}) + \gamma) = O(\log n + \gamma)$ time [15]. Rectangular range queries on γ range trees (line 3-8) take time $O\big(\sum_{i=1}^{\gamma} \log n + k_i\big) = O(\gamma \log n + w)$, where k_i is the number of reported quadrangles in segment i and w is #Quad in $G_{i,j}$.

Space: Total number of intervals in our problem is bounded by $O(a(G)m)$. By Theorem 2.5, the interval tree uses $O(a(G)m)$ space. Therefore, total required space is dominated by the space required by range trees, i.e., $O(a(G)m \log(a(G)m)) = O(a(G)m \log n)$ (Theorem 2.4). The following theorem summarizes the results for quadrangle counting in a relational event graph.

THEOREM 2.23. *Given an RE graph* G *with* m *edges, the number of quadrangles in the query time slice* $[i, j]$ *can be determined in* $O(\gamma \log n + w)$ *time, where* $\gamma \leqslant \min\{\binom{n}{2}, a(G)m\}$ *and* w *is the number of reported quadrangles. Preprocessing takes* $O(a(G)m \log n)$ *time and* $O(a(G)m \log n)$ *space.*

2.6 APPLICATIONS

We now discuss some of the applications of these data structures. The problem of finding whether an MST exists in a queried graph slice can be directly applied in the design of any type of connected networks in a query time interval, such as, telecommunication, transportation, computer networks or electrical grids. The problem of graph edit distance for spanning forests provides a cost effective measure (i.e., the number of operations required) to maintain the connectivity within the connected components of a network. We find that the applications of subgraph counting data structures are specifically pertinent to some useful social network analyses. So we further elaborate them below.

2.6.1 *Clustering Coefficient*

Clustering coefficient is also known as network transitivity and is an important model for extracting community structure from social networks [29]. Multiple definitions are available for clustering coefficient depending on the context in which it is being used and the type of network is being

studied. In this paper we define the clustering coefficient of a graph G as the measure of the degree to which vertices in G tend to cluster together [30]. It is formulated as follows,

$$CC(G) = \frac{\text{total number of closed triplets}}{\text{total number of triplets}} = \frac{3 \times \text{total number of triangles}}{\text{total number of 2-paths}}$$

Since clustering coefficient of any graph is the ratio of total number of closed triplets over all the open and closed triplets, its value ranges between 0 and 1. For an illustration, see Figure 2.1. The clustering coefficient of $G_{1,5}$ in Figure 2.1 (a) is $CC(G_{1,5}) = \frac{3\times1}{7} = 0.43$. Similarly, $CC(G_{2,4}) = \frac{3\times1}{3} = 1$ and $CC(G_{2,5}) = \frac{3\times1}{5} = 0.6$. For bipartite graphs, the clustering coefficient is based on the number of quadrangles (C_4). More formally, the clustering coefficient of a bipartite graph is the ratio of the number of quadrangles to the number of 3-paths [30], i.e.,

$$CCB(G) = \frac{4 \times \text{total number of quadrangles } (C_4)}{\text{total number of 3-paths}}$$

Thus, for general graphs we obtain the following results.

COROLLARY 2.24. *Let G be an RE graph consisting of m edges. Let $\mathcal{K}$ be the total number of 2-paths in G. The problem of computing the clustering coefficient of $G_{i,j}$ can be reduced to dominance counting in $(\alpha(G)m)$ time using $O(m+n+\mathcal{K})$ space and each query can be answered in $O(\log \mathcal{W}/\log\log \mathcal{K})$ time, where $\mathcal{W}$ is the width of the query window.*

For clustering coefficient of a bipartite graph slice $G_{i,j}$, we can count the total number of 3-paths in $O(m+n)$ time and the number of quadrangles in $O(\gamma \log n + w)$ time, where w is the number of total quadrangles in $G_{i,j}$ and $\gamma \leqslant \min\{\binom{n}{2}, (\alpha(G)m)\}$. Note that the total preprocessing time is dominated by the quadrangle processing time. The following corollary summarizes the result.

COROLLARY 2.25. *Let $G = (V, E)$ be a bipartite RE graph with n vertices and m edges. The graph G can be preprocessed in $O(\alpha(G)m \log n)$ time and the clustering coefficient of a bipartite graph slice $G_{i,j}$ can be computed in $O(\gamma \log n + w)$ time, where w is the total number of quadrangles in $G_{i,j}$ and $\gamma \leqslant \min\{\binom{n}{2}, (\alpha(G)m)\}$.*

2.6.2 *Embeddedness and Neighborhood Overlapping*

Embeddedness of an edge (u, v), denoted as $emb(u, v)$, in a graph G is the number of common neighbors the two endpoints u and v have, i.e., $emb(u, v) = |N(u) \cap N(v)|$[16]. Embeddedness of an edge (u, v) in a graph represents the trustworthiness of its neighbors, and the confidence level in the integrity of the transactions that take place between two vertices u and v. Note that the embeddedness of an edge (u, v) represents the number of triangles that share (u, v).

The value of embeddedness is also used to compute the *neighborhood overlap* of an edge (u, v), denoted as $NOver(u, v)$, and defined as the ratio of the number of vertices who are neighbors of both u and v, and the number of vertices who are neighbors of only one of them [16].

$$NOver(u, v) = \frac{emb(u, v)}{|N(u) \cup N(v)| - emb(u, v) - 2}$$

Neighborhood overlap of an edge represents the strength (in terms of connectivity) of that edge in its neighborhood. The neighborhood overlap of an entire graph G is defined as the average of the neighborhood overlap values of all the edges of G, i.e., $\mathrm{NOver}(G) = \frac{1}{|E|}\sum_{k=1}^{|E|}\mathrm{NOver}(e_k)$ [28].

The result for computing the neighborhood overlap of a query graph slice has been presented in [11] and is also stated below. These results use the time windowed data structures presented in this paper and also include colored range searching data structures [5, 17, 18].

THEOREM 2.26. *(Theorem 6 in [11]) Given an RE graph $G = (V, E)$ the problem of computing the average neighborhood overlap of $G_{i,j}$ can be reduced to the colored range counting in $O(mn)$ time. For a query time slice $[i, j]$, $\mathrm{NOver}(G_{i,j})$ can be computed in $O(\log^2 n + (t + s)\log n)$ time, where t is the number of edges in $G_{i,j}$ with positive embeddedness and s is the number of edges having some neighboring edges in $E_{i,j}$.*

2.7 CONCLUSION

In this paper, we present time window data structures to answer various queries involving both decision and reporting problems based on relational event graphs. In the first part of the paper, we provide dynamic algorithm based data structures that can answer time windowed decision problems under monotone graph properties, such as disconnectedness and bipartiteness, and can report the weight of a minimum spanning tree, the minimum spanning interval and the graph edit distance for obtaining a spanning forest for a query graph slice. In the second part of the paper, we present window data structures for counting subgraphs of a given pattern. We consider 2-paths (for general graphs), 3-paths (for bipartite graphs), complete subgraphs of order $\ell \geqslant 3$ and quadrangles as valid patterns. We also show some applications of our subgraph counting results for computing graph parameters that are important for social network analysis, such as clustering coefficients, embeddedness and neighborhood overlapping.

Remarks. The query time of the time windowed decision problem for monotone graph properties can be improved so that given a query interval $[i, j]$, with $1 \leqslant i \leqslant j \leqslant n$, a data structure of size $O(n)$ can answer a query in $O(1)$ time as follows. Let A be an array of size m. For each $A[k]$, with $1 \leqslant k \leqslant n$, we store the index ℓ such that $[k, \ell]$ is the maximal subsequence of edges that satisfies some monotone property $\mathcal{P}$ (see Section 2.3.1). If there is no such maximal subsequence starting with some k, with $1 \leqslant k \leqslant m$, then we store $A[k] = +\infty$. For a given query interval $q = [i, j]$, if $j \leqslant A[i]$ we report that $G_{i,j}$ satisfies $\mathcal{P}$.

OPEN PROBLEM 1. *The techniques we presented here require the type of subgraphs to be specified at the preprocessing step. An interesting variation of this problem would be to develop data structures that can count total number of subgraphs of any pattern (specified at the query time) or a fixed graph on a small number of vertices within a given query time slice.*

OPEN PROBLEM 2. *Some of our data structures require $O(m \log n)$ space to answer window queries in $O(\log n)$ time, for example computing the $\mathcal{GED}$ or the minimum spanning interval for the queried graph slice, or finding whether a minimum spanning tree for G exists in the query interval. Data structures of size $O(m^2)$ can be built by precomputing answers for all possible edge-pairs so that queries can be answered in $O(1)$ time. It will be interesting to examine the non-trivial bounds on the time-space trade-off for these window problems.*

BIBLIOGRAPHY

[1] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3):209–223, 1997. URL: https://doi.org/10.1007/BF02523189, doi:10.1007/BF02523189.

[2] M. J. Bannister, W. E. Devanny, M. T. Goodrich, J. A. Simons, and 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.

[3] M. J. Bannister, C. DuBois, D. Eppstein, and P. Smyth. Windows into relational events: Data structures for contiguous subsequences of edges. In *Proceedings of the 24th ACM-SIAM SODA*, pages 856–864. SIAM, 2013.

[4] D. Bokal, S. Cabello, and D. Eppstein. Finding all maximal subsequences with hereditary properties. In *31st International Symposium on Computational Geometry, SoCG 2015*, pages 240–254, 2015.

[5] P. Bozanis, N. Kitsios, C. Makris, and A. Tsakalidis. New upper bounds for generalized intersection searching problems. In *ICALP*, pages 464–474. Springer, 1995.

[6] U. Brandes, J. Lerner, and T. A. Snijders. Networks evolving step by step: Statistical analysis of dyadic event data. In *International Conference on Advances in Social Network Analysis and Mining, ASONAM'09*, pages 200–205. IEEE, 2009.

[7] T. M. Chan, K. G. Larsen, and M. PătraC$_s$cu. Orthogonal range searching on the RAM, revisited. In *Proc. of the 27th symposium on Computational geometry*, pages 1–10. ACM, 2011.

[8] T. M. Chan and S. Pratt. Two approaches to building time-windowed geometric data structures. In *32nd International Symposium on Computational Geometry, SoCG 2016*, pages 28:1–28:15, 2016.

[9] Chanchary, F., Maheshwari, A.: Counting subgraphs in relational event graphs. In: WALCOM: Algorithms and Computation - 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9627, pp. 194–206. Springer (2016), https://doi.org/10.1007/978-3-319-30139-6_16

[10] Chanchary, F., Maheshwari, A.: Time windowed data structures for graphs. Journal of Graph Algorithms and Applications 23(2), 191–226 (2019)

[11] F. Chanchary, A. Maheshwari, and M. Smid. Querying relational event graphs using colored range searching data structures. In *Proceedings of the third international conference on Algorithms and Discrete Applied Mathematics, Lecture Notes in Computer Science*, volume 10156, pages 83–95. Springer, 2017.

[12] F. Chanchary, A. Maheshwari, and M. Smid. Window queries for problems on intersecting objects and maximal points. In *Proceedings of the fourth international conference on Algorithms and Discrete Applied Mathematics, Lecture Notes in Computer Science*, volume 10743, pages 199–213. Springer, 2018.

[13] B. Chazelle and L. J. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.

[14] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[15] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational geometry*. Springer, 2000.

[16] D. Easley and J. Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[17] P. Gupta, R. Janardan, S. Rahul, and M. Smid. Computational geometry: Generalized (or colored) intersection searching. *Handbook of Data Structures and Applications, 2nd edn.(in press) Google Scholar*, In press.

[18] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

[19] F. Harary. *Graph theory.* Addison-Wesley, Reading, MA, 1969.

[20] F. Harary and H. J. Kommel. Matrix measures for transitivity and balance. *Journal of Mathematical Sociology*, 6(2):199–210, 1979.

[21] M. R. Henzinger and V. King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM (JACM)*, 46(4):502–516, 1999.

[22] J. Holm, K. De Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, 2001.

[23] P. Holme. Network reachability of real-world contact sequences. *Physical Review E*, 71(4):046119, 2005.

[24] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.

[25] J. JaJa, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Algorithms and Computation*, pages 558–568. Springer, 2004.

[26] T. Kloks, D. Kratsch, and H. Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters*, 74(3):115–121, 2000.

[27] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[28] N. Meghanathan. A greedy algorithm for neighborhood overlap-based community detection. *Algorithms*, 9(1):8, 2016.

[29] M. E. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[30] T. Opsahl. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(2):159–167, 2013.

[31] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *Journal of computer and system sciences*, 26(3):362–391, 1983.

[32] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976.

[33] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[34] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *nature*, 393(6684):440–442, 1998.

[35] P. Zhang, J. Wang, X. Li, M. Li, Z. Di, and Y. Fan. Clustering coefficient and community structure of bipartite networks. *Physica A: Statistical Mechanics and its Applications*, 387(27):6869–6875, 2008.

Table 2.1: Summary of previous and new results using relational event graph $G = (V, E)$. Here $n = |V|$, $m = |E|$, $\mathcal{W} = j - i + 1$ is the width of the query window $[i, j]$, r (respectively, f) is the number of past (respectively, future) neighbors of an edge, h is the h-index (the largest number h such that the graph contains at least h vertices of degree at least h), z is the size of the output, s is the number of edges having neighboring edges, t is the number of edges that are contained in some triangles, p is the number of vertex pairs having some common neighbors, k is the number of vertices having some neighbors in $G_{i,j}$, ϵ is a small positive constant, X and Y are set of output edges, $a(G) = O(\sqrt{m})$ is the arboricity of G (the min number of edge-disjoint spanning forests into which G can be partitioned), $\gamma \leqslant \min\{\binom{n}{2}, a(G)m\}$, ℓ is the order of a complete subgraph, w is the number of a specific subgraph (i.e., 2-path, 3-path (in bipartite graphs), quadrangle, or complete subgraph) in the query interval, and $\mathcal{K}$ is the total number of the same subgraph in G. All results are in 'big-oh' notations. Results of this paper are marked with $(*)$.

Problem	**Preprocessing**	**Space**	**Query time**	**Ref.**
Reciprocated edges, degree-k vertices	$m + n$	$m + n$	$\log \mathcal{W} / \log\log m$	[3]
Reachable vertices,				
Connected and tree components	$m \log n$	$m + n$	$\log \mathcal{W} / \log\log m$	[3]
Edges with fixed number of neighbors	$(r + f)m$	$m + n$	$\log \mathcal{W} / \log\log m$	[3]
Triad closure	hm	$m + n$	$\log \mathcal{W} / \log\log m$	[3]
Number of vertices, Graph density	$m + n$	$m + n$	$\log n$	[11]
Degrees of vertices	$m + n$	$m + n$	$\log^2 n + z$	[11]
k-stars	$m + n$	$m + n$	$\log n + z$	[11]
h-index (approx.)	$n \log^2 n$	$n \log n$	$\log^2 n + h \log n$	[11]
Embeddedness	$a(G)m$	$a(G)m$	$\log^2 n + t \log n$	[11]
$\mathrm{NOver}(G_{i,j})$	mn	mn	$\log^2 n + (t + s) \log n$	[11]
$\mathrm{NOver}(G_{i,j})$-bipartite	$a(G)m$	$a(G)m$	$\log^2 n + p \log n + k$	[11]
Influenced vertices	$m \log n$	$m \log n$	$\log n + z$	[11]
Monotone properties*				
Bipartiteness*	$m \log^3 n$	$m + n \log n$	$\log n$	Th. 2.10
Disconnectedness*	$m \log^2 n$	$m + n \log n$	$\log n$	Th. 2.11
MST*	$m \log^4 n$	$m \log n$	$\log n$	Th. 2.13
Min spanning interval*	$m \log^4 n$	$m \log^\epsilon n$	$\log\log n$	Th. 2.14
$\mathcal{GED}$ (Forest)*	$m \log n$	$m \log n$	$\log n + \lvert X\rvert$	Th. 2.18
2-paths*	m	m	n	Th. 2.19
	$a(G)m$	$m + n + \mathcal{K}$	$\log \mathcal{W} / \log\log \mathcal{K}$	Th. 2.22
3-paths*	$m + n$	$m + n$	$m + n$	Th. 2.19
	n^4	$m + n + \mathcal{K}$	$\log \mathcal{W} / \log\log \mathcal{K}$	Th. 2.22
Complete Subgraphs, fixed $\ell \geqslant 3$*	$a(G)^{\ell-2}m$	$m + n + \mathcal{K}$	$\log \mathcal{W} / \log\log \mathcal{K}$	Th. 2.20
Complete Subgraphs, $\ell \geqslant 3$*	$a(G)^{\ell-1}m$	$m + n +$ $\mathcal{K} \log \mathcal{K} / \log\log \mathcal{K}$	$(\log \mathcal{K} / \log\log \mathcal{K})^2$	Th. 2.21
Quadrangles*	$a(G)m \log n$	$a(G)m \log n$	$\gamma \log n + w$	Th. 2.23

3

WINDOWS DATA STRUCTURES USING COLORED RANGE SEARCH

We present a general approach for analyzing structural parameters of a relational event graph within arbitrary query time intervals using colored range query data structures. Relational event graphs generally represent social network datasets, where each graph edge carries a timestamp. We provide data structures based on colored range searching to efficiently compute several graph parameters (e.g., density, neighborhood overlap, h-index).

This chapter is a combination of results that have been presented in the 3rd International Conference on Algorithms and Discrete Applied Mathematics (CALDAM 2017) [4] and results that have been accepted for publication in the Journal of Discrete Applied Mathematics [3].

3.1 INTRODUCTION

A *relational event (RE) graph* $G = (V, E)$ is defined to be an undirected graph with set of vertices V and a set of edges (or relational events) $E = \{e_k \mid 1 \leqslant k \leqslant m\}$ between pairs of vertices. We assume that each edge has a unique *timestamp*. We denote the timestamp of an edge $e_k \in E$ by $t(e_k)$. Without loss of generality, we assume that $t(e_1) < t(e_2) < \cdots < t(e_m)$. Given a relational event graph G, for a pair of integers $1 \leqslant i \leqslant j \leqslant m$, we define the *graph slice* $G_{i,j} = (V', E' = \{e_i \cup e_{i+1} \cup \cdots \cup e_j\})$, where V' is the set of vertices incident on edges of E'. In this chapter, for a query time interval $[i, j]$, where $1 \leqslant i \leqslant j \leqslant m$, we are interested in answering questions about various graph parameters on the graph slice $G_{i,j}$.

A social network can naturally be represented by an RE graph, where each vertex of the graph represents an entity of the social network, and edges represent communication events between pair of entities occurred at some specific time. The RE graph model was first proposed by Bannister et al. [1]. They presented data structures to find the number of connected components, number of components containing cycles, number of vertices with some predetermined degree and number of reachable vertices on a time-increasing path within a query time window. Later, Chanchary and Maheshwari [2] presented data structures to solve subgraph counting problems for triangles, quadrangles and complete subgraphs in RE graphs. In this paper, we present a general approach to construct data structures on a set of colored points in $\mathbb{R}^d$, $(d \geqslant 1)$, that support colored (or generalized) range queries, and efficiently count and/or report various structural parameters of the underlying RE graph G within a query pair of indices $[i, j]$.

3.1.1 *Preliminaries*

We define some structural graph parameters that we want to compute using our window data structures. One of the basic indicators for measuring graph structure is the *density* of a graph. It

evaluates how close the graph is to a complete graph. The density of an undirected simple graph $G = (V, E)$ is defined as $D(G) = \frac{|E|}{\binom{|V|}{2}}$ [5].

In social networks, center vertices of k-stars are considered as *hubs*. In network analysis, hubs have been extensively studied as they are the basis of many tasks, for example web search and epidemic outbreak detection [6]. A k-*star* is defined to be a complete bipartite graph $K_{1,k}$, i.e., a tree with one internal node and k leaves. The h-*index* is the largest number h such that the graph contains h vertices of degree at least h. For any graph with m edges, $h = O(\sqrt{m})$ [1].

Embeddedness of an edge (u, v), denoted as $emb(u, v)$, in a network is the number of common neighbors the two endpoints u and v have, i.e., $emb(u, v) = |N(u) \cap N(v)|$[8]. Embeddedness of an edge (u, v) in a network represents the trustworthiness of its neighbors, and the confidence level in the integrity of the transactions that take place between two vertices u and v [8]. A *local bridge* in a graph G is an edge whose endpoints have no common neighbour. *Neighborhood overlap* of an edge (u, v), denoted as $NOver(u, v)$, is the ratio of the number of vertices who are neighbors of both u and v, and the number of vertices who are neighbors of only one of them [8].

$$NOver(u, v) = \frac{emb(u, v)}{|N(u) \cup N(v)| - emb(u, v) - 2} \tag{3.1}$$

In the denominator, u or v are not counted as the neighbor of one another. Neighborhood overlap of an edge represents the strength (in terms of connectivity) of that edge in its neighborhood. The neighborhood overlap of an entire graph G is defined as the average of the neighborhood overlap values of all the edges of G, i.e., $NOver(G) = \frac{1}{|E|}\sum_{k=1}^{|E|} NOver(e_k)$ [9]. Suppose $G = (V = A \cup B, E)$ is a bipartite graph. Neighborhood overlap of a pair of vertices $u, v \in A$ of G is defined as the following ratio [8].

$$NOverB(u, v) = \frac{emb(u, v)}{|N(u) \cup N(v)| - emb(u, v)} \tag{3.2}$$

In social network analysis, this bipartite graph is known as the *affiliation network*. An affiliation network represents how entities of a network (i.e., vertices in A) are affiliated with other groups or activities (i.e., vertices in B).

Modularity of a network is a measure that quantifies the quality of a given network division into disjoint partitions or communities [5]. Many real world complex networks, e.g., the world wide web, and biological or social networks, demonstrate some forms of community structures that are important for various topological studies. The *modularity of a vertex pair* (u, v) is defined as $\frac{d(u) \times d(v)}{2|E|}$, where $d(u)$ is the degree of vertex u [5].

Analysis of the *diffusion phenomena* in graphs is a very widely studied research area in many communities including computer science, social sciences and epidemiology. Although the basic model of diffusion shares common properties across various disciplines, it is highly context dependent [11, 12, 13]. In particular, the spread of information in any social network requires some influence from a set of designated agents (vertices) and depends on the connectivity of the network. We want to solve the problem of reporting *influenced vertices* in an RE graph slice using the following model.

Suppose, $G = (V, E)$ is an RE graph with n vertices and m edges with a fixed set of influential vertices $V' \subseteq V$. Let $f : V \to N$ be a function assigning threshold values to vertices such that, (a)

$f(v) = 0$ if v is an influential vertex; (b) $1 \leqslant f(v) \leqslant d(v)$ otherwise, where $d(v)$ is the degree of $v \in V$. A vertex can be influenced only if it is on a *path of influence* (we define it in Definition 3.1). A simpler model for counting influential vertices has been presented in [1].

DEFINITION 3.1. *For a pair of vertices u and v, and a positive integer r, a path denoted by $\pi = (u = v_1, v_2, v_3, \cdots, v_k, v_{k+1} = v)$ is a path of influence with parameter r, if the following holds:*

1. *u is an influential vertex and v is a non-influential vertex*
2. *$v_2, v_3, \cdots, v_k$ are influenced vertices (see Definition 3.2)*
3. *$t(v_i, v_{i+1}) < t(v_{i+1}, v_{i+2})$*
4. *$t(v_k, v_{k+1}) - t(v_1, v_2) \leqslant r$.*

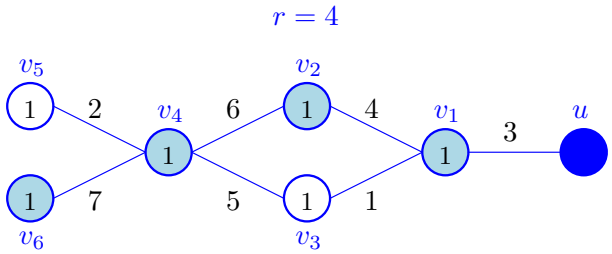


Figure 3.1: A path of influence $\pi = (u, v_1, v_2, v_4, v_6)$ starting from an influential vertex u (shaded dark blue) to vertices v_1, v_2, v_4 and v_6 (shaded light blue) given $r = 4$. The value of $f(v) = 1$ for all $v_k \in V$, with $k = 1, \ldots, 6$. Vertices v_3 and v_5 (opaque) are not influenced by u since there is no path with increasing timestamps from u to these vertices.

DEFINITION 3.2. *A vertex v is* influenced with respect to r *if either v is an influential vertex, i.e., $v \in V'$ or v is a non-influential vertex and there are at least $f(v)$ edge-disjoint paths of influence with parameter r by which v can be reached from some influential vertices.*

See Figure 3.1 for an example. Here vertices v_1, v_2, v_4 and v_6 are influenced, while vertices v_3 and v_5 are not.

We assume an influential vertex u can influence other vertices $v_1, v_2, \cdots, v_{k+1}$ on a path π of influence such that the time difference $t(v_k, v_{k+1}) - t(v_1, v_2) \leqslant r$, where $r > 0$. After r rounds, v_i's influence becomes inactive on other vertices on this particular path of influence. This is known as the degradation of influence in social networks. A similar model has been proposed by Gargano et al. [13] for non-temporal graphs.

3.1.2 *New Results*

Let G be a relational event graph consisting of m edges and n vertices, and let $q = [i, j]$ be an arbitrary query time interval, where $1 \leqslant i \leqslant j \leqslant m$. Our data structures can efficiently answer queries on computing the graph density, number of vertices, degrees of all vertices, k-stars, h-index, embeddedness of edges, average neighborhood overlap (for general and bipartite graphs) and the number of influenced vertices in the graph slice $G_{i,j}$. The contributions of this paper are

summarized in Table 3.1. Some of the queries mentioned above are obtained by reducing the problem to a new colored range query problem stated as follows: Given a set of n colored points on the real line and a fixed threshold value k, we can construct an insertion-only persistent data structure in $O(n \log n)$ time using $O(n)$ space that can report those colors that have at least k elements within any query interval $q = [a, b]$ in $O(\log n + w)$ time, where w is the number of reported colors (Theorem 3.6).

Table 3.1: Summary of Results. Here, w is the size of the output, h is the h-index, s is the number of edges having neighboring edges, t is the number of edges with positive embeddedness, p is the number of vertex pairs having some common neighbors, k is the number of vertices having some neighbors in $G_{i,j}$, $a(G)$ is the arboricity of G, and $n = |V|$, $m = |E|$. The reduction time refers to time required to reduce the window problem into colored range searching problem and does not include the time to build the data structure.

Problems	Reduction time	Space	Query time	
$\lvert V_{i,j}\rvert$	$O(m+n)$	$O(m \log n)$	$O(\log n)$	Th. 3.7
$D(G_{i,j})$	$O(m+n)$	$O(m \log n)$	$O(\log n)$	Th. 3.7
$d(v \in V_{i,j})$	$O(m+n)$	$O(m \log n)$	$O(\log^2 n + w)$	Th. 3.8
k-stars	$O(m+n)$	$O(m \log n)$	$O(\log n + w)$	Th. 3.9
h-index (approx.)	$O(m+n)$	$O(m \log n)$	$O(\log^2 n + h \log n)$	Th. 3.10
Embeddedness	$O(a(G)m)$	$O(a(G)m)$	$O(\log^2 n + t \log n)$	Th. 3.11
$NOver(G_{i,j})$	$O(mn)$	$O(mn \log n)$	$O(\log^2 n + (t+s) \log n)$	Th. 3.11
$NOverB(G_{i,j})$	$O(a(G)m)$	$O(a(G)m)$	$O(\log^2 n + p \log n + k)$	Th. 3.12
Influenced vertices	$O(m \log n)$	$O(n^2)$	$O(\log n + w)$	Th. 3.13

3.1.3 *Organization*

The rest of this paper is organized as follows. Section 2 presents some existing and new results on colored range query data structures. Section 3 presents the general approach for solving problems in RE graphs using colored range searching queries. In Section 4, we provide algorithms and their analysis for solving window queries. Section 5 concludes this paper.

3.2 COLORED RANGE SEARCHING DATA STRUCTURES

Colored range searching problems are variations of the standard range searching problems, where a set of colored objects (e.g., points, lines or rectangles) is given. Typically, the color of an object represents its category. In the generic instance of the range searching problems, a set of objects S is to be preprocessed into a data structure so that given a query range q, it can efficiently answer counting or reporting queries based on the intersection of q with the elements in S. In the colored version, the query should efficiently report those colors that have at least one object intersecting the query range. For example, we have a set of points $S \in \mathbb{R}^2$ and each point is colored by one of the four different colors, i.e., $C = \{c_1, c_2, c_3, c_4\}$ (see Figure 3.2). Suppose a query rectangle $q = [a, b] \times [c, d]$ is given. The standard colored range counting (reporting) query will count

(report) the number of colors that have at least one point inside q. In this example, there are three colors (red, green and blue) that have points inside q, hence they will be reported. A different version namely *Type-2 counting problem* reports the number of points for each color intersected by q as a pair of values $(c_k, \#c_k\text{-colored points intersected by } q)$. Following our example, the type-2 counting query will generate the output as $(\mathtt{red}, 2), (\mathtt{green}, 2), (\mathtt{blue}, 1)$. A variety of colored range searching problems have been studied extensively, see for example [14, 15, 16, 17].

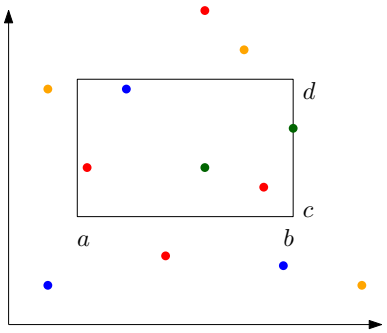


Figure 3.2: Example of colored range queries.

3.2.1 k-*threshold color queries*

We wish to preprocess a set S of n colored points on a horizontal line so that given a threshold k, where $k \geqslant 0$, an integer, and a query interval $q = [a, b]$, we can quickly report the colors that have at least k points in $q \cap S$. First we present a chaining technique by which we can build a data structure of size $O(n \log n)$ in $O(n \log n)$ time to report each query in $O(\log^2 n + w)$ time, where w is the number of reported colors. Next we show an insertion-only persistent structure that improves both total space and the query time by a factor of $O(\log n)$.

3.2.1.1 *Chaining Technique*

We transform each 1-dimensional point to a 3-dimensional point by applying the following chaining technique. For each color c, we sort the distinct points of this color by increasing values of their x-coordinates. For each point x_i of color c, we map x_i to $x'_i = (x_{i-1}, x_i, x_{i+k-1}) \in \mathbb{R}^3$ (indices are with respect to the sorted order) and assign it color c. For the leftmost point x_ℓ, let $x_{l-1} = -\infty$ and for the rightmost point x_r, let $x_{r+1} = +\infty$. Let S' be the new set of points in $\mathbb{R}^3$. Now a given query interval $q = [a, b]$ is mapped to a new query $q' = (-\infty, a) \times [a, +\infty) \times (-\infty, b]$.

LEMMA 3.3. *There are at least k points of color c in the query slice $q = [a, b]$ if and only if there is a point of color c in $q' = (-\infty, a) \times [a, +\infty) \times (-\infty, b]$. Moreover, the c-colored point in q' is unique, if there exists any.*

Proof. Let $x'_i = (x_{i-1}, x_i, x_{i+k-1})$ be a c-colored point in q' for some c-colored point x_i in the original set of points. Since x'_i is in q', then $x_i \geqslant a$ and $x_{i+k-1} \leqslant b$ which indicates that starting from x_i there are at least k c-colored elements in the original query interval $[a, b]$. Thus $x_i \in [a, b]$.

Assume there are at least k points of color c in q. Let x_i be the leftmost point of color c in the query interval $[a, b]$. Therefore, $a \leqslant x_i \leqslant b$ and the k-th element from x_i, i.e., $x_{i+k-1} \leqslant b$. Moreover, since x_i is the leftmost point in $[a, b]$ of color c, we know $x_{i-1} \in (-\infty, a)$, i.e., $x_{i-1} < a$. It follows that $x'_i = (x_{i-1}, x_i, x_{i+k-1})$ is in q'.

We prove the uniqueness of point x'_i in q' by contradiction. Suppose, there is a second c-colored point y'_i in q'. It implies that $y_i \geqslant a$ and $y_{i+k-1} \leqslant b$. Since x_i is the leftmost point in $[a, b]$, it must be the case that $y_i > x_i$ and thus $y_{i-1} \geqslant x_i \geqslant a$. Then it contradicts the assumption that $y'_i \in q'$. □

Now we have a set of n colored points in $\mathbb{R}^3$. We build a standard 2-dimensional range tree [7], where the first level of the tree is based on the x-coordinates of the points. At the second level of the tree, for each canonical node we build a priority search tree (PST) [10] using the y and the z-coordinates of our 3-dimensional points. There can be at most $O(\log n)$ canonical nodes on the search path of a range tree. For each canonical node, the second level PST takes linear space. So, total space requirement is $O(n \log n)$. This data structure can be built naively in $O(n \log^2 n)$ time. This time can be improved by building PSTs in linear time by bottom-up heap construction technique using points sorted on z-coordinates [7]. Therefore, total time required to built this data structure becomes $O(n \log n)$. PSTs support insertions and deletions in $O(\log n)$ time and can report w points lying inside a grounded query rectangle in $O(\log n + w)$ time. We query the first level of the range tree using $(-\infty, a)$. In the second level, we query using $[a, +\infty) \times (-\infty, b]$. Thus, given any query interval $q = [a, b]$ this data structure can report the colors that have at least k elements in q in $O(\log^2 n + w)$ time, where w is the number of reported colors.

THEOREM 3.4. *Given a set of n colored points on the real line and a fixed threshold value k, we can build a data structure in $O(n \log n)$ time using $O(n \log n)$ space that can report those colors that have at least k elements within any query interval $q = [a, b]$ in $O(\log^2 n + w)$ time, where w is the number of reported colors.*

Now we present an efficient insertion-only persistence based data structure for this problem.

3.2.1.2 *The Static Counting Problem:*

At first, we assume k is fixed and solve a similar k-threshold color counting problem where the query interval is of the form $q = [a, \infty)$. Suppose, S is a set of n colored points. For each color c, we sort all points of color c by non-decreasing x co-ordinate and let S_c be this sorted list. Let X be another sorted list, where each element X_c of X is the k-th largest element in S_c. For a query interval $[a, \infty)$, we start from the right end of list X, walk up till we reach a and report all points in between. Total space required for this structure is $O(n)$. Queries can be answered in time $O(w)$, where w is the number of reported colors.

LEMMA 3.5. *Let $k \geqslant 1$ be a fixed integer. A set S of n colored points on the real line can be preprocessed into a data structure of size $O(n)$ such that the number of colors that have at least k points contained in any query interval $q = [a, \infty)$ can be reported in $O(w)$ time, where w is the number of reported colors. Total preprocessing time is $O(n \log n)$.*

3.2.1.3 *The Insertion-only Persistent Structure:*

Now we solve the problem of k-threshold color queries for a query interval $q = [a, b]$. The idea is as follows. We first sort the list S of n colored points. Then we start with an empty list L. We insert the points of S, according to their sorted order, into L. During this sequence of insertions, we maintain the data structure of Lemma 3.5. A persistent version of this list will allow us to answer queries for arbitrary intervals $[a, b]$.

During the insertions, we maintain the following information: For each color c, L_c is a sorted list containing the k largest elements among all points having color c that have been inserted so far. In case fewer than k elements of color c have been inserted, L_c stores all these elements having color c. We also maintain the number $size(L_c)$ of elements in the list L_c. Note that $size(L_c) \leqslant k$ at any moment. We maintain a list L that stores the following elements (in sorted order): For each color c with $size(L_c) = k$, the list L stores the k-th largest element having color c. Finally, we maintain a balanced binary search tree T whose leaves store, in sorted order, the elements of the current list L. This tree will allow us to search and update the list L.

Initially, the list L, the tree T, and all lists L_c are empty. Moreover, $size(L_c)$ is zero for each color c.

When we insert a point p of color c, we add p at the end of L_c and increase $size(L_c)$ by one. In case $size(L_c) = k$, we insert p into T and L. In case $size(L_c) = k + 1$, we delete, from T, L, and L_c, the smallest element of L_c, we insert p into T and L, and we decrease $size(L_c)$ by one. See Algorithm 5 for details.

Algorithm 5: k-threshold(S, k)

Input : Set S of n colored points, a positive integer k.

```
1  Sort S in the non-decreasing order of the x-coordinate values of n colored points.
2  Set T, L and all L_c empty.
3  Set all size(L_c) = 0.
4  for each point p in S do
5      Let c be the color of p.
6      Add p at the end of L_c.
7      Set size(L_c) = size(L_c) + 1.
8      if size(L_c) = k then
9          Insert p into T and L.
10     if size(L_c) = k + 1 then
11         Set q = the first element of L_c.
12         Remove q from T, L and L_c.
13         Insert p into T and L.
14         Set size(L_c) = size(L_c) − 1.
```

Sorting S takes $O(n \log n)$ time. Total preprocessing time required for inserting n points into tree T is at most $O(n \log n)$. The amount of memory changes in L, per insertion, is $O(1)$. Total space required to maintain T is $O(n)$. Given a query interval $[a, b]$, we do a binary search on b in the sorted sequence S. The result of this search gives us the version number of L in which we

report all colors that occur at least k times in the query interval $[a, \infty)$. This gives a query time of $O(\log n + w)$, where w is the number of reported colors.

THEOREM 3.6. *Let $k \geqslant 1$ be a fixed integer. A set S of n colored points on the real line can be preprocessed into a persistent data structure of size $O(n)$ in time $O(n \log n)$ such that the number of colors that have at least k points contained in any query interval $q = [a, b]$ can be reported in $O(\log n + w)$ time, where w is the number of reported colors.*

3.3 GENERAL APPROACH FOR MODELING PROBLEMS

In this paper we present the following general approach to solve problems in RE graphs. Suppose an RE graph $G = (V, E)$ is given and we want to answer queries about some structural parameters of G. To solve each problem, we define a set of colors $C = \{c_1, c_2, \cdots, c_p\}$, where each color $c_k \in C$ is encoded as an integer in the range $[1, p]$ for some integer p. We process each edge $e = (u, v)$ of G according to the timestamps of the edges in increasing order and scan either each adjacent edge of e or each neighboring vertex of both u and v. Depending on the problem in hand, our algorithm associates C either to the set of vertices V (i.e., $|C| = n$) or to the set of edges E (i.e., $|C| = m$). When each vertex $v_k \in V$ is associated with a color c_k, the algorithm scans each neighbor N_i of v_k, where $1 \leqslant i \leqslant d(v_k)$, generates a c_k colored point p, and assigns the timestamp $t(v_k, N_i)$ as p's coordinate value (see Figure 3.3(a)). Similarly, when each edge $e_k \in E$ is associated with a color c_k, the algorithm scans each adjacent edge M_i of e_k, and generates a c_k colored point p with the timestamp of M_i assigned as p's coordinate value (see Figure 3.4(b)). This general approach will be extended when we report influenced vertices or preprocess bipartite graphs. We explain these in later sections.

Now, we summarize the components of our general approach as follows. To solve any problem with our model, we need to specify the following components.

1. The set C of colors and the corresponding graph elements (i.e., vertices, edges);
2. The representation of points in $\mathbb{R}^d$, where $d \geqslant 1$;
3. Appropriate colored range searching data structure to answer the queries.

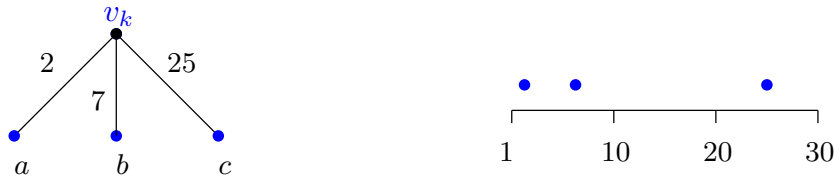
3.4 ALGORITHMS FOR ANSWERING WINDOW QUERIES

3.4.1 *Counting Vertices, Density, and Degrees of Vertices*

Given an RE graph $G = (V, E)$ and a query time slice $[i, j]$, we want to find the number of vertices $|V_{i,j}|$, the density $D(G_{i,j})$, and report all the vertices with non-zero degrees in $V_{i,j}$.

Counting number of vertices $|V_{i,j}|$: We use a set $C = \{c_1, c_2, \cdots, c_n\}$ of n colors, where each c_k is associated with a vertex v_k, for $1 \leqslant k \leqslant n$. For each vertex $v_k \in V$, we maintain a linked list $adj[v_k]$, where each node of the list contains its neighboring vertex N_i, where $1 \leqslant i \leqslant d(v_k)$, and the timestamp of the edge between them, i.e., $t(v_k, N_i)$. We associate color c_k with all timestamps stored in $adj[v_k]$, for all $1 \leqslant k \leqslant n$. Now we have exactly $2|E|$ colored points on the real line (see

Figure 3.3 for an example). Let this set of colored points be P. Using 1-dimensional colored range counting data structure on P with query interval $q = [i, j]$, we can find all distinct colors intersected by q. Each of these distinct colors represents a vertex having adjacent edges in $G_{i,j}$. Thus, we can report $|V_{i,j}|$ in $O(\log m) = O(\log n)$ time (by [16], Theorem 3.2) using a data structure of size $O(m \log n)$.



(a) (b)

Figure 3.3: (a) A vertex v_k and its three neighbors a, b and c. Since the timestamp $t(v_k, a)$ of the edge (v_k, a) is 2, (v_k, a) is shown as a c_k-colored point on the line with x-coordinate value = 2. (b) The set of all c_k colored points on the line.

Computing density $D(G_{i,j})$: This step requires computing the value of $|E_{i,j}|$ in addition to $|V_{i,j}|$. Note that G is a simple graph. For any graph slice $G_{i,j}$ of G, the value of $|E_{i,j}|$ can be computed in constant time since $|E_{i,j}| = j - i + 1$. We summarize these results as follows.

THEOREM 3.7. *Given an RE graph* G *with* m *edges and* n *vertices, and a query time interval* $[i, j]$*, the problem of computing* $|V_{i,j}|$ *and the density* $D(G_{i,j})$ *of the graph slice can be reduced to 1-dimensional colored range queries in linear time. Queries can be answered in* $O(\log n)$ *time using* $O(m \log n)$ *space.*

Reporting degrees of vertices $d(v)$: We want to report the degree of each vertex $v_k \in V$ such that v_k has some neighboring edges in $G_{i,j}$. So, we construct a 1-dimensional type-2 color counting data structure on P. This query will report all vertices that have non-zero degrees in $G_{i,j}$ in $O(\log n + w)$ time, where w is the number of reported vertices.

THEOREM 3.8. *Given an RE graph* G *with* m *edges and* n *vertices, and a query time interval* $[i, j]$*, the problem of computing degrees of all vertices* $v_k \in V_{i,j}$ *can be reduced to 1-dimensional type-2 colored range queries in linear time. Queries can be answered in* $O(\log n + w)$ *time, where* w *is the number of reported vertices.*

3.4.2 *Counting* k*-stars and* h*-index*

Given an RE graph G with m edges and n vertices, a fixed threshold k and a query time slice $[i, j]$, we want to count all k-stars in $G_{i,j}$. We use a similar model as described in the previous section (see Figure 3.3). Each edge adjacent to a vertex v_k will have the color c_k. Thus we again have a set of colored points on the real line. By applying k-threshold color queries (Theorem 3.6) with query interval $q = [i, j]$ we can report all vertices that have at least k neighbors in $G_{i,j}$, hence are at the center of k-stars in $G_{i,j}$.

THEOREM 3.9. *Given an RE graph* G *with* m *edges and* n *vertices, a query time slice* $[i, j]$*, and a fixed threshold value* k*, the problem of finding all* k*-stars in* $G_{i,j}$ *can be reduced to* k*-threshold color queries in linear time. Queries can be answered in* $O(\log n + w)$ *time, where* w *is the number of* k*-stars.*

Counting h*-index.* Given an RE graph G with m edges and n vertices, and a query time slice $[i, j]$, we want to compute the h-index of $(G_{i,j})$. To answer this query we use a similar coloring model that we described for counting k-stars and build the parameterized k-threshold color counting data structures (see Theorem 3.6). A k-threshold color counting data structure reports the number of vertices (w) that have at least k neighbors in $G_{i,j}$. Therefore, if the number of reported vertices $w \geqslant k$, then the h-index of $G_{i,j}$ is at least k. Now we perform a set of decision problems to compute the h-index$(G_{i,j})$, i.e., *'Are there at least* k *vertices of degree* k *in* $G_{i,j}$*?'*

We query the k-threshold color counting data structure with parameter $k = 1$ and compare the number of colors (i.e., w) reported by this query with the value of k. We execute Algorithm 6 using $O(\log n)$ values of k, i.e., $k = 1, (1+\epsilon), (1+\epsilon)^2, \ldots, \log_{1+\epsilon} n$, where ϵ is an arbitrarily small positive constant. By Theorem 3.6 the total reduction time becomes $O(m \log^2 n)$ using $O(m \log n)$ space.

Algorithm 6: Compute h-index

Input : Set S of n colored points, a small positive number ϵ.

1 Set $k = 1$.
2 Execute k-threshold colored query and return w.
3 **while** $w > k$ **do**
4 Set $k \leftarrow k(1+\epsilon)$.
5 Execute k-threshold colored query and return w.
6 **if** $w = k$ **then**
7 $z = k$.
8 **else**
9 $z = k/(1+\epsilon)$
10 Return z.

CLAIM 1. *Let* h *be the* h*-index of* $G_{i,j}$ *and* ℓ *be a positive number, with* $1 \leqslant \ell \leqslant \log_{1+\epsilon} n$*, such that* $(1+\epsilon)^\ell < h \leqslant (1+\epsilon)^{\ell+1}$*. Let* z *be the value returned by Algorithm 6. Then* $(1+\epsilon)^\ell < z \leqslant (1+\epsilon)^{\ell+1}$*.*

Proof. When $z = k = (1+\epsilon)^\ell$, with $1 \leqslant \ell \leqslant \log_{1+\epsilon} n$, the algorithm returns the actual value of h (see lines 6-7). When $w < k$ $(= (1+\epsilon)^{\ell+1})$ the algorithm returns $k/(1+\epsilon)$ (see lines 8-9), which is in the range of $(1+\epsilon)$ from the actual value of h. Note that both h and z lie in the same range of values. Therefore the claim holds. □

THEOREM 3.10. *Given an RE graph* G *with* m *edges and* n *vertices, and a query time slice* $[i, j]$*, the problem of computing the* h*-index of* $G_{i,j}$ *can be reduced to the parameterized* k*-threshold color counting problem for* $O(\log n)$ *values of* k *using* $O(m \log n)$ *space. Queries can be answered in* $O(\log^2 n + h \log n)$ *time with an approximation ratio of* $(1+\epsilon)$*, where* ϵ *is a small positive constant and* h *is the* h*-index of* $G_{i,j}$*.*

3.4.3 *Computing Neighborhood Overlap and Embeddedness*

We notice, from the definition of neighborhood overlap of an edge (u, v) that the numerator term represents the embeddedness of the edge (u, v) (i.e., the number of triangles containing the edge (u, v)) and the denominator is the number of edges adjacent to edge (u, v) minus the common edges (triangles). So, to answer the main query we need to solve two subproblems; i.e., for each edge $e_k \in G_{i,j}$, where $i \leqslant k \leqslant j$, we count (a) the number of edges adjacent to e_k, and (b) the number of triangles containing e_k. We describe below the preprocessing steps for these subproblems.

Counting the number of adjacent edges: We use a set of m colors, $C = \{c_1, c_2, \cdots, c_m\}$, where each color c_k will be associated with an edge e_k, for $1 \leqslant k \leqslant m$. Now, for each edge e_k we associate each adjacent edge of e_k with a color $c_k \in C$. Suppose, e_k has a set of adjacent edges $A(e_k) = \{e', e'', \cdots\}$. We create a c_k-colored point in $\mathbb{R}^2$ for each edge $e' \in A(e_k)$ with coordinate values $(t(e_k), t(e'))$ (see Figure 3.4). The total number of colored points is $\sum_{e=(u,v)}(d(u) + d(v)) = \sum_u d(u)^2 \leqslant (n-1)\sum_u d(u) = 2m(n-1) = O(mn)$. Using 2-dimensional type-2 counting query data structure [14, Theorem 6.2], with query rectangle $[i, j] \times [i, j]$, the number of edges adjacent to each $e_k \in E_{i,j}$ can be found in $O(\log^2 mn + s \log mn) = O(\log^2 n + s \log n)$ time using $O(mn \log n)$ space, where s is the total number of edges in $E_{i,j}$ having some neighboring edges.

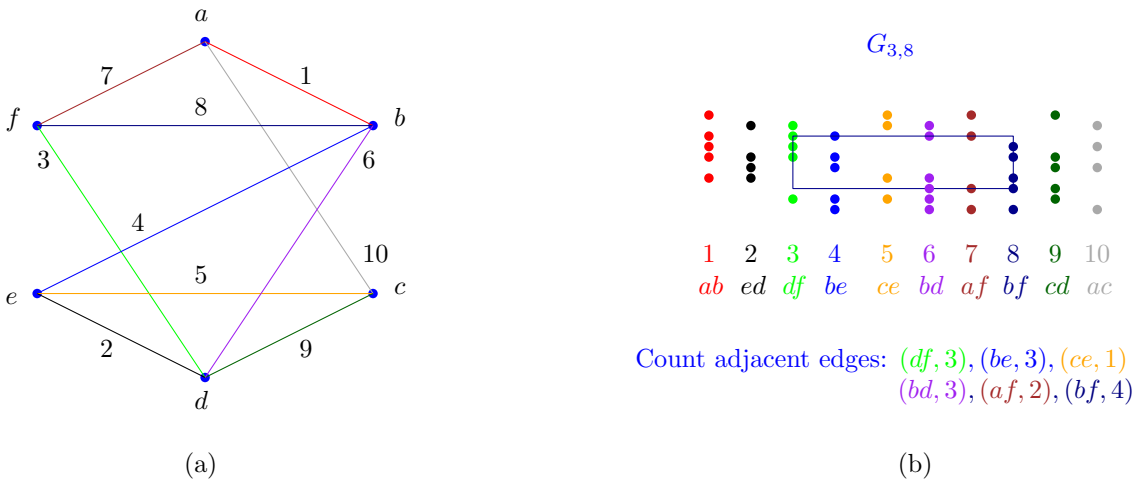


Figure 3.4: (a) Relational event graph G, and (b) Number of adjacent edges in the graph slice $G_{3,8}$.

Counting the number of triangles: We observe that G can be preprocessed in $O(\alpha(G)m)$ time to count the number of triangles in $G_{i,j}$ ([2], Theorem 2), where $\alpha(G)$ is the arboricity of G. Arboricity is defined as the minimum number of edge-disjoint spanning forests into which G can be partitioned [18]. For a general connected graph G, $\alpha(G) = O(\sqrt{m})$ [19]. Algorithm-1 presented in [2] represents each triangle of G as a point $p = (high, low) \in \mathbb{R}^2$, where $high$ (low) is the maximum (minimum) timestamp of the participating edges of that triangle. Now we modify the algorithm in [2] so that each point p in the original algorithm representing a triangle will now have three copies of itself and each copy will be associated with the color of each participating

edge of the triangle (see Figure 3.5). There will be exactly $3\mathcal{K}$ colored points in $\mathbb{R}^2$, where $\mathcal{K}$ is the number of triangles in G. Using 2-dimensional type-2 range counting query data structure [14, Theorem 6.2], with query rectangle $[i, j] \times [i, j]$, the number of triangles containing edges e_k, where $(i \leqslant k \leqslant j)$, can be found in $O(\log^2 \mathcal{K} + t \log \mathcal{K})$ time using $O(\mathcal{K} \log \mathcal{K})$ space, where t is the number of output edges. The maximum number of triangles in any graph G can be $O(n^3)$. So the query time can be re-defined as $O(\log^2 n + t \log n)$ and the space required is $O(n^3 \log n)$. From this result, we can also report (i) total number of local bridges $\#LB(G_{i,j}) = (j - i - t + 1)$ and (ii) value of embeddedness of any edge in $G_{i,j}$.

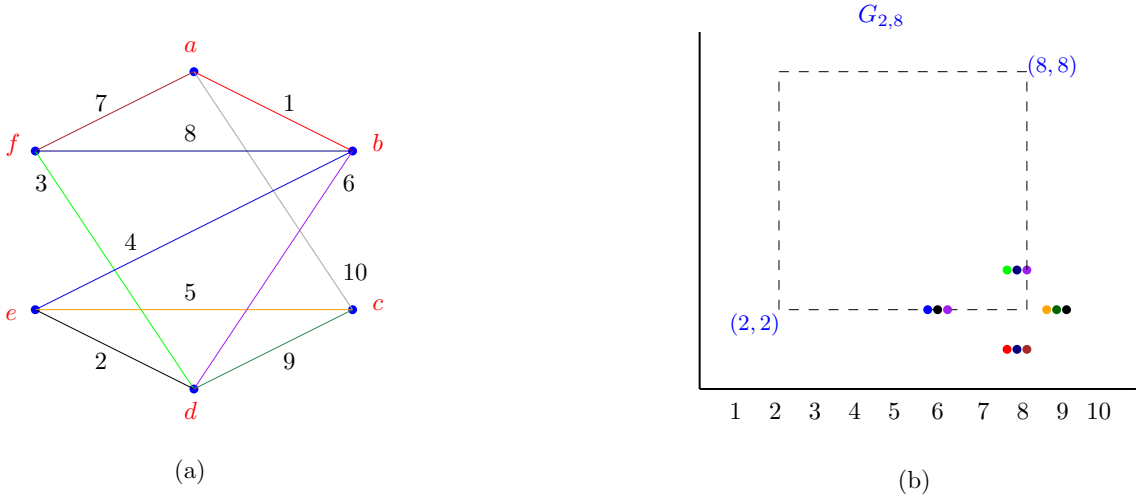


Figure 3.5: (a) Relational event graph G, and (b) Number of triangles adjacent to edges in the graph slice $G_{2,8}$.

Analysis: For part (a), there are overall $O(mn)$ points representing all edge adjacencies in G. For part (b), identifying all triangles in G requires $O(a(G)m)$ preprocessing time. Thus, the total time required to reduce the problem of computing neighborhood overlap to colored range queries takes $O(a(G)m \log n + mn \log n) = O(mn \log n)$ time. Total query time is $O(\log^2 n + t \log n) + O(\log^2 n + s \log n) = O(\log^2 n + (t + s) \log n)$. The following theorem summarizes the results.

THEOREM 3.11. *Given an RE graph* G *with* m *edges and* n *vertices, and a query time slice* $[i, j]$*, the problem of computing the average neighborhood overlap of* $G_{i,j}$ *can be reduced to the colored range counting in* $O(mn \log n)$ *time.* $\mathsf{NOver}(G_{i,j})$ *can be computed in* $O(\log^2 n + (t + s) \log n)$ *time, where* t *is the number of edges in* $G_{i,j}$ *with positive embeddedness and* s *is the number of edges having some neighboring edges in* $E_{i,j}$.

3.4.4 *Neighborhood Overlap in Bipartite Graphs*

Given a bipartite RE graph $G = (V = \{A \cup B\}, E)$ with m edges and n vertices, and a query time interval $[i, j]$, we now want to compute the neighborhood overlap of $G_{i,j}$. Recall from Definition 3.2 that for every pair of vertices $(u_1, u_2) \in A$, we need to compute (a) total number of common neighbors and (b) the total number of neighbors of the vertex pair.

Counting common neighbors of vertex pairs: To count all common neighbors of a pair of vertices $(u_1, u_2) \in A$ of a bipartite graph G, we use the edge searching technique applied in the quadrangle counting algorithm in [2]. The original algorithm represents a set of quadrangles sharing a common pair of vertices (two opposite corners of a rectangle) (u_1, u_2) as a tuple $(u_1, u_2, X = \{v_1, v_2, \cdots\})$ such that every pair of elements in X is connected to u_1 and u_2. In our case, for every tuple $u_1, u_2 \in A$, $X = \{v_1, v_2, \cdots\} \in B$, and we want to count $|X|$. Therefore, we modify the original algorithm presented in [2] such that the following holds.

1. The algorithm always starts searching with the vertices of set A.
2. Once a tuple $(u_1, u_2, X = \{v_1, v_2, \cdots\})$ is identified, (a) the pair of vertices (u_1, u_2) will correspond to a color $c_{u_1u_2}$, which is then added to the set of colors C, and (b) for each element $v_i \in X$ we store a point $(t(u_1, v_i), t(u_2, v_i)) \in R^2$ of color $c_{u_1u_2}$, where $t(u_1, v_i)$ is the timestamp of the edge (u_1, v_i).

Analysis: The modified algorithm identifies all neighbors of all pairs of vertices of A in $O(\alpha(G)m)$ time [2]. For a complete bipartite RE graph G, $|C|$ will be at most $O(n^2)$, and there can be at most $O(n^3)$ colored points identified using this algorithm. Now given a query time slice $[i, j]$, we can find the total number of neighbors of each pair of vertices in the bipartite graph slice $G_{i,j}$ using 2-dimensional type-2 color counting data structure [14, Theorem 6.2]. This query can be answered in $O(\log^2 n + p \log n)$ time, where p is the total number of vertex pairs having some common neighbors.

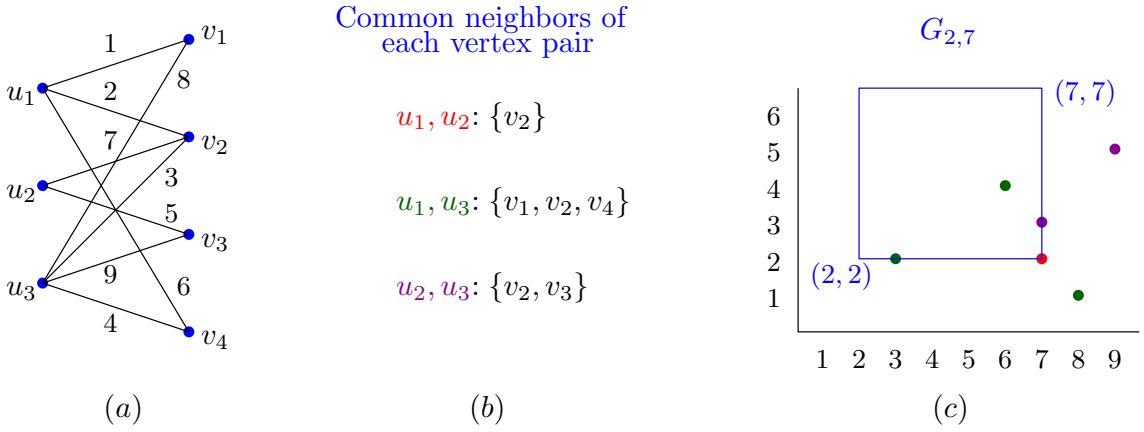


Figure 3.6: (a) A bipartite RE graph (b) Set of neighbors adjacent to each vertex pairs, (c) Common neighbors for vertex pairs within query interval $q = [2, 7]$.

Counting all neighbors of vertex pairs: To count the total number of neighbors of each vertex pair, we count total degrees of all vertices $u_i \in A$ using 1-dimensional Type-2 colored range counting data structure in $O(\log n + k)$ time, where k is the total number of vertices having some neighbors in $G_{i,j}$ (see Theorem 3.8, Section 3.4.1). Thus the average neighborhood overlap of a bipartite graph slice $G_{i,j}$ can be computed in $O(\log^2 n + p \log n + k)$ time. The preprocessing time for the whole process is dominated by the process of counting common neighbors of all vertex pairs. Hence, total time required for preprocessing is $O(\alpha(G)m)$. Thus we have the following results for bipartite RE graphs.

THEOREM 3.12. *Given a bipartite RE graph* G *with* m *edges and* n *vertices, and a query time slice* $[i, j]$*, the problem of computing the average neighborhood overlap of a bipartite graph slice* $G_{i,j}$ *can be reduced to type-2 colored range searching problem in* $O(\alpha(G)m)$ *time.* $\mathsf{NOverB}(G_{i,j})$ *can be computed in* $O(\log^2 n + p \log n + k)$ *time, where* p *is the total number of vertex pairs having some common neighbors in* $G_{i,j}$ *and* k *is the total number of vertices having some neighbors in* $G_{i,j}$*.*

3.4.5 *Reporting Influenced Vertices*

Given an RE graph $G = (V, E)$ with a fixed set of influential vertices $V' \subseteq V$, a positive integer r, and a query time slice $[i, j]$, we want to find the total number of influenced vertices in $G_{i,j}$. We associate each vertex $v_k \in V$ with a color $c_k \in C = \{c_1, c_2, \cdots, c_n\}$. Each point representing an influenced vertex v_k will be colored with c_k.

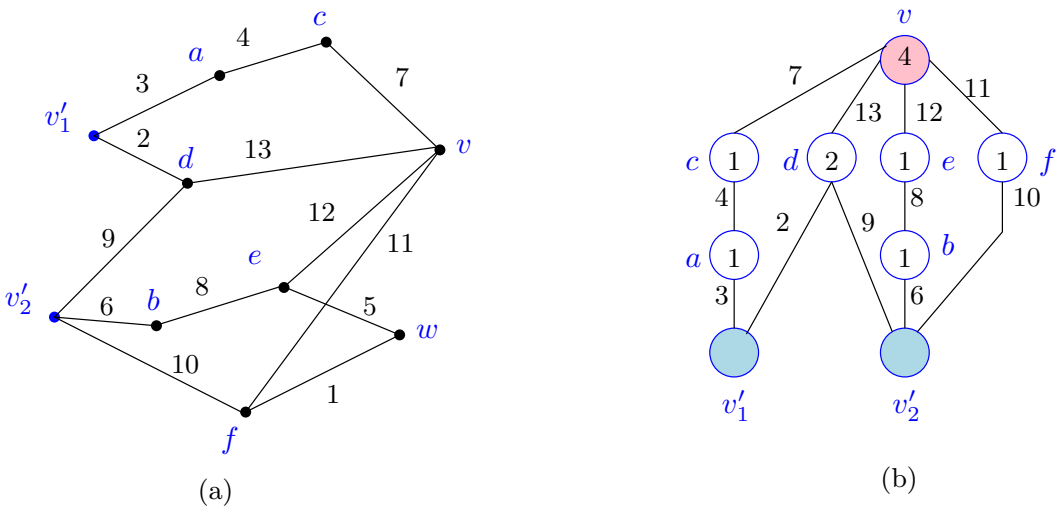


Figure 3.7: (a) An RE graph with two influential vertices v'_1 and v'_2, (b) An influenced vertex v with $r = 3$ and $f(v) = 4$. The influence threshold $f(v)$ is mentioned inside the circle of each vertex v.

Preprocessing Step: Recall Definitions 3.1 and 3.2. For each vertex $v \in V$ having thresholds $f(v) \geqslant 1$, we associate it with a queue $Q[v]$ of size $f(v)$, initially empty. Queues are maintained by balanced binary search trees, so that the minimum element of the queue can be retrieved when required. In addition, we maintain $\alpha(v)$, for all $v \in V$, where $\alpha(v)$ is the largest timestamp of an edge e such that e is adjacent to an influential vertex v' and there is a path of influence from v' to v. Initially $\alpha(v)$ is set to zero, for all $v \in V$. We process each edge $e_k = (u_k, v_k)$ according to the sequence of their timestamps, i.e., $t(e_1) < t(e_2) < \cdots < t(e_m)$, and set $\alpha(u_k)$ and $\alpha(v_k)$. We explain the process as follows. At time k, let $e_k = (u_k, v_k)$.
Case 1: Both u_k and v_k are influential vertices. Then u_k and v_k are already influenced. We set both $\alpha(u_k)$ and $\alpha(v_k)$ to k. We also store two points (k, k) of colors c_{u_k} and c_{v_k} respectively.
Case 2: u_k is an influential vertex and v_k is a non-influential vertex. We store a point (k, k) of color c_{u_k} and set $\alpha(v_k)$ to k.

Case 3: u_k is already an influenced non-influential vertex and $\alpha(u_k) = \ell$ has been set. If $k - \ell \leqslant r$, u_ℓ influences v_k. If $\alpha(v_k) < \alpha(u_k)$ and we set $\alpha(v_k) = \alpha(u_k) = \ell$. Otherwise, we keep $\alpha(v_k)$ unchanged.

Each time we set $\alpha(v_k) = \ell$ for some vertex v_k, we add ℓ to $Q[v_k]$. In case $Q[v_k]$ is full after inserting ℓ, we remove the smallest element s from the queue, and store (k, s) as a point in $\mathbb{R}^2$ with color c_{v_k} (see Algorithm 7 for details). The c_{v_k}-colored point (k, s) implies that v_k is influenced by $f(v_k)$ paths of influence in the graph slice $G_{s,k}$. Vertices v_s and v_ℓ are respectively the first and the last influential vertex that influence v_k in $G_{s,k}$. See Figure 3.7 for an example.

Algorithm 7: Influence(G, r)

Input : An RE graph G, a positive integer r.
Output: A colored point set $P \in R^2$.

```
1  for each vertex v_k ∈ V for k = 1 to n do
2      Set α(v_k) ← 0.
3      if f(v_k) > 0 then
4          Set queue Q[v_k] ← ∅ with size f(v_k).
5  for each edge e_k = (u_k, v_k) ∈ E for k = 1 to m do
6      if u_k is an influential vertex then
7          Set α(u_k) ← k and α[v_k] ← k.
8          Store a point (k, k) ∈ R^2 of color c_{u_k}.
9          if v_k is a non-influential vertex then
10             if f(v_k) = 1 then
11                 Store a point (k, k) ∈ R^2 of color c_{v_k}.
12             else
13                 Add α(u_k) to Q[v_k]
14             if the queue Q[v_k] is full then
15                 Remove the minimum element s from Q[v_k].
16                 Store a point (k, s) ∈ R^2 of color c_{v_k}.
17     if u_k is an influenced vertex then
18         if α(u_k)(= ℓ) > α(v_k) and (k − ℓ) ⩽ r then
19             Set α[v_k] ← ℓ.
20             Add ℓ to Q[v_k].
21             if the queue Q[v_k] is full then
22                 Remove the minimum element s from Q[v_k].
23                 Store a point (k, s) ∈ R^2 of color c_{v_k}.
```

Preprocessing Analysis: All initializations take linear time (lines 1-4, Algorithm 7). There will be n queues, one for each vertex and each queue can have size at most equal to the degree of the vertex $d(v)$. Maintaining a queue for each vertex using balanced binary search tree requires

$O(d(v) \log d(v))$ time, and both insertion and removal of a queue element requires $O(\log d(v))$ time. We visit each edge exactly once to set the values of α (lines 7 and 19), and to perform operations on queues (lines 13 and 20 for insertions, lines 15 and 22 for deletions). Thus, the total preprocessing time will be $\sum_v O(d(v) \log d(v)) = O(m \log n)$.

Now, we can reduce this problem to the colored range searching problem. In the worst case, every vertex can be influenced by each of its neighboring vertices. Thus, we have a set of at most $O(n^2)$ colored points in plane. Hence our problem of reporting all influenced vertices within $G_{i,j}$ reduces to the problem of reporting the number of distinct colors in plane. By using 2-dimensional range reporting data structure with query rectangle $[0, j] \times [i, \infty]$, we can report all influenced vertices in $G_{i,j}$ in $O(\log n + w)$ time using $O(n^2)$ space, where w is the number of influenced vertices ([15], Theorem 1.12).

THEOREM 3.13. *Given an RE graph G with m edges and n vertices, and a query time slice $[i, j]$, the problem of reporting the total number of influenced vertices can be reduced to 2-dimensional colored range reporting problem in $O(m \log n)$ time using $O(n^2)$ space. Queries can be answered in $O(\log n + w)$ time, where w is the number of influenced vertices in $G_{i,j}$.*

3.5 CONCLUSION

In this paper, we have presented a general approach for solving queries regarding various structural parameters of relational event graphs in an arbitrary query time slice using colored range query data structures. Our approach models the reachability relationships between vertices and edges of a given RE graph by transforming them into colored points in $\mathbb{R}^d$, where $d \geqslant 1$. Subsequently, we reduce original problems into colored range searching problems to efficiently answer the queries. Following our model, we showed (a) how to compute the value of a specific structural parameter of any graph slice $G_{i,j}$; e.g., density, embeddedness, neighborhood overlap, h-index; and (b) how to count and report vertices in any $G_{i,j}$ with some specific properties; e.g., being influenced vertices or k-stars. We also presented a new persistence based k-threshold colored range searching data structure, where k is a fixed positive integer.

We leave the following open problems:

OPEN PROBLEM 3. *The problem of k-threshold colored range searching data structure is still open if the value of k is given during the query time.*

OPEN PROBLEM 4. *Given a set of points in $\mathbb{R}^d$, with $d \geqslant 2$, and a fixed integer $k > 0$ construct the k-threshold colored range searching data structure.*

OPEN PROBLEM 5. *The problem of reporting the degree of any vertex v in the query window can be answered in $O(1)$ time using an $O(m)$-bit binary* rank and select *data structure [20], where for $k = 1, \ldots, m$, the kth bit denotes whether the kth edge is adjacent to v. Hence the complete data structure requires $O(mn)$ bits. It will be interesting to further investigate whether space can be improved preserving the query time and whether this technique can be used for other window problems and obtain better bounds.*

BIBLIOGRAPHY

[1] 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

[2] F. Chanchary, A. Maheshwari, Counting subgraphs in relational event graphs, in: WALCOM: Algorithms and Computation - 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings, Vol. 9627 of Lecture Notes in Computer Science, Springer, 2016, pp. 194–206. doi:10.1007/978-3-319-30139-6_16.
URL https://doi.org/10.1007/978-3-319-30139-6_16

[3] Chanchary, F., Maheshwari, A., Smid, M.: Querying relational event graphs using colored range searching data structures. Accepted for publication in Journal of Discrete Applied Mathematics (2019)

[4] Chanchary, F., Maheshwari, A., Smid, M.H.M.: Querying relational event graphs using colored range searching data structures. In: Algorithms and Discrete Applied Mathematics - Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10156, pp. 83–95. Springer (2017), https://doi.org/10.1007/978-3-319-53007-9_8

[5] N. Santoro, W. Quattrociocchi, P. Flocchini, A. Casteigts, F. Amblard, Time-varying graphs and social network analysis: Temporal indicators and metrics, Vol. abs/1102.0629, 2011. arXiv:1102.0629.
URL http://arxiv.org/abs/1102.0629

[6] M. Berlingerio, M. Coscia, F. Giannotti, A. Monreale, D. Pedreschi, The pursuit of hubbiness: Analysis of hubs in large multidimensional networks, J. Comput. Science 2 (3) (2011) 223–237. doi:10.1016/j.jocs.2011.05.009.
URL https://doi.org/10.1016/j.jocs.2011.05.009

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

[8] D. A. Easley, J. M. Kleinberg, Networks, Crowds, and Markets - Reasoning About a Highly Connected World, Cambridge University Press, 2010.
URL http://www.cambridge.org/gb/knowledge/isbn/item2705443/?site_locale=en_GB

[9] N. Meghanathan, A greedy algorithm for neighborhood overlap-based community detection, Algorithms 9 (1) (2016) 8. doi:10.3390/a9010008.
URL https://doi.org/10.3390/a9010008

[10] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985. URL: https://doi.org/10.1137/0214021, doi:10.1137/0214021.

[11] L. M. Bettencourt, A. Cintrón-Arias, D. I. Kaiser, C. Castillo-Chávez, The power of a good idea: Quantitative modeling of the spread of ideas from epidemiological models, Physica A: Statistical Mechanics and its Applications 364 (2006) 513–536.

[12] D. Centola, M. Macy, Complex contagions and the weakness of long ties, American Journal of Sociology 113 (3) (2007) 702–734.

[13] L. Gargano, P. Hell, J. G. Peters, U. Vaccaro, Influence diffusion in social networks under time window constraints, Theor. Comput. Sci. 584 (2015) 53–66. doi:10.1016/j.tcs.2015.02.015.
URL https://doi.org/10.1016/j.tcs.2015.02.015

[14] P. Bozanis, N. Kitsios, C. Makris, A. K. Tsakalidis, New upper bounds for generalized intersection searching problems, in: Automata, Languages and Programming, 22nd International Colloquium, ICALP95, Szeged, Hungary, July 10-14, 1995, Proceedings, 1995, pp. 464–474. doi:10.1007/3-540-60084-1_97.
URL https://doi.org/10.1007/3-540-60084-1_97

[15] P. Gupta, R. Janardan, S. Rahul, M. Smid, Computational geometry: Generalized (or colored) intersection searching, Handbook of Data Structures and Applications, 2nd Edition, (Dinesh Mehta and Sartaj Sahni, editors), CRC Press, Boca Raton, Chapter 67 (2018) 1042–1057.

[16] P. Gupta, R. Janardan, M. H. M. Smid, Further results on generalized intersection searching problems: Counting, reporting, and dynamization, J. Algorithms 19 (2) (1995) 282–317. doi:10.1006/jagm.1995.1038.
URL https://doi.org/10.1006/jagm.1995.1038

[17] R. Janardan, M. A. López, Generalized intersection searching problems, Int. J. Comput. Geometry Appl. 3 (1) (1993) 39–69. doi:10.1142/S021819599300004X.
URL https://doi.org/10.1142/S021819599300004X

[18] F. Harary, Graph theory, Addison-Wesley, 1991.

[19] N. Chiba, T. Nishizeki, Arboricity and subgraph listing algorithms, SIAM J. Comput. 14 (1) (1985) 210–223. doi:10.1137/0214017.
URL https://doi.org/10.1137/0214017

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

4

INTERSECTING OBJECTS, MAXIMAL POINTS AND APPROXIMATIONS USING CORESETS

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 $\mathbb{R}^d$, $d \geqslant 2$. Finally we present techniques to approximate window-aggregate queries for $(1+\epsilon)$-approximations for various geometric problems such as diameter, width, radius of a minimum enclosing ball, volume of the smallest bounding box, and the cost of ℓ-center clustering ($\ell \geqslant 2$) for a set of points using coresets. All data structures presented in this paper answer queries in polylogarithmic time and use subquadratic space.

This chapter is a combination of results that have been presented in the 4th international conference on Algorithms and Discrete Applied Mathematics (CALDAM 2018) [17] and results that have been submitted in the Journal of Discrete Applied Mathematics [16].

4.1 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 $1 \leqslant i < j \leqslant n$, let $S_{i,j}$ denote the subsequence $(s_i, s_{i+1}, \ldots, s_j)$. We want to preprocess S into 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 (for example, see [6, 7, 9, 13, 14, 15]), where the authors mapped a sequence of geometric objects (or graph edges) to a sequence of *timestamped events*, where for each k, with $1 \leqslant k \leqslant n$, 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 window reporting problems using points on maximal layers. We define the *windowed intersection decision problem* as 'Given a pair of indices (i, j), where $1 \leqslant i < j \leqslant n$, report whether there is any intersection between objects in $(s_i, \ldots, s_j)$'. In [13], 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 $d \geqslant 2$, and we answer queries related to maximal layers and dominance. More specifically we solve three types of window 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 $i \leqslant k \leqslant j$, decide if point p_k is on the maximal layer L_γ of $P_{i,j}$, where $\gamma = 1$, or 2, or $\geqslant 3$, and (iii) for a fixed integer $k \geqslant 1$, (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 $f(\epsilon)$ 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 the cost of the ℓ-center clustering of a queried point set $P_{i,j}$.

4.1.1 *Previous Work*

Bannister et al. [7] were the first to consider this window model for preprocessing timestamped graph edges into data structures that can answer window queries. Subsequently more results on window graph problems were presented in [14] and [15]. Similar time window model for geometric objects was first studied in [6], where the authors presented results for reporting the convex hull of points in the plane, and skyline and proximity relations of point sets in $\mathbb{R}^d$, with $d \geqslant 2$. They used a hierarchical decomposition in time to construct binary decomposition trees on a given set of temporal points to answer window 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. [27] considered problems of monitoring top-k maximal layers (mentioned as *k-skyband* in [27]) using fixed width sliding query windows. Our results for points on maximal layers presented in this paper are different from those of [6] and [27].

Subsequently more results have been presented by Bokal et al. [9], and Chan and Pratt [13] on window queries. In [9] and [13], the authors 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. In [9], Bokal et al. 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 $1 \leqslant i \leqslant n$, 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 [13] 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 window 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 as *FIFO updates* and used this technique to solve the 2D convex hull area decision problem and the 2D width decision problem. Table 4.1 summarizes previous results of window queries for geometric problems.

Table 4.1: 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	Preprocessing time	Query time	Reference
Convex hull reporting	$O(n \log n)$	$O(h \log^2 \mathcal{W})$	[6]
Gift wrapping, line stabbing, tangent queries	$O(n \log n)$	$O(\log^2 \mathcal{W})$	[6]
Linear prog., line decision queries	$O(n \log n)$	$O(\log \mathcal{W})$	[6]
Skyline reporting	$O(n^{1+\epsilon})$	$O(w)$	[6]
Spherical range reporting	$O(n \log n)$	$O(\log \mathcal{W} + w)$	[6]
Approx. nearest neighbor	$O(n \log n)$	$O(\log \mathcal{W})$	[6]
Diameter decision problem (2D)	$O(n \log^2 n)$	$O(1)$	[9]
	$O(n \log n)$	$O(1)$	[13]
Diameter decision problem (3D)	$O(n \log^2 n)$	$O(1)$	[13]
Convex hull area decision problems	$O(n \log n \log\log n)$	$O(1)$	[9]
	$O(n \alpha(n) \log n)$	$O(1)$	[13]
Monotone paths	$O(n)$	$O(1)$	[9]
Orthogonal segment intersection detection	$O(n \log n \log\log n)$	$O(1)$	[13]
Width decision problem (2D)	$O(n \log^8 n)$	$O(1)$	[13]

4.1.2 *New Results*

The main contributions of this paper are listed below, and are also summarized in Table 4.2.

1. *Intersection decision problems*: Given a sequence S of n geometric objects we can preprocess S for the windowed intersection decision problem in $O(n^{4/3} \cdot \mathrm{polylog}(n))$ time using $O(n)$ 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 $i \leqslant k \leqslant j$, 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 $i \leqslant k \leqslant j$, we can report whether p_k is on layer 2 or $\geqslant 3$ of the sequence $P_{i,j}$ 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 $d \geqslant 4$. 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 $2 \leqslant d \leqslant 3$. For $d \geqslant 4$, 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 $2 \leqslant d \leqslant 5$. For $d \geqslant 6$, 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. P can be preprocessed into a data structure of size $O(n\log n)$ such that for a query interval $[i,j]$, with $1 \leqslant i < j \leqslant n$ and a parameter $\epsilon > 0$, we can compute a $(1+\epsilon)$-coreset of size $O(f(\epsilon))$ in $O(f(\epsilon)\log n)$ time for geometric problems that admit some decomposable coresets.
   - Let P be a sequence of n points in $\mathbb{R}^2$. 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 $\ell \geqslant 2$ and $\epsilon > 0$, a coreset of size $O(\ell(f(\ell)/(c\epsilon))^2)$ for the ℓ-center clustering can be computed in $O(\ell((f(\ell)/(c\epsilon))\log n) + \ell(f(\ell)/(c\epsilon))^2 + W)$ time, where W is the width of the query window and c is a positive constant.

4.1.3 *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 window 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.

4.2 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 $1 \leqslant i < j \leqslant n$, report whether there is any intersection between objects in $(s_i, \ldots, s_j)$'.

4.2.1 *Preliminaries*

To solve windowed intersection decision problem of geometric objects given in $\mathbb{R}^d$, we use the following structures described in [12].

COROLLARY 4.1. *[12, 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*

Table 4.2: Summary of results. n is the number of input objects, $\gamma \geqslant 2$ is the number of maximal layer, w is the output size, k is a fixed parameter, $W = j - i + 1$ is the width of the query window, $\ell \geqslant 2$ is an input parameter, $\epsilon > 0$ and $c > 0$ are small constants.

Problems	Preprocessing	Space	Query	
Intersection Decision				
Segment, Bichromatic segment, Triangle, c-gon (c is constant)	$O(n^{4/3}\mathrm{polylog}(n))$	$O(n)$	$O(\log n)$	Th. 4.6
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. 4.29
p_k on max layer $L_\gamma, \gamma = 2, \geqslant 3$	$O(n \log^{d+1} n)$	$O(n \log^{d+1} n)$	$O(\log^{d+1} n)$	Th. 4.33
Count maximal points: $d = 2, 3$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(\log^2 n)$	Th. 4.30
Count maximal points: $d \geqslant 4$	$O(n \log^{d-1} n)$	$O(n \log^{d-2} n)$	$O(\log^2 n)$	Th. 4.30
k-dominated points: $2 \leqslant d \leqslant 3$	$O(kn \log^2 n)$	$O(kn \log n)$	$O(\log^2 n + kw)$	Th. 4.31
k-dominated points: $d \geqslant 4$	$O(kn \log^{d-1} n)$	$O(kn \log^{d-2} n)$	$O(\log^2 n + kw)$	Th. 4.31
k-dominant points: $2 \leqslant d \leqslant 5$	$O(kn \log^4 n)$	$O(kn \log^3 n)$	$O(\log^4 n + kw)$	Th. 4.32
k-dominant points: $d \geqslant 6$	$O(kn \log^{d-1} n)$	$O(kn \log^{d-2} n)$	$O(\log^4 n + kw)$	Th. 4.32
Approximation using Coresets				
Decomposable coresets	$O(n \log n)$	$O(n \log n)$	$O(f(\epsilon) \log n)$	Th. 4.38
ℓ-clustering using coresets	$O(n \log n)$	$O(n \log n)$	$O(\ell((f(\ell)/(c\epsilon)) \log n) + \ell(f(\ell)/(c\epsilon))^2 + W)$	Th. 4.41

be reported as a union of disjoint canonical subsets C_i with $\sum_i |C_i|^{1-1/d} \leqslant O(n^{1-1/d} \log n)$ in time $O(n^{1-1/d} \log n)$ with high probability with respect to n.

COROLLARY 4.2. *[12, 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 4.3. *[12, 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 4.4. *[12, 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 $m \in [n \log n, n^d/\log^d n]$.*

4.2.2 *Overview of Our Data Structure*

Before we discuss our data structure, we define a *valid pair* of indices (α, β) with $1 \leqslant \alpha < \beta \leqslant n$ as follows: For each $1 \leqslant \alpha \leqslant n$, 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 $\beta = +\infty$. See Figure 4.1 for an illustration.

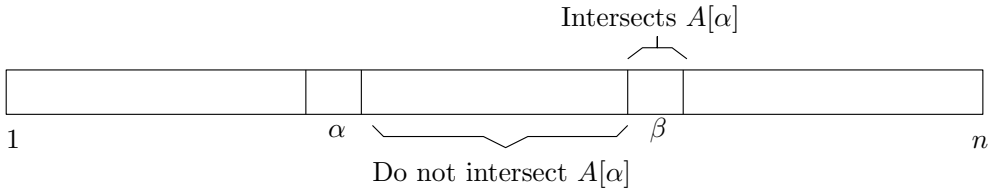


Figure 4.1: A valid pair (α, β).

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 [24]. 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 4.5. *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 $1 \leqslant \alpha \leqslant n$ 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. Suppose X is a set of n geometric objects and we assume that we have a data structure $DS(X)$ that can find 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

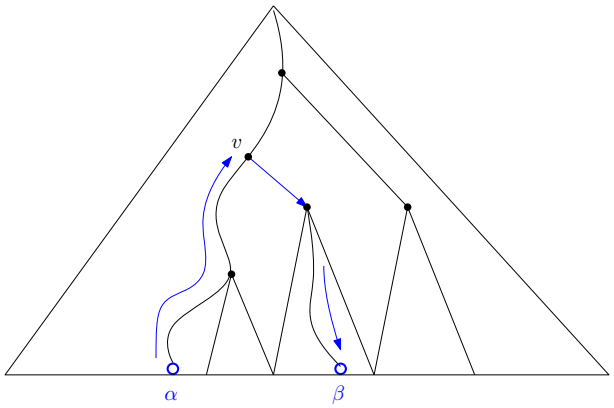


Figure 4.2: Query path from α to β in our multilevel data structure.

$$\begin{aligned} O(n) + \sum_{i=0}^{\log n} 2^i \cdot M(n/2^i) &= O(n) + \sum_{i=0}^{\log n} n \cdot \frac{M(n/2^i)}{n/2^i} \\ &\leqslant O(n) + \sum_{i=0}^{\log n} n \cdot \frac{M(n)}{n} \\ &\leqslant O(n) + \sum_{i=0}^{\log n} M(n) \\ &= O(M(n) \cdot \log(n)). \end{aligned}$$

The total preprocessing time can analogously be computed as $O(P(n) \log n)$. Now for any $1 \leqslant \alpha \leqslant n$, 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 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 $p(v')$ to locate the leaf node containing β (see Figure 4.2). In this way, the total time required to find all valid pairs is $O(n \cdot Q(n) \log n)$. From Lemma 4.5 we obtain the following result.

THEOREM 4.6. *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 window 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 the 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* $1 \leqslant a < b \leqslant n$*?'*. For a sequence of n line segments in the plane, we can obtain a data structure of $O(n \log^3 n)$ 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 [12] three times, where $d = 2$. Finally, by repeated applications of Corollary 7.3(i) and Corollary 7.8 in [12] with $d = 2$, we can build a data structure to answer above mentioned queries that 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 Theorem 4.6, 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 4.7. *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 Intersection: 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 during 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 technique remains unchanged.

COROLLARY 4.8. *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 4.3 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.

Case (a): A sequence of triangles is stored and we ask the query: Given a point p, is p contained in some triangle?
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_q, does t_q 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

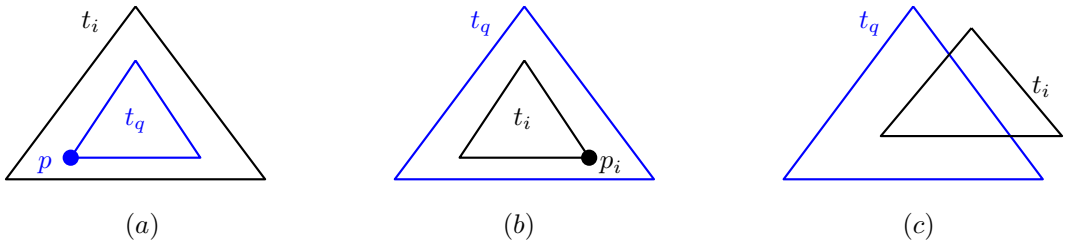


Figure 4.3: Three possible orientations of intersections of a query triangle t_q (blue) with some triangle t_i (black).

t_q, does t_q overlap some triangle t_i?

For cases (a) and (c), by repeated applications of Corollary 7.3(i) and Corollary 7.5 in [12] 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 [12], which also 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 4.9. *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 increase the polylog factors in the preprocessing time. Hence we obtain the following result.

COROLLARY 4.10. *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.*

4.3 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 $d \geqslant 2$, are presented in Section 4.3.6.

DEFINITION 4.11. *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 $p[x] \leqslant p'[x]$ and $p[y] \leqslant p'[y]$, and $p \neq p'$. For $\gamma > 1$, the γ'th maximal layer L_γ is the set of maximal points in $P - \cup_{\ell=1}^{\gamma-1} L_\ell$ [19].*

DEFINITION 4.12. *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., $NE(q) = \{(a, b) \in \mathbb{R}^2 : a > q_1, \text{ and } b > q_2\}$, and $SW(q)$ to be the set of points in $\mathbb{R}^2$ that lie in the South-West quadrant of q, i.e., $SW(q) = \{(c, d) \in \mathbb{R}^2 : c < q_1, \text{ and } d < q_2\}$ (see Figure 4.4).*

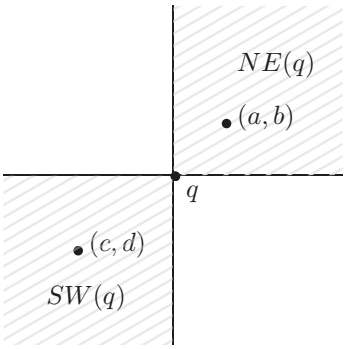


Figure 4.4: A point $(a, b) \in NE(q)$ and a point $(c, d) \in SW(q)$.

4.3.1 *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 $i \leqslant k \leqslant j$ 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 $0 \leqslant i \leqslant n+1$. For any k with $1 \leqslant k \leqslant n$, we define the following two functions: $\alpha(k) = \min\{i : i > k \text{ and } p_i \in NE(p_k)\}$ and $\beta(k) = \max\{i : i < k \text{ and } p_i \in NE(p_k)\}$

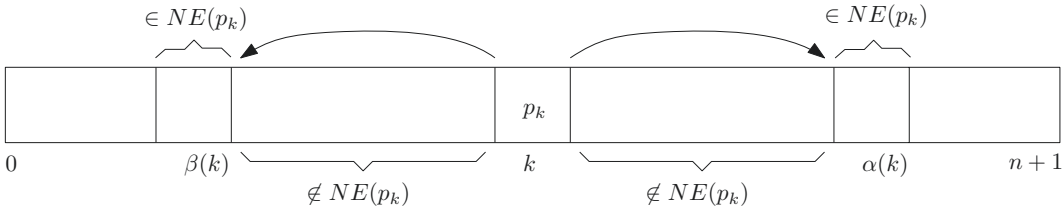


Figure 4.5: $A[\alpha(k)]$ and $A[\beta(k)]$ of a point p_k in array $A[0..n+1]$.

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 in $[i, j]$. We have the following lemma.

LEMMA 4.13. *Suppose $1 \leqslant i \leqslant k \leqslant j \leqslant n$. 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, we have a data structure that computes $\alpha(k)$ and $\beta(k)$ for each $p_k \in P$. 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. According to Lemma 4.13, we can answer the query whether a given point p_k is on the maximal layer of $P_{i,j}$, with $i \leqslant k \leqslant j$, in $O(1)$ time by checking $A[\alpha(k)]$ and $A[\beta(k)]$.

LEMMA 4.14. *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. 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 $i \leqslant k \leqslant j$, we can report whether p_k is on the maximal layer L_1 of $P_{i,j}$ in $O(1)$ time.*

Data structure for computing $\alpha(.)$ ***and*** $\beta(.)$ ***for points in*** $\mathbb{R}^2$**:** Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in plane, we want to build a data structure to compute $\alpha(k)$ and $\beta(k)$ for each p_k, with $1 \leqslant k \leqslant n$. We present the technique for computing $\alpha(k)$ here (see Algorithm 8). We initialize an empty priority search tree (PST) T. For $1 \leqslant i \leqslant n$, we query T with $q = (-\infty, p_{i,x}] \times (-\infty, p_{i,y}]$, and find points that appear before p_i in the sequence and are dominated by p_i. Let this set of points be S. According to the definition of $\alpha(.)$, i becomes the $\alpha(.)$ 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 $1 \leqslant k \leqslant i$: if p_k is in T, then $\alpha(k) \geqslant i$. 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 $1 \leqslant k \leqslant n$. Similarly we can compute $\beta(k)$ for all p_k by reversing their order of insertion to T.

Algorithm 8: SetAlpha(P)

```
Input : A sequence of n points P = (p_1, p_2, ..., p_n) ∈ R^2.
Initialize an empty PST T.
for i = 1 to n do
    Query T with q = (−∞, p_{i,x}] × (−∞, p_{i,y}].
    Let S be the output of this query.
    for each p_k ∈ S do
        Set α(k) = i.
        Delete p_k from T.
    Insert p_i into T.
```

LEMMA 4.15. *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 $\alpha(.)$ ***and*** $\beta(.)$ ***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 2 coordinates of the subset of points stored at v. 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 $d \geqslant 2$, we obtain the following result.

THEOREM 4.16. *Given a sequence of* n *points* $P = (p_1, p_2, \ldots, p_n)$ *in* $\mathbb{R}^d$*, we can compute the values of* $\alpha(k)$ *and* $\beta(k)$ *for all* $p_k \in P$*, in* $O(n \log^{d-1} n)$ *total time using* $O(n \log^{d-2} n)$ *space.*

Remark: Bannister et al. [6] used a dynamic data structure for dominance queries by Mortensen [25] to compute $\alpha(.)$ and $\beta(.)$ values for all points in $\mathbb{R}^d$. Our data structure for computing all $\alpha(.)$ and $\beta(.)$ improves the amount of time by a factor of $O(\log n)$ and the amount of space by a factor of $O(\log^2 n)$.

Finally from Lemmas 4.14 and 4.15 we obtain the following theorem for points in $\mathbb{R}^2$.

THEOREM 4.17. *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* $i \leqslant k \leqslant j$*, we can report whether* p_k *is on the maximal layer* L_1 *of points* $P_{i,j}$ *in* $O(1)$ *time.*

4.3.2 *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 $1 \leqslant i \leqslant j \leqslant n$, 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 4.13, for $1 \leqslant k \leqslant n$, we transform each point $p_k = (p_{k,x}, p_{k,y}) \in P$ into a point $p'_k = (k, \alpha(k), \beta(k)) \in \mathbb{R}^3$. 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 4.15. We build a standard 3-dimensional range tree [8], 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 [8].

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 4.18. *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.*

4.3.3 *Is* p_k *on Maximal Layer* L_γ*, where* $\gamma = 2$ *or* $\gamma \geqslant 3$*?*

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 $1 \leqslant i \leqslant k \leqslant j \leqslant n$, we want to report if point p_k is on maximal layer L_γ, where $\gamma = 2$ or $\geqslant 3$ of $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$. First we solve the problem for $\gamma \geqslant 3$ and then show that the result for $\gamma = 2$ follows.

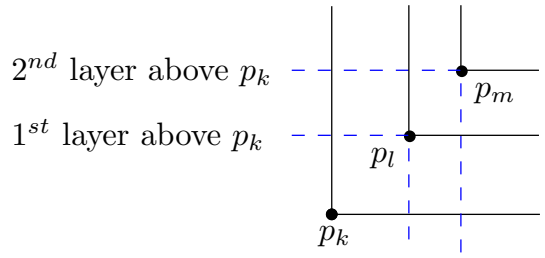


Figure 4.6: Point p_k on some maximal layer $\geqslant 3$ iff $p_\ell \in NE(p_k)$ and $p_m \in NE(p_\ell)$.

LEMMA 4.19. *Recall the definitions of* $\alpha(.)$ *and* $\beta(.)$ *from Section 4.3.1. Suppose* $1 \leqslant i \leqslant k \leqslant j \leqslant n$*. The point* p_k *is on layer* L_γ*, where* $\gamma \geqslant 3$ *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* $\ell \geqslant i$, $p_\ell \in NE(p_k)$, *and* $\alpha(\ell) \leqslant j$
2. *There exists some* ℓ *such that* $\ell \leqslant j$, $p_\ell \in NE(p_k)$, *and* $\beta(\ell) \geqslant i$.

Lemma 4.19. The 'if' part is obvious from the definitions of $\alpha(\ell)$ and $\beta(\ell)$, and by Lemma 4.13. To prove the converse, we assume p_k to be a point on some maximal layer L_γ, where $\gamma \geqslant 3$, of points in $P_{i,j}$. Then there must exist some ℓ and m such that $i \leqslant l \leqslant j$, $i \leqslant m \leqslant j$, $p_\ell \in NE(p_k)$ and $p_m \in NE(p_\ell)$ (see Figure 4.6). Note that 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., $m > \ell$. Since $\alpha(\ell) \leqslant m$ by the definition of $\alpha(\ell)$, we obtain $\alpha(\ell) \leqslant m \leqslant j$. *Case 2*: Suppose p_m comes before p_ℓ, i.e., $m < \ell$. By the similar argument since $\beta(\ell) \geqslant m$ by the definition of $\beta(\ell)$, we obtain $\beta(\ell) \geqslant m \geqslant j$. □

We map each point $p_\ell = (p_{\ell,x}, p_{\ell,y}) \in P$, where $1 \leqslant l \leqslant n$, to a point in $\mathbb{R}^4$ as follows. For part (1) of Lemma 4.19, we define a function $f(\ell) = (p_{\ell,x}, p_{\ell,y}, \ell, \alpha(\ell)) \in \mathbb{R}^4$. We set $S = \{f(\ell) : 1 \leqslant l \leqslant n\}$. Similarly, for part (2) of Lemma 4.19, we define a function $g(\ell) = (p_{\ell,x}, p_{\ell,y}, \ell, \beta(\ell)) \in \mathbb{R}^4$. We set $T = \{g(\ell) : 1 \leqslant l \leqslant n\}$. For each $1 \leqslant l \leqslant n$, $\alpha(\ell)$ and $\beta(\ell)$ 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 $\geqslant 3$ 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 $1 \leqslant i \leqslant k \leqslant j \leqslant n$, 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 $G(i, j, k) = [p_{k,x}, \infty) \times [p_{k,y}, \infty) \times (-\infty, j] \times [i, \infty)$. This gives us the following lemma.

LEMMA 4.20.

1. *There exists some* ℓ *such that* $\ell \geqslant i$, $p_\ell \in NE(p_k)$, *and* $\alpha(\ell) \leqslant j$ *if and only if* $F(i, j, k) \cap S \neq \emptyset$.
2. *There exists some* ℓ *such that* $\ell \leqslant j$, $p_\ell \in NE(p_k)$, *and* $\beta(\ell) \geqslant i$ *if and only if* $G(i, j, k) \cap T \neq \emptyset$.

Combining Lemma 4.19 and Lemma 4.20, the query of the form 'Given $1 \leqslant i \leqslant k \leqslant j \leqslant n$, decide if p_k is on maximal layer $\gamma \geqslant 3$ 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 $\gamma \geqslant 3$. Each query takes $O(\log^3 n)$ time. The following theorem summarizes the results.

THEOREM 4.21. *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* $i \leqslant k \leqslant j$, *we can answer whether* p_k *is on some maximal layer* L_γ, *where* $\gamma \geqslant 3$, *of* $P_{i,j} = (p_i, p_{i+1}, \ldots, p_j)$ *in* $O(\log^3 n)$ *time.*

Corollary 4.22 follows from the results of Theorem 4.17 and Theorem 4.21.

COROLLARY 4.22. *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* $i \leqslant k \leqslant j$, *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.*

4.3.4 *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 $k \geqslant 1$ is an integer, and still p can be on some maximal layer L_ϕ, where $\phi \leqslant k+1$. See Figure 4.7 for an example.

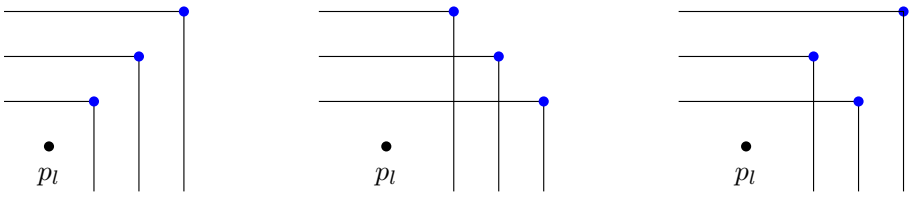


Figure 4.7: Suppose k=3, then p_ℓ is a k-dominated point in all examples. However, the maximal layer point p_ℓ is on is L_4 in (a), L_2 in (b), and L_3 in (c).

Problem statement: Given a sequence of n points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$, and a fixed integer $k \geqslant 1$, 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 $1 \leqslant l \leqslant n$, we define the following. Let $p_{\ell_1}, p_{\ell_2}, \ldots, p_{\ell_k}$ be the first k points that dominate p_ℓ, where $\ell < \ell_1 < \ell_2 < \ldots < \ell_k$. Similarly, let $p_{\ell'_k}, \ldots, p_{\ell'_2}, p_{\ell'_1}$ be the last k points that dominate p_ℓ, where $\ell'_k < \ell'_{k-1} < \ldots < \ell'_1 < \ell$ (see Figure 4.8). Then each interval (ℓ'_{k-a}, ℓ_a), with $0 \leqslant a \leqslant k$, represents k points that dominate point p_ℓ. Here $\ell'_0 = \ell_0 = \ell$. There exist at most $k+1$ such intervals for each point in P. We obtain the following lemma.

LEMMA 4.23. *Suppose* $1 \leqslant i < j \leqslant n$, *and* k *is a fixed integer. A point* $p_\ell \in P_{i,j}$ *is dominated by at least* k *points in* $P_{i,j}$ *if and only if there exists some* a *with* $0 \leqslant a \leqslant k$, *such that* $\ell_a \leqslant j$ *and* $\ell'_{k-a} \geqslant i$.

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

We can find the points $p_{\ell_1}, p_{\ell_2}, \ldots, p_{\ell_k}$ and $p_{\ell'_k}, \ldots, p_{\ell'_2}, p_{\ell'_1}$ for each p_ℓ by extending the data structure from Section 4.3.1 (see Algorithm 8) as follows. Each point p_ℓ now has two additional arrays $B_\ell[1..k]$ and $A_\ell[1..k]$ of size k that store k points that dominate p_ℓ and appears, respectively, before and after time ℓ. To achieve this, we modify line 6 of Algorithm 8 by lines 9-11 in Algorithm 9. Moreover, we do not delete any points from T (line 7 of Algorithm 8 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

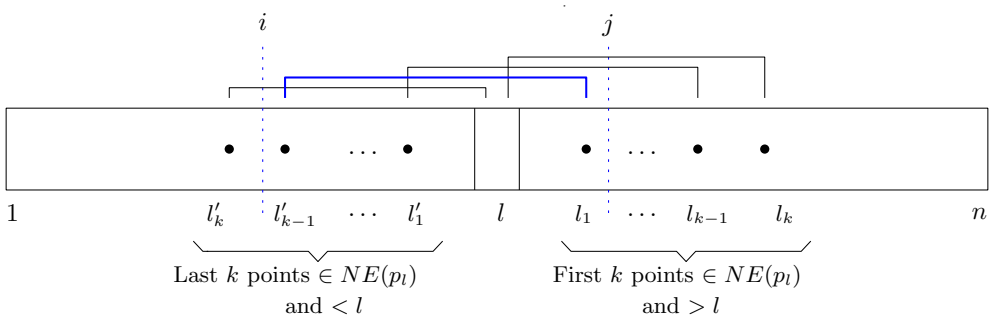


Figure 4.8: The highlighted interval (ℓ'_{k-1}, ℓ_1) satisfies the query interval $[i, j]$ for k-dominated points.

(i.e., starting 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 9: 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      Query T with q = (−∞, p_{i,x}] × (−∞, p_{i,y}].
7      Let S be the output of this query.
8      for each p_ℓ ∈ S do
9          if SizeA[l] < k then
10             Increase SizeA[l] by 1.
11             Store i to A_ℓ[SizeA].
12     Insert p_i into T.
```

We build a range tree on the first coordinate of the $(k+1)n$ points $(\ell, \ell_a, \ell'_{k-a})$. For each canonical node we add a PST that is built on the last two coordinates of the points stored in that node. It 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 $1 \leqslant l \leqslant n$. 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 update $R[l] \leftarrow 1$. If $R[l]$ contains 1 then p_ℓ is already reported before and we do not report it this time. Reset $R[1 \ldots n]$ to 0 in $O(w)$ time. The entire query takes $O(\log^2 n + kw)$ time, where w is the output size.

THEOREM 4.24. *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. 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.*

4.3.5 *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 $1 \leqslant l \leqslant n$, let $p_{\ell_1}, p_{\ell_2}, \ldots, p_{\ell_k}$ be the first k points that are dominated by the point p_ℓ, where $\ell < \ell_1 < \ldots < \ell_k$. Similarly, $p_{\ell'_k}, \ldots, p_{\ell'_2}, p_{\ell'_1}$ be the last k points that are dominated by p_ℓ, where $\ell'_k < \ell'_{k-1} < \ldots < \ell'_1 < \ell$ (see Figure 4.9). As shown before each interval (ℓ'_{k-a}, ℓ_a) with $0 \leqslant a \leqslant k$ represents k points that are dominated by point p_ℓ and $\ell'_0 = \ell_0 = \ell$. Now we obtain the following lemma.

LEMMA 4.25. *Suppose* $1 \leqslant i < j \leqslant n$, *and* k *is a fixed integer. A point* $p_\ell \in P_{i,j}$ *dominates at least* k *points in* $P_{i,j}$ *if and only if there exists some* a *with* $0 \leqslant a \leqslant k$, *such that* $i \leqslant l \leqslant j$, $\ell_a \leqslant j$ *and* $\ell'_{k-a} \geqslant i$.

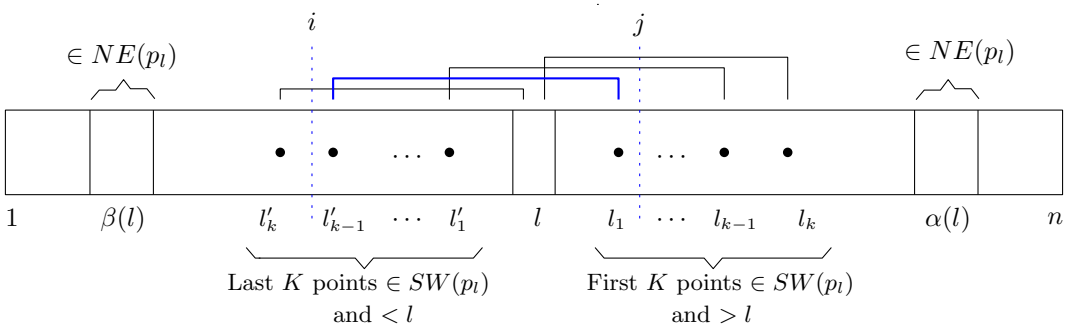


Figure 4.9: Proof of Lemma 4.26. For an example, the highlighted interval (ℓ'_{k-1}, ℓ_1) satisfies the query interval $[i, j]$.

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 $1 \leqslant l \leqslant n$, we map point p_ℓ to at most $k+1$ points $(\ell, \alpha(\ell), \beta(\ell), \ell_a, \ell'_{k-a}) \in \mathbb{R}^5$, where $0 \leqslant a \leqslant k$. This again gives us a set of at most $(k+1)n$ points. From Lemma 4.13 and Lemma 4.25 we obtain the following.

LEMMA 4.26. *Suppose* $1 \leqslant i \leqslant l \leqslant j \leqslant n$ *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* $(\ell, \alpha(\ell), \beta(\ell), \ell_a, \ell'_{k-a}) \in \mathbb{R}^5$, $\alpha(\ell) > j$ *and* $\beta(\ell) < i$.

We modify the query q in line 6 of Algorithm 9 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 $(\ell, \alpha(\ell), \beta(\ell), \ell_a, \ell'_{k-a})$. 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 same technique applied in Section 4.3.4. Each query takes $O(\log^4 n + kw)$ time, where w is the output size.

THEOREM 4.27. *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. 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.*

4.3.6 *Generalization to Higher Dimensions*

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

DEFINITION 4.28. *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 $p[k] \leqslant p'[k]$ for $1 \leqslant k \leqslant d$ and $p \neq p'$.*

Recall that we presented the data structure for computing $\alpha(.)$ and $\beta(.)$ for points in $\mathbb{R}^d$ in Section 4.3.1. 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 dimension 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 dimension 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 4.16 we directly obtain the following results for the general case (for a fixed $d \geqslant 2$).

THEOREM 4.29. *A sequence of n points $P = (p_1, p_2, \cdots, p_n)$ in $\mathbb{R}^d$, where $d \geqslant 3$, 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 $i \leqslant k \leqslant j$, we can report whether p_k is on the maximal layer L_1 of points $P_{i,j}$ in time $O(1)$.*

THEOREM 4.30. *A sequence of n points $P = (p_1, p_2, \cdots, p_n)$ in $\mathbb{R}^d$, where $d \geqslant 4$, 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 $2 \leqslant d \leqslant 3$, $O(n \log^2 n)$ space and $O(n \log^2 n)$ time are required for preprocessing.*

THEOREM 4.31. *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 $2 \leqslant d \leqslant 3$, $O(kn \log n)$ space and $O(kn \log^2 n)$ time are required for preprocessing.*

THEOREM 4.32. *Suppose $P = (p_1, p_2, \cdots, p_n)$ 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* $2 \leqslant d \leqslant 5$, $O(kn \log^3 n)$ *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 $f(\ell)$ and $g(\ell)$ to map each point $p_\ell = (p_{\ell,x1}, p_{\ell,x2}, \cdots, p_{\ell,xd}) \in P$, where $1 \leqslant l \leqslant n$, to a point in $\mathbb{R}^{d+2}$ as follows. We redefine the functions as $f(\ell) = (p_{\ell,x1}, p_{\ell,x2}, \cdots, p_{\ell,xd}, \ell, \ell') \in \mathbb{R}^{d+2}$ and $g(\ell) = (p_{\ell,x1}, p_{\ell,x2}, \cdots, p_{\ell,xd}, \ell, \ell'') \in \mathbb{R}^{d+2}$. 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 4.33. *Suppose* $P = (p_1, p_2, \cdots, p_n)$ *is a sequence of* n *points in* $\mathbb{R}^d$*, where* $d \geqslant 3$. 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* $i \leqslant k \leqslant j$*, we can decide whether* p_k *is on some maximal layer* L_γ*, where* $\gamma = 2, \geqslant 3$*, of the point* $P_{i,j} = (p_i, p_{i+1}, \cdots, p_j)$ *in* $O(\log^{d+1} n)$ *time.*

4.4 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 $\mu(P) \geqslant 0$ 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 $1 \leqslant i < j \leqslant n$ 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 [10] 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 (for example, [1, 3, 4, 10]) showed that many of these geometric problems can be linearly approximated by computing and using constant size *coresets* (see Definition 4.34).

DEFINITION 4.34. *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* $\mu(P') \geqslant (1-\epsilon) \cdot \mu(P)$ *[3].*

More specifically, Agarwal et al. [3] 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 orthogonal to the direction that encloses P.

DEFINITION 4.35. *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 [3]. 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 $f(\epsilon)$ 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 $f(\epsilon)$ (note that the size of the coreset depends on d but not on the size of P). We restate Lemma 23.3 from [21] here.

LEMMA 4.36. *[21, 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 4.37. *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 $O(f(\epsilon))$ time [28].*

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 $\ell \geqslant 2$ is an input parameter [1].

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

4.4.1 *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 in [28] for computing the range-aggregate queries. We divide the sequence of points in P into a sequence of $n/f(\epsilon)$ blocks $S = (S_1, S_2, \ldots S_{n/f(\epsilon)})$ such that each block S_k contains $f(\epsilon)$ 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 $f(\epsilon)$ points belonging to the block S_k, for all $1 \leqslant k \leqslant n/f(\epsilon)$. Each leaf node contains the coreset of $f(\epsilon)$ points. As we move to the upper levels of T, coresets of the child nodes can be combined to compute the coreset of size $O(\log(n/f(\epsilon)) \cdot f(\epsilon))$ for each canonical node v of T. However, we can reduce the size of each coreset to $O(f(\epsilon))$ 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 $f(\epsilon) = O(1/\epsilon^{d-1})$. When $d = 2$, we obtain $f(\epsilon) = 2\pi/\epsilon$. During a query, the coreset of each canonical node can be computed by comparing the $O(f(\epsilon))$ points of its child nodes in $O(f(\epsilon))$ time. Total number of canonical nodes in T is $O(n/f(\epsilon))$. Therefore total space required for T is $O(n/f(\epsilon) \cdot \log(n/f(\epsilon)) \cdot f(\epsilon))$=$O(n \log n)$.

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 these S_x and S_y nodes contains at most $f(\epsilon)$ 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 $O(\log(n/f(\epsilon)))$ canonical nodes of T. Therefore, the coreset $C(P_{i,j}, \epsilon)$ of size $f(\epsilon)$ can be computed in $O(\log(n/f(\epsilon)) \cdot f(\epsilon)) = O(f(\epsilon) \log n)$ time.

THEOREM 4.38. *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 $1 \leqslant i < j \leqslant n$, we can compute a $(1+\epsilon)$-coreset of size $f(\epsilon)$ in time $O(f(\epsilon) \log n)$ for geometric problems that admit some decomposable coresets.*

Applications: Theorem 4.38 can be used to compute $\mu(P_{i,j})$ for any queried sequence of points if $\mu(\cdot)$ admits a decomposable coreset of size $f(\epsilon)$. 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 $f(\epsilon)$ such that $\mu(P') \geqslant (1-\epsilon)\mu(P)$. Then by Theorem 4.38 we compute $C(P_{i,j}, \epsilon)$ of size $f(\epsilon)$ 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 $O((f(\epsilon)^2 + f(\epsilon) \log n)$ time. Table 4.3 shows running times for computing $(1+\epsilon)$-approximations to $\mu(\cdot)$ for some well known geometric problems.

Diameter: the maximum distance over all pairs of points in P. In $d = 2$ and 3, $O(n \log n)$ time bound is available (randomized for $d = 3$ [18]). 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 Δ refers to a region between two parallel hyperplanes of distance Δ. In $d = 3$, $O(n^2)$ time algorithm is available [22]. *Smallest enclosing disc*: the disc with the minimum radius that contains all the points in P. An expected linear time algorithm is available in [29]. *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 $d = 3$ and for a small constant $\delta > 0$, a $O(n^{3+\delta})$ time algorithm is presented in [2]; *Minimum-width annulus*: the minimum width over all annuli that encloses 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 algorithm is available in [5].

Table 4.3: Window-aggregate query time for $(1+\epsilon)$-approximation for some geometric problems. Here n is the total number of input points and $\delta > 0$ is a small constant.

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

4.4.2 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 $1 \leqslant i < j \leqslant n$, and an integer $\ell \geqslant 2$, 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 2 types of cost functions of a ℓ-center clustering denoted by $\Phi(C)$; (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 of the clusters: $\min \Phi^{sum}(C) = \sum_{c_\ell \in C} radius(c_\ell)$.

Abrahamsen et al. [1] present a coreset based algorithm that computes a $(1+\epsilon)$-approximation to the range-clustering query for ℓ-center clustering by using a ϵ-coreset of the input points. To be more specific, given a set P of n points in $\mathbb{R}^d$, for any query box Q, a fixed integer $\ell \geqslant 2$ 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 ℓ-center clustering C for the points in $P \cap Q$ using points from P' such that $\Phi(C) \leqslant (1+\epsilon) \cdot \Phi(C_{opt})$, 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 (Algorithm $\mathcal{A}$) that returns an ϵ-coreset P' of P_Q (i.e., points in $P \cap Q$) of size $O(\ell(f(\ell)/(c\epsilon))^d)$ and then show how this data structure can be used to compute coresets for ℓ-center clustering in the window query setting.

Algorithm $\mathcal{A}$ from [1]: 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 and let LB be the size of the largest square after the refinement (this LB is the lower bound on the optimal value of the ℓ-clustering of P_Q). Set a parameter $r = \epsilon \cdot LB/f(\ell)$. Repeat refinement of $\mathcal{B}$ 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 from [1] in terms of $d = 2$.

LEMMA 4.39. *[1, 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 $r = \epsilon \cdot LB/f(\ell)$ of size $O(\ell(f(\ell)/(c\epsilon))^2)$.*

LEMMA 4.40. *[1, Lemma 6] Algorithm $\mathcal{A}$ runs in $O(\ell(f(\ell)/(c\epsilon))^2 + \ell((f(\ell)/(c\epsilon)) \log n))$ 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 the increasing order of 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 $1 \leqslant q < r \leqslant n$, in the subtree rooted at v. Algorithm $\mathcal{A}$ can be applied for computing coresets from a quad tree 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 $O(\ell(f(\ell)/(c\epsilon))^2)$ can be computed for $P_{i,j}$ according to Algorithm

$\mathcal{A}$. We apply the technique presented in [6] to construct a compressed quad-tree for the queried subset of points in time linear to 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 [26]. A skip-quadtree is a linear size variant of the compressed quadtree that is built on a structure similar to the skip lists [20].

During any query $q = [i, j]$, we find two overlapping canonical subsets in T each of width less than $2\mathcal{W}$ in $O(\log \mathcal{W})$ time 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 [6]). According to [11], a compressed quad tree can be computed from the final Z-order of $P_{i,j}$ in linear $O(\mathcal{W})$ time. Now we apply Algorithm $\mathcal{A}$ to compute a coreset for points in $P_{i,j}$ for ℓ-center clustering. By Lemmas 11 and 12 we summarize the following result.

THEOREM 4.41. *Let P be a sequence of n points in $\mathbb{R}^2$. 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 $\ell \geqslant 2$ and $\epsilon > 0$, a coreset of size $O(\ell(f(\ell)/(c\epsilon))^2)$ for the ℓ-center clustering can be computed in $O(\ell((f(\ell)/(c\epsilon)) \log n) + \ell(f(\ell)/(c\epsilon))^2 + \mathcal{W})$ 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 $1 \leqslant i < j \leqslant n$, we can compute an $(1+\epsilon)$-approximation to the cost of the ℓ-center clustering of points in $P_{i,j}$ within the following bounds (by Corollary 8 in [1]).

- 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 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.

4.5 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 window queries for counting maximal points, and reporting whether a given point is on the maximal layers L_γ ($\gamma = 1, 2, \geqslant 3$), all k-dominated points and all k-dominant points in $\mathbb{R}^d$, where $d \geqslant 2$. 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 ($\ell \geqslant 2$) using coresets.

Remarks. The query time of the windowed intersection decision problem can be improved so that given a query interval $[i, j]$, with $1 \leqslant i \leqslant j \leqslant n$, 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 $1 \leqslant \alpha \leqslant n$, 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 $\beta > \alpha$ (see Section 4.2.2). Let $\mathcal{V}$ be the set of these n valid pairs (points). We store these (α, β) points in an orthogonal range successor data structure of size $n + o(n)$ such that for a query region $Q = [\alpha, +\infty) \times [\alpha, +\infty)$, in $O(\log n)$ time, it reports the point $p = (x, y)$ such that $p \in \mathcal{V} \cap Q$ and y has the minimum β value among all points that lie in $\mathcal{V} \cap Q$ [23]. We set $B[\alpha] = y$. For all $\alpha = 1, \ldots, n$, we can find $B[\alpha]$ in $O(n \log n)$ time which is upper bounded by the original preprocessing time (see, for example, Corollary 4.7). For a given query interval $q = [i, j]$, if $j \geqslant B[i]$ we report that some objects intersect in q, otherwise all objects in q are disjoint. This improves the query time to $O(1)$.

BIBLIOGRAPHY

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

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

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

[4] Agarwal, P.K., Har-Peled, S., Varadarajan, K.R.: Geometric approximation via coresets. Combinatorial and computational geometry 52, 1–30 (2005)

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

[6] Bannister, M.J., Devanny, W.E., Goodrich, M.T., Simons, J.A., Trott, L.: 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), http://www.cccg.ca/proceedings/2014/papers/paper02.pdf

[7] Bannister, M.J., DuBois, C., Eppstein, D., Smyth, P.: 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. pp. 856–864 (2013), https://doi.org/10.1137/1.9781611973105.61

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

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

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

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

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

[13] Chan, T.M., Pratt, S.: 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. LIPIcs, vol. 51, pp. 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016), https://doi.org/10.4230/LIPIcs.SoCG.2016.28

[14] Chanchary, F., Maheshwari, A.: Counting subgraphs in relational event graphs. In: WALCOM: Algorithms and Computation - 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9627, pp. 194–206. Springer (2016), https://doi.org/10.1007/978-3-319-30139-6_16

[15] Chanchary, F., Maheshwari, A., Smid, M.H.M.: Querying relational event graphs using colored range searching data structures. In: Algorithms and Discrete Applied Mathematics - Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10156, pp. 83–95. Springer (2017), https://doi.org/10.1007/978-3-319-53007-9_8

[16] Chanchary, F., Maheshwari, A., Smid, M.: Window queries for problems on intersecting objects, maximal points and approximation using core sets. Submitted to Journal of Discrete Applied Mathematics (2018)

[17] Chanchary, F., Maheshwari, A., Smid, M.H.M.: 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. Lecture Notes in Computer Science, vol. 10743, pp. 199–213. Springer (2018), https://doi.org/10.1007/978-3-319-74180-2_17

[18] Clarkson, K.L.: 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. pp. 1–11. ACM (1988), https://doi.org/10.1145/73393.73394

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

[20] Eppstein, D., Goodrich, M.T., Sun, J.Z.: 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. pp. 296–305 (2005), https://doi.org/10.1145/1064092.1064138

[21] Har-Peled, S.: Geometric approximation algorithms. American Mathematical Soc. (2011)

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

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

[24] McCreight, E.M.: Priority search trees. SIAM J. Comput. 14(2), 257–276 (1985), https://doi.org/10.1137/0214021

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

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

[27] Mouratidis, K., Bakiras, S., Papadias, D.: 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. pp. 635–646. ACM (2006), https://doi.org/10.1145/1142473.1142544

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

[29] Welzl, E.: Smallest enclosing disks (balls and ellipsoids). New results and new trends in computer science pp. 359–370 (1991)

5

THE MOST LIKELY OBJECT TO BE SEEN THROUGH A WINDOW

We study data structures to answer *window queries* using stochastic input sequences. We present algorithms to solve two window problems defined as follows. The first problem is the c*-colored most likely maximal point* in a query window, where $c \geqslant 2$ is a constant. Let α_1 and α_2 be constants with $0 < \alpha_1 < \alpha_2 < 1$ and c=2. A sequence of blue and red stochastic points $P = (p_1, \ldots, p_n)$ in $\mathbb{R}^d$ is given, where every blue point is associated with probability α_1 and every red point is associated with probability α_2 and $d \geqslant 1$. A point $p = (x_1, \ldots, x_d)$ in P is on the maximal layer of P if there is no other point $q = (x'_1, \ldots, x'_d)$ in P such that $x'_1 > x_1, x'_2 > x_2, \ldots,$ and $x'_d > x_d$. Let Z be a random subset of P obtained by including each blue point of P in Z independently with probability α_1 and each red point of P in Z independently with probability α_2. For a query interval $[i, j]$, with $1 \leqslant i \leqslant j \leqslant n$, we report the point that has the highest probability to be on the *maximal layer* of the queried points $(p_i, \ldots, p_j)$. The second problem we consider is the *most likely common element* problem. Let $\mathcal{U} = \{1, 2, \ldots, n\}$ be the universe. Let $S_1, S_1, \ldots, S_m$ be a sequence of random subsets of $\mathcal{U}$ such that for $p = 1, \ldots, m$ and $i = 1, \ldots, n$, element i is added to S_p with probability α_{pi} (independently of other choices). Let τ be a fixed real number with $0 < \tau \leqslant 1$. For query indices p, q, i and j, with $1 \leqslant p \leqslant q \leqslant m$ and $1 \leqslant i \leqslant j \leqslant n$, decide whether there exists an element k with $i \leqslant k \leqslant j$ such that $\Pr(k \in \cap_{r=p}^{q} S_r) \geqslant \tau$.

This chapter presents the results that have been submitted to the International Journal of Computational Geometry & Applications [13].

5.1 INTRODUCTION

Recent developments in real-world data acquisition methods (for example, reading data through sensor networking), various forms of scientific measurements, and subsequent data management techniques (such as cleaning and integrating datasets) have led to a massive data generation with some inherent uncertainty. A variety of uncertainty models have been devised to categorize and study these data. These models are studied not just in the field of data structures and algorithms, but also in data mining [6, 31], database management [18], statistics [28], physics [26], GIS and remote sensing [19].

In a *window query* we are given a sequence of input data and a predicate $\mathcal{P}$. We want to preprocess the input into a suitable data structure such that given a query window, it can efficiently answer the query based on only the input elements that lie in the query window and matches $\mathcal{P}$. Let i and j be two positive integers, with $i \leqslant j$, such that the interval $[i, j]$ represents a *query window* with width $j - i + 1$. In this paper we study *window queries* for *stochastic* input sequences such as geometric objects (points) and elements of sets. We present data structures to solve two window problems, namely *the most likely maximal point problem* and *the most likely common element problem*. The problem definitions are as follows.

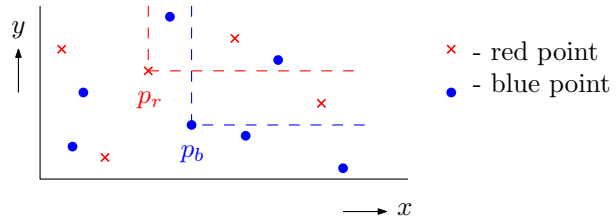


Figure 5.1: An instance of MLMP$(B, \alpha_1; R, \alpha_2)$ in $\mathbb{R}^2$ is given. Pr(p_b is on maximal layer in P) $= \alpha_1(1-\alpha_1)(1-\alpha_2)^2$ and Pr(p_r is on maximal layer in P) $= \alpha_2(1-\alpha_1)^2(1-\alpha_2)$.

MLMP: Most Likely Maximal Point problem. For a point $p = (x_1, \ldots, x_d) \in P$ in $\mathbb{R}^d$, with $d \geqslant 1$, we define $\mathcal{D}$ to be the set of points of P that dominate p, i.e., $\mathcal{D}(p) = \{q = (x'_1, \ldots, x'_d)$ of P: $x'_1 > x_1, \ldots, x'_d > x_d\}$. A point p is dominated by q if $q \in \mathcal{D}(p)$. A point $p \in P$ is a maximal point if $\mathcal{D}(p) \cap P = \emptyset$. The maximal layer is the set of all maximal points in P.

Let α_1 and α_2 be constants with $0 < \alpha_1 < \alpha_2 < 1$. Let $P = (p_1, \ldots, p_n)$ be a sequence of n points in $\mathbb{R}^d$ with $d \geqslant 1$. Let each point of P be either colored blue or red. Let $B \subseteq P$ be the set of all blue points and let $R \subseteq P$ be the set of all red points. Furthermore, each blue point is associated with probability α_1 and each red point is associated with probability α_2. Note that $P = B \cup R$ and we denote this instance of P as $(B, \alpha_1; R, \alpha_2)$. Let Z be a random subset of P given by including each blue point of P in Z independently with probability α_1 and each red point of P in Z independently with probability α_2.

For a point $p' \in P$, let s be the number of blue points of P in $\mathcal{D}(p')$. Similarly let t be the number of red points of P in $\mathcal{D}(p')$. If p' itself is a blue point then the probability that p' is on the maximal layer in Z is $\alpha_1(1-\alpha_1)^s(1-\alpha_2)^t$. Similarly if p' is a red point then the probability that p' is on the maximal layer in Z is $\alpha_2(1-\alpha_1)^s(1-\alpha_2)^t$. See Figure 5.1 for an example.

For $1 \leqslant i < j \leqslant n$, let $P_{i,j}$ denote the subsequence of stochastic points $(p_i, p_{i+1}, \ldots, p_j)$. Given a query interval $[i, j]$, with $i \leqslant j$, we report the most likely maximal point (MLMP) defined to be the point with the highest probability such that it is not dominated by any other input points in $P_{i,j}$.

Given a query interval $[i, j]$, we define $P_{i,j} = B_{i,j} \cup R_{i,j}$, where $B_{i,j} = B \cap P_{i,j}$ and $R_{i,j} = R \cap P_{i,j}$. We present data structures that can answer window queries for the most likely maximal point in the above mentioned instance of P. Now the window problem of the most likely maximal point is denoted as MLMP$(B_{i,j}, \alpha_1; R_{i,j}, \alpha_2)$.

Next we show how to extend this technique to solve the c-colored MLMP problem defined as follows. Let $\alpha_1, \ldots, \alpha_c$ be constants, with $0 < \alpha_1 < \alpha_2 < \ldots < \alpha_c < 1$. Given a sequence of stochastic points $P = P_1 \cup P_2 \cup \ldots \cup P_c$ in $\mathbb{R}^d$, where c is a constant and for $r = 1, \ldots, c$, each r-colored point belongs to P_r has a probability α_r associated with it and $d \geqslant 1$. P can be preprocessed into a data structure that can report the most likely maximal point within $P_{i,j}$, with $1 \leqslant i \leqslant j \leqslant n$.

We also answer the following query. Suppose a sequence of stochastic points $P = (p_1, p_2, \ldots, p_n)$ in $\mathbb{R}^2$ is given such that for $k = 1, 2, \ldots, n$, point p_k exists with probability γ_k, with $0 \leqslant \gamma_k \leqslant 1$, independently of the other points. Given a query interval $[i, j]$, with $i \leqslant j$, and an additional integer $t \in [i, j]$, we can report the probability for the point p_t to be on the maximal layer in $P_{i,j}$.

MLCE: Most Likely Common Element problem. Let $\mathcal{U} = \{1, 2, \ldots, n\}$ be the universe. Let $S_1, S_1, \ldots, S_m$ be a sequence of random subsets of $\mathcal{U}$ such that for $p = 1, \ldots, m$ and $i = 1, \ldots, n$, element i is added to S_p with probability α_{pi} (independently of other choices). Let τ be a fixed real number with $0 < \tau \leqslant 1$. We answer queries of the following type: Given query indices p, q, i and j, with $1 \leqslant p \leqslant q \leqslant m$ and $1 \leqslant i \leqslant j \leqslant n$, decide if there exists an element $k \in \{i, i+1, \ldots, j\}$ that is contained in $S_p \cap S_{p+1} \cap \ldots, \cap S_q$ with probability at least τ.

Observe that $\Pr(k \in \cap_{r=p}^{q} S_r) = \prod_{r=p}^{q} \alpha_{rk}$. Thus, the problem is equivalent to the following: Let $\mathcal{S}$ be an $m \times n$ matrix in which each entry $\mathcal{S}[p, i]$ is a real number $\alpha_{pi} \in [0, 1]$. We want to preprocess $\mathcal{S}$ into a data structure such that for any query values p, q, i and j, we can decide if there exists some integer k for which $i \leqslant k \leqslant j$ and $\prod_{r=p}^{q} \alpha_{rk} \geqslant \tau$.

5.1.1 *Related Work*

The window data structures have been considered in many recent studies including [8, 9, 12, 14, 15, 16, 17]. These studies solve window queries for various geometric and graph problems and consider sequences of geometric objects (such as, points, line segments and polygons) or graph edges as input. However none of these papers have considered any *model of uncertainty* in their studies. Typically, models of uncertainty describe a distribution or regions for each point's location in the input sequence. Many papers have studied (offline) geometric problems with one or more of these models. The most widely studied models of uncertainty are *stochastic points*, where each point p_k has a fixed location which only exists with a probability α_k, with $0 < \alpha_k \leqslant 1$ (see [7, 22, 23, 32]); *uncertain points*, where each point p_k's location is described by a probability distribution α_k (see [21, 2, 29]); *indecisive points* or *multipoint* model, where each point can take one of a finite number of locations (see [4, 21, 30]) and *imprecise points*, where each point's location is not known precisely but it is restricted to a region (see [20, 25]).

Although these uncertainty models have not been studied so far in the window query setting, various authors presented range searching data structures on stochastic or uncertain geometric objects (e.g., points and line segments) such as range counting coresets for uncertain data [1], range reporting with a query interval and a probability threshold [3], range-maximum query on uncertain data [5], range queries (i.e., report top-k points with highest probabilities) given some query interval [24].

5.1.2 *New Results*

The main contributions of this paper are listed below.

1. *Window queries for c-colored MLMP problems:* We show how to preprocess a sequence of n stochastic points in $\mathbb{R}^d$, for a fixed $d \geqslant 1$, into a data structure of size $O(n \log n)$ so that given a query interval $[i, j]$ with $1 \leqslant i \leqslant j \leqslant n$, it can report the most likely maximal point of $P_{i,j}$ in $O(1)$ time.

2. *Window queries for MLCE problems:* We show how to preprocess a matrix $\mathcal{S}$ of size $m \times n$ into a data structure of size $O(mn)$ in $O(mn)$ time such that for any query values p, q, i and j, we can decide if there exists some integer k for which $i \leqslant k \leqslant j$ and $\prod_{r=p}^{q} \alpha_{rk} \geqslant \tau$ in $O(1)$

time, where τ is a fixed real number with $0 < \tau \leqslant 1$. The reporting version of the query can be answered in $O(\log n + w)$ time, where w is the total number of all elements k for which $i \leqslant k \leqslant j$ and $\prod_{r=p}^{q} \alpha_{rk} \geqslant \tau$.

5.1.3 *Organization*

The rest of this paper is organized as follows. Section 2 presents results for the c-colored windowed most likely maximal point problem. This section also outlines a data structure that can report the probability with which a given point can exist on the maximal layer of the queried points. In Section 3, we present algorithms and data structures for the most likely common element problem. Section 4 concludes this paper.

5.2 MOST LIKELY MAXIMAL POINTS

The point that has the highest probability to be on the maximal layer in P is defined to be the *most likely maximal point* in P. The window problem of finding the most likely maximal point is as follows. Given a query interval $q = [i, j]$, with $1 \leqslant i \leqslant j \leqslant n$, report the point that has the highest probability to be on the maximal layer in $P_{i,j}$.

In this section, we initially present the MLMP problem where the points of P are only colored by two colors. We later extend to c-colored MLMP, where $c \geqslant 2$ is a constant. We further present the data structure for computing the probability for a point to be on the maximal layer in $P_{i,j}$.

5.2.1 *Blue-Red Stochastic Points*

Given a query interval $[i, j]$, $1 \leqslant i \leqslant j \leqslant n$, we define $P_{i,j} = \{p_i, p_{i+1}, \ldots, p_j\}$, $B_{i,j}$ is the set of blue points in $B \cap P_{i,j}$ and $R_{i,j}$ is the set of blue points in $R \cap P_{i,j}$. We denote the windowed version of the most likely maximal point problem as $\mathrm{MLMP}(B_{i,j}, \alpha_1; R_{i,j}, \alpha_2)$.

LEMMA 5.1. *Consider a query interval $[i, j]$ with $i \leqslant j$. If the answer to the most likely maximal point problem is a blue point then it is a maximal point of all points in $P_{i,j}$. If the answer to the most likely maximal point problem is a red point then it is a maximal point in $R_{i,j}$.*

Proof. Suppose the answer to our query is a blue point p_b. We show, by contradiction, that p_b is a maximal point of $P_{i,j}$. Suppose it is not. Then $\mathcal{D}(p_b) \cap P_{i,j} \neq \emptyset$ and assume that there are s points of color blue and t points of color red in $\mathcal{D}(p_b)$, such that $s + t > 0$. Then the probability that p_b is the most likely maximal point is $\alpha_1(1-\alpha_1)^s(1-\alpha_2)^t$ but that is strictly less than α_1. It contradicts the assumption that p_b is the most likely maximal point in $P_{i,j}$.

By a similar argument the second statement can also be proved. Suppose the answer to our query is a red point p_r but there exists t other red points in $\mathcal{D}(p_r)$ in $R_{i,j}$. Then the probability that p_r is the most likely maximal point in $P_{i,j}$ is $\alpha_2(1-\alpha_2)^t$ which is strictly less than α_2. □

First we discuss the case where points of $P = B \cup R$ lie on the line. Without loss of generality, we assume P lies on a horizontal line.

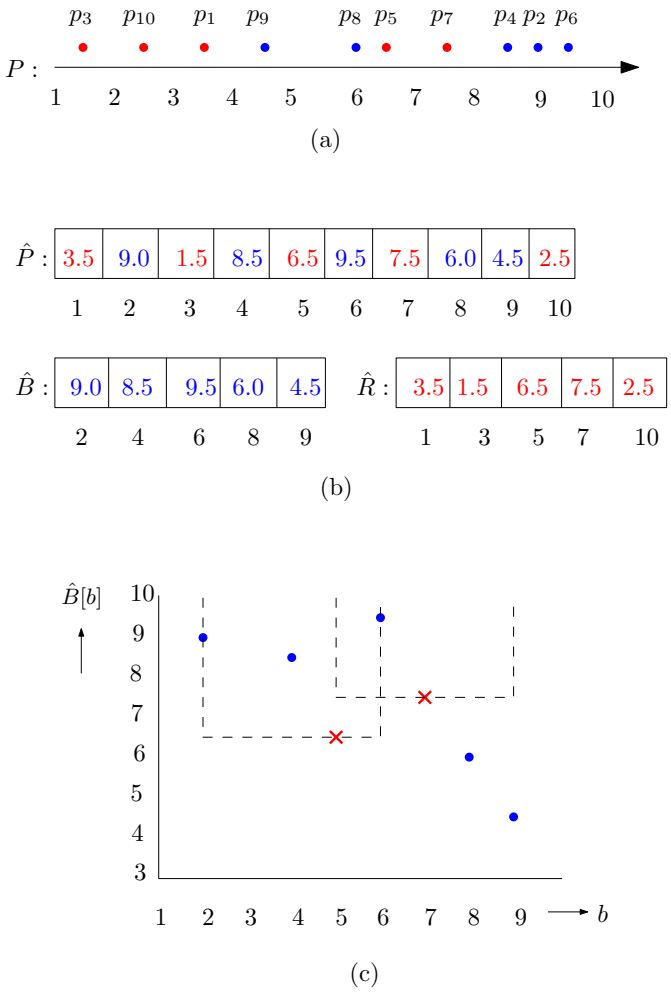


Figure 5.2: An example illustrating the 1-dimensional MLMP query. (a) Points $P = (p_1, \ldots, p_{10})$ are on the line. Each blue (respectively, red) point of P has an existential probability $\alpha_1 = 0.3$ (respectively, $\alpha_2 = 0.7$) associated with it. (b) Each array cell $\hat{P}[k]$ stores the x-coordinate value of the point p_k. $\hat{B}$ and $\hat{R}$ are respectively the arrays for all blue and red points. (c) Each blue point p_b is mapped to a 2-dimensional point $(b, \hat{B}[b])$. Given a query interval $[2, 6]$ we first find the red point p_5 ($\mathcal{R}_{max}$) with the maximum value 6.5 ($\mathcal{V}_{max}$). Now we query the range tree with $[2, 6] \times [6.5, +\infty)$ and count that there are three blue points p_2, p_4 and p_6 in the query range with values larger than 6.5. Since $0.7 \cdot (1 - 0.3)^3 = 0.24 < 0.3$, we report that a blue point p_6 (with value 9.5) to be the MLMP in $P_{2,6}$ with probability 0.3. Similarly, we can show that for a query interval $[5, 9]$, point p_7 is the $\mathcal{R}_{max}$ with value $\mathcal{V}_{max} = 7.5$. There is exactly on blue point p_6 in the query range $[5, 9] \times [7.5, +\infty)$. Since $0.7 \cdot (1 - 0.3) = 0.49 > 0.3$, the red point p_7 is the MLMP in $P_{5,9}$ with probability 0.7.

Points are in 1-dimension. Let $\hat{P}[1..n]$ be an array such that for $k = 1$ to n, $\hat{P}[k] = p_{k,x}$, where $p_{k,x}$ is the x-coordinate of the point p_k. Figure 5.2 illustrates an example. Now given a query interval $[i, j]$ the problem of computing the most likely maximal point in $P_{i,j}$ reduces to the problem of computing the most likely maximum value in $\hat{P}[i], \ldots, \hat{P}[j]$.

Data structure: We can directly use the data structure of Agarwal et al.'s result for a set of uncertain points [5] to find the most likely maximum value in the range $[i, j]$ in $O(n^{1-t} + \log n)$ time using $O(n^{1+t})$ space for any $t \in [0, 1]$. Let the most likely maximum value be X. Then the point at position X is the most likely maximal point in $P_{i,j}$. However, we present a simpler data structure here.

Recall that we consider $(B, \alpha_1; R, \alpha_2)$ to be an instance of P, where B contains all the blue points of P and R contains all the red points of P. Let $\hat{B}$ be the array similar to $\hat{P}$ for all blue points and let $\hat{R}$ be the array similar to $\hat{P}$ for all red points. We build two separate 1-dimensional range maximum data structures on $\hat{B}$ and $\hat{R}$. Given a 1-dimensional array A with N entries, the range maximum (RM) query asks for the maximum element within a contiguous subarray of A. Linear time and space preprocessing algorithms are known for the 1-dimensional case that can answer queries in constant time (see for example [10]). Let RM_B be the 1-dimensional RM structure built on $\hat{B}$ and let RM_R be the 1-dimensional RM structure built on $\hat{R}$. Both RM_B and RM_R require $O(n)$ space in total. In addition, we map each blue point p_b in $\hat{B}$ to a 2-dimensional point $(b, \hat{B}[b])$, where b is the position of p_b in P. Suppose $\bar{B}$ is the set of all 2-dimensional points obtained in this way. Now we build a 2-dimensional range tree $T_{\bar{B}}$ on these points. Total space and time required for constructing this range tree is $O(n \log n)$ [11].

Now we discuss how we answer a query. Given a query interval $[i, j]$ in $\hat{P}$, we first search RM_R data structure to find the maximum value for a red point in the range $[i, j]$. Let this point be $\mathcal{R}_{max}$ with the value $\mathcal{V}_{max}$. Then we search $T_{\bar{B}}$ with a three sided range query $[i, j] \times [\mathcal{V}_{max}, +\infty)$ and count the total number of blue points in this range. Let this number be m. That is, these are the m blue points that exist in the interval $[i, j]$ and have values greater than $\mathcal{V}_{max}$. In other words, these m blue points in $P_{i,j}$ have higher x-coordinate values than that of the red point $\mathcal{R}_{max}$. Then the probability that the point $\mathcal{R}_{max}$ is the most likely maximum point in $P_{i,j}$ is $\alpha_2(1-\alpha_1)^m$. Now we have the following two cases to consider.

- *Case 1*: If $\alpha_2(1-\alpha_1)^m > \alpha_1$ then the red point $\mathcal{R}_{max}$ at position max is the most likely maximal point with probability $\alpha_2(1-\alpha_1)^m$. We report this point as the answer to our query.
- *Case 2*: Otherwise, the most likely maximum point in $P_{i,j}$ is the blue point with maximum x-coordinate in $[i, j]$. To find this blue point with the maximum value we search the range maximum data structure RM_B and report the output point to be the maximum point with probability α_1.

RM_R and RM_B can answer queries in $O(1)$ time. Range counting query can be answered in $O(\log n)$ time. So the total query time is bounded by $O(\log n)$. Therefore we obtain the following theorem.

THEOREM 5.2. *Let α_1 and α_2 be constants with $0 < \alpha_1 < \alpha_2 < 1$. Given an instance of $MLMP(B, \alpha_1; R, \alpha_2)$ in 1-dimension, $P = B \cup R$ can be preprocessed into a data structure of size $O(n \log n)$ in $O(n \log n)$ time such that given a query interval $[i, j]$, with $1 \leqslant i < j \leqslant n$, it can report the most likely maximal point in $P_{i,j}$ in $O(\log n)$ time.*

Now we present an efficient data structure that can report the window query Q for the most likely maximal point, where the points $(B, \alpha_1; R, \alpha_2)$ are in $\mathbb{R}^d$, with $d \geqslant 1$, and Q: *Given a query interval* $q = [i, j]$ *with* $1 \leqslant i < j \leqslant n$, *report the most likely maximal point in* $P_{i,j}$.

Points are in d***-dimension.*** Let $A[1 \cdots n]$ be an array such that $A[k] = p_k$ for all $k = 1, \ldots, n$. We assume that all n points have distinct coordinates in all dimensions. Let $D(p; i, j)$ be the number of points in $\mathcal{D}(p)$ in $P_{i,j}$, i.e.,

$$D(p; i, j) = |\mathcal{D}(p) \cap \{p_i, p_{i+1}, \cdots, p_j\}|.$$

By Lemma 5.1, we have to consider two cases for answering the most likely maximal point query. Either a blue point that is on the maximal layer of all points in $P_{i,j}$ can be the most likely maximal point in $P_{i,j}$ with probability α_1 or a red point in $R_{i,j}$ with minimum m numbers of blue points to its $\mathcal{D}$-region can be the most likely maximal point in $P_{i,j}$ with probability $\alpha_2(1-\alpha_1)^m$. Thus we report the answer to query Q as $\max\{\alpha_1, \alpha_2(1-\alpha_1)^m\}$.

Now we present solutions for the case where the most likely maximal point is a red point. Observe that if the answer to query Q is a red point with probability $\alpha_2(1-\alpha_1)^m \geqslant \alpha_1$ then $m \leqslant \lfloor \log(\alpha_2/\alpha_1)/\log(1/1-\alpha_1) \rfloor$. Therefore, for a fixed integer λ, with $0 \leqslant \lambda \leqslant n$, we ask query Q' as follows. *Given* ℓ *and* r *with* $1 \leqslant \ell \leqslant r \leqslant n$, *is there a red point* p *in* $A[\ell, r]$ *for which* $D(p; \ell, r) \leqslant \lambda$*?*

We first solve two simpler queries.

1. Q'_1: *For fixed indices* ℓ *and* k *with* $1 \leqslant \ell \leqslant k \leqslant n$, *given an integer* r *with* $r \geqslant k$, *is there a red point* p *in* $A[\ell, \cdots, k]$ *for which* $D(p; \ell, r) \leqslant \lambda$*?*

2. Q'_2: *For fixed indices* k *and* r *with* $1 \leqslant k \leqslant r \leqslant n$, *given an integer* ℓ *with* $\ell \leqslant k$, *is there a red point* p *in* $A[k, \cdots, r]$ *for which* $D(p; \ell, r) \leqslant \lambda$*?*

Solution for Q'_1**:** Observe that if p is a red point in $A[\ell \cdots k]$ for which $D(p; \ell, k) \geqslant \lambda + 1$, then p does not satisfy the query Q'_1. Therefore, let p be a red point in $A[\ell \cdots k]$ for which $D(p; \ell, k) \leqslant \lambda$. We define

$$r_p = \max\{r : r \leqslant n \text{ and } D(p; \ell, r) \leqslant \lambda\}$$

i.e., for a fixed ℓ, $[\ell, r_p]$ is the largest query interval for which point p satisfies the query. Note that, we assume here the maximum of an empty set is $\max\{\} = -\infty$. We also define the following.

$$R_{\ell k} = \max\{r_p : p \text{ is a red point in } A[\ell \cdots k] \text{ and } D(p; \ell, k) \leqslant \lambda\}$$

We answer the query as follows. Given an integer r with $r \geqslant k$, if $r \leqslant R_{\ell k}$ then return *yes*, otherwise return *no*. Since we only store the value of $R_{\ell k}$ for fixed indices ℓ and k, the total space required is $O(1)$. We answer the query in $O(1)$ time.

Solution for Q'_2**:** The solution is symmetric to the one discussed above for Q'_1. Let p be a red point in $A[k \cdots r]$ for which $D(p; k, r) \leqslant \lambda$. We define

$$\ell_p = \min\{\ell : \ell \geqslant 1 \text{ and } D(p; \ell, r) \leqslant \lambda\}$$

We assume that minimum of an empty set $\min\{\} = \infty$. We define

Algorithm 10: Compute Log Values

```
Input : An array L[1...n].
Initialize ℓ = 0 and x = 1.                    //x = 2^ℓ.
while x ⩽ n do
    for i = x to min(2x − 1, n) do
        Set L[i] = ℓ.                          //L[i] = ⌊log i⌋
    Set x = 2x.
    Set ℓ = ℓ + 1.                             // x = 2^ℓ
```

$$L_{kr} = \min\{\ell_p : p \text{ is a red point in } A[k \cdots r] \text{ and } D(p; k, r) \leqslant \lambda\}$$

We answer the query as follows. Given an integer ℓ with $\ell \leqslant k$, if $L_{kr} \leqslant \ell$ then return *yes*, otherwise return *no*. For the similar reasoning, both the space and the query bound is $O(1)$.

Answering Q': We now describe how we answer query Q' using results of queries Q'_1 and Q'_2. For all $\ell = 1, 2, \cdots, n$, we store the solutions for Q'_1 for intervals of width $j = 0, 1, \cdots, \lfloor \log(n - \ell + 1) \rfloor$. More formally, we store the following.

```
for ℓ = 1, 2, ··· , n:
    for j = 0, 1, ··· , ⌊log(n − ℓ + 1)⌋:
        store R_{ℓ,ℓ+2^j−1}
```

Similarly, we also store the following.

```
for r = 1, 2, ··· , n:
    for j = 0, 1, ··· , ⌊log r⌋:
        store L_{r−2^j+1,r}
```

For every point in $A[1 \cdots n]$ we store at most $O(\log n)$ many L and R values. Therefore the total space required is $O(n \log n)$. For a given query with integers ℓ and r, with $1 \leqslant \ell \leqslant r \leqslant n$, we answer the query as follows.

1. If $\ell = r$: if p_ℓ is a red point then we return *yes*, else we return *no*.
2. If $\ell < r$: let $h = \lfloor \log(r - \ell) \rfloor$. Observe that $\{\ell, \ell + 1, \cdots, r\} = \{\ell, \cdots, \ell + 2^h - 1\} \cup \{r - 2^h + 1, \cdots, r\}$. If $r \leqslant R_{\ell,\ell+2^h-1}$ or $L_{r-2^h+1,r} \leqslant \ell$ then we return *yes*, else we return *no*.

The query Q' can be answered in $O(1)$ time.

We also present an alternative method for computing $\lfloor \log(r - \ell) \rfloor$ in $O(1)$ time. We precompute the values of $\lfloor \log(r - \ell) \rfloor$ and store them in an array so that the corresponding value can be fetched when required. Let $L[1 \ldots n]$ be an array that is initially empty. We initialize $\ell = 0$ and $x = 2^\ell$. See Algorithm 10 for details on how to compute $L[i]$ for $i = 1, \ldots, n$.

We summarize the result for querying Q' in the following theorem.

THEOREM 5.3. *Let α_1 and α_2 be constants with $0 < \alpha_1 < \alpha_2 < 1$, and let λ be a fixed integer with $0 \leqslant \lambda \leqslant n$. Suppose $A[1 \cdots n]$ is an array such that for $k = 1$ to n each $A[k]$ is either a blue point with existential probability α_1 or a red point with existential probability α_2. A can be preprocessed into a data structure of size $O(n \log n)$ such that given a query interval $[\ell, r]$ with $1 \leqslant \ell \leqslant r \leqslant n$, it can report whether there exists a red point in $A[\ell \cdots r]$ with at most λ blue points in its $\mathcal{D}$-region in $O(1)$ time.*

Answering Q**:** Finally given a query interval $[i, j]$, with $1 \leqslant i < j \leqslant n$, we answer query Q as follows. Recall that m is a constant with value at most $\beta = \lfloor \log(\alpha_2/\alpha_1)/\log(1/(1-\alpha_1)) \rfloor$. We build our data structures for solving query Q′ for $\lambda = 0, 1, \cdots, \beta$. Given a query interval $[i, j]$, we query these data structures sequentially starting with $\lambda = 0$ and onwards. If any of these queries with $0 \leqslant \lambda \leqslant \lfloor \log(\alpha_2/\alpha_1)/\log(1/1-\alpha_1) \rfloor$ returns *yes*, then we stop and report that a red point to be the most likely maximal point with probability $\alpha_2(1-\alpha_1)^\lambda$. Otherwise we report that a blue point is the most likely maximal point with probability α_1. Thus we obtain the following result.

THEOREM 5.4. *Let α_1 and α_2 be constants with $0 < \alpha_1 < \alpha_2 < 1$. Given an instance of* MLMP$(B, \alpha_1; R, \alpha_2)$, *$P = B \cup R$ can be preprocessed into a data structure of size $O(n \log n)$, such that given a query interval $[i, j]$ with $1 \leqslant i < j \leqslant n$, one can report the most likely maximal point in $P_{i,j}$ in $O(1)$ time.*

5.2.2 c-colored MLMP

In this section, we generalize our algorithm so that it can answer the c-colored MLMP query defined as follows. Let $\alpha_1, \ldots, \alpha_c$ are constants, with $0 < \alpha_1 < \alpha_2 < \ldots < \alpha_c < 1$. Let $P = P_1 \cup P_2 \cup \ldots \cup P_c$ be a set of n points in $\mathbb{R}^d$, for some fixed d. For $r = 1, 2, \ldots, c$, let all points in P_r be colored by the r-th color. Furthermore, for $r = 1, 2, \ldots, c$, each point in P_r is associated with probability α_r.

We first describe the extension for $c = 3$. Let α_1, α_2 and α_3 be constants with $0 < \alpha_1 < \alpha_2 < \alpha_3 < 1$. Let $P = B \cup R \cup G$ be a subsequence of B blue, R red and G green points, where each blue point has a probability α_1 associated with it, each red point has a probability α_2 associated with it and each green point has a probability α_3 associated with it. Let Z be a random subset of P where each blue (respectively, red and green) point of P is added to Z with probability α_1 (respectively, α_2 and α_3) independently of other points. For a query window $[i, j]$, we define $P_{i,j} = \{p_i, \ldots, p_j\}$, $B_{i,j} = B \cap P_{i,j}$, $R_{i,j} = R \cap P_{i,j}$ and $G_{i,j} = G \cap P_{i,j}$. Lemma 5.1 can be extended as follows.

LEMMA 5.5.

1. *A blue point is the answer to our query with probability α_1 if it is on the maximal layer of all points in $P_{i,j}$.*
2. *A red point p' is the answer to our query with probability $\alpha_2(1-\alpha_1)^m$ if it is on the maximal layer of $R_{i,j}$ such that at most m blue points in $B_{i,j}$ exist in $\mathcal{D}(p')$, with $0 \leqslant m \leqslant \lfloor \log(\alpha_2/\alpha_1)/\log(1/1-\alpha_1) \rfloor$, and no green points in $G_{i,j}$ exist in $\mathcal{D}(p')$.*
3. *A green point p'' is the answer to our query with probability $\alpha_3(1-\alpha_1)^k(1-\alpha_2)^\ell$ if it is on the maximal layer of $G_{i,j}$ such that at most k blue points in $B_{i,j}$ exist in $\mathcal{D}(p'')$, with $0 \leqslant k \leqslant \lfloor \log(\alpha_3/\alpha_1)/\log(1/(1-\alpha_1)) \rfloor$, and at most ℓ red points in $R_{i,j}$ exist in $\mathcal{D}(p'')$, with $0 \leqslant \ell \leqslant \lfloor \log(\alpha_3/\alpha_2)/\log(1/(1-\alpha_2)) \rfloor$.*

The proof is similar to that of Lemma 5.1 and hence omitted.

Answering queries: The final answer to our query in this setting can be obtained by computing the $\max\{\alpha_1, \alpha_2(1-\alpha_1)^m, \alpha_3(1-\alpha_1)^k(1-\alpha_2)^\ell\}$. Note that the data structures we presented in the previous section can answer for the first two cases of Lemma 5.5.

For case 3, similar to the analysis presented earlier, we can show that if $\ell = 0$ then $k \leqslant \lfloor \log(\alpha_3/\alpha_1)/\log(1/(1-\alpha_1)) \rfloor$ and if $k = 0$ then $\ell \leqslant \lfloor \log(\alpha_3/\alpha_2)/\log(1/(1-\alpha_2)) \rfloor$. Since $\alpha_1 < \alpha_2$, the maximum possible value of k is larger than the maximum possible value of ℓ. So, a green point p_g can be the most likely maximal point if there are at most $\lambda' = \lfloor \log(\alpha_3/\alpha_1)/\log(1/(1-\alpha_1)) \rfloor$ points in $\mathcal{D}(p_g)$. So we ask this query: '*Is there a green point* p_g *with at most* λ' *points in* $\mathcal{D}(p_g)$ *in* $P_{i,j}$*?*'. The data structure for Q' (see Section 5.2.1) can answer this query. Since we assume that all probability values α_1, α_2 and α_3 are constants, we obtain constant number of pairs (k, ℓ), with $k + \ell \leqslant \lambda'$, to query for case 3. Now we query the data structure for Q' with $\lambda = 0, 1, \ldots, \lambda'$. As mentioned above, the final answer for the 3-colored MLMP point can be answered by computing the maximum of all probabilities obtained for all the three cases in constant time. Total space required is $O(n \log n)$ and query can be answered in $O(1)$ time. We can generalize this to a constant number of colors. The result is summarized as follows.

THEOREM 5.6. *For a fixed integer* $c > 0$*, let* $\alpha_1, \ldots, \alpha_c$ *be constant real numbers with* $0 < \alpha_1 < \ldots < \alpha_c < 1$*. Let* $P = P_1 \cup P_2 \cup \ldots \cup P_c$ *be a sequence of* n *stochastic points such that for* $r = 1, \ldots, c$*, each* r*-colored point belongs to* P_r *and has a probability* α_r *associated with it.* P *can be preprocessed into a data structure so that given a query window* $[i, j]$*, with* $1 \leqslant i \leqslant j \leqslant n$*, it can report the* c*-colored MLMP in* $P_{i,j}$ *in* $O(1)$ *time using* $O(n \log n)$ *space.*

5.2.3 *Report* $\Pr(p_t$ *is on the Maximal Layer in* $P_{i,j})$

Let $P = (p_1, p_2, \ldots, p_n)$ be a sequence of n points in $\mathbb{R}^2$, where for $k = 1$ to n each point $p_k \in P$ has a probability $\gamma_k \in (0, 1]$ associated with it. In this section we solve the following problem: '*Given three integers* i, j *and* t*, with* $1 \leqslant i \leqslant t \leqslant j \leqslant n$*, report the probability that* p_t *is on the maximal layer in* $P_{i,j}$'.

For k=1 to n, let each point p_k be represented as (p_{k,x_1}, p_{k,x_2}), where x_1 and x_2 are the coordinates of p_k. We build a 3-dimensional range tree T on the points in P, where the first level is on the sequence of the points $1 \ldots n$ from left to right. At each of the canonical nodes of this tree we build two more levels based on, respectively, the x_1 and the x_2 coordinates of the points that are stored in this node. Now at each canonical node w of the third level of T, we store the probability that none of the points stored in the subtree T_w exist. We denote this value by $\rho(w) = \prod_{p_\ell \in T_w}(1-\gamma_\ell)$. The time and space required to build the range tree T is $O(n \log^2 n)$ [11].

Answering a query. Given i, j and t, with $1 \leqslant i \leqslant t \leqslant j \leqslant n$, we want to report the probability of p_t to be on the maximal layer in $P_{i,j}$. This requires us to find all points p_ℓ, with $i \leqslant \ell \leqslant j$, such that $p_\ell \in \mathcal{D}(p_t)$. Recall that $\mathcal{D}(p_t)$ is the set of points in $P_{i,j}$ that dominates p_t. We obtain the following lemma.

LEMMA 5.7. *Suppose* $1 \leqslant i \leqslant t \leqslant j \leqslant n$. *The point* p_t *is on the maximal layer in* $P_{i,j}$ *if* p_t *exists and none of the points* $p_\ell \in \mathcal{D}(p_t)$ *exist, where* $i \leqslant \ell \leqslant j$.

Therefore, the probability that p_t belongs to the maximal layer in $P_{i,j}$ is the value $\gamma_t \cdot \prod_{p_\ell \in \mathcal{D}(p_t)}(1-\gamma_\ell)$. To find all points in the $\mathcal{D}$-region of a given point $p_t = (p_{t,x1}, p_{t,x2})$, we query T with $[i,j] \times [p_{t,x1}, +\infty) \times [p_{t,x2}, +\infty)$. As the result, we obtain $O(\log^2 n)$ number of ρ values from $O(\log^2 n)$ canonical nodes that cover all points in $P_{i,j}$. We denote these ρ-values by $\rho_1, \ldots, \rho_{\log^2 n}$. We report $\gamma_t \cdot (\prod_{m=1}^{\log^2 n} \rho_m)$ as the probability that point p_t is on the maximal layer in $P_{i,j}$. Therefore, it takes $O(\log^2 n)$ time to answer a query. We can generalize this data structure for points in d-dimension, for any fixed $d \geqslant 2$.

THEOREM 5.8. *A sequence* $P = (p_1, p_2, \ldots, p_n)$ *of* n *points in* $\mathbb{R}^d$ *is given, where for* $k = 1$ *to* n *each point* $p_k \in P$ *has a probability* $\gamma_k \in (0,1]$ *associated with it.* P *can be preprocessed into a data structure of size* $O(n \log^d n)$ *in* $O(n \log^d n)$ *time such that given a query interval* $[i,j]$ *and an integer* t *with* $i \leqslant t \leqslant j$, *the probability that* p_t *to be on the maximal layer of* $P_{i,j}$ *can be reported in* $O(\log^d n)$ *time.*

5.3 MOST LIKELY COMMON ELEMENTS

Let S be an $m \times n$ matrix in which each entry $S[p,i]$ is a real number $\alpha_{pi} \in [0,1]$. We want to preprocess S into a data structure such that for query values p, q, i and j, with $1 \leqslant p \leqslant q \leqslant m$ and $1 \leqslant i \leqslant j \leqslant n$, it can decide if there exists some integer k (the most likely common element) for which $i \leqslant k \leqslant j$ and $\prod_{r=p}^{q} \alpha_{rk} \geqslant \tau$. We also present a data structure that can report all common elements in the query window.

Preprocessing: Let S' be the matrix of size $m \times n$. For $p = 1, \ldots, m$ and for $i = 1, \ldots, n$, we store $S'[p,i] = r - p + 1$, where $r = \max\{h : \prod_{\ell=p}^{h} \alpha_{\ell i} \geqslant \tau\}$, i.e., the length of the maximum subsequence of rows such that the product of the entries $S[\ell, i]$, with $p \leqslant \ell \leqslant h$, is at least τ. Figure 5.3 illustrates an example. For $i = 1, \ldots, n$, we scan each column i and find the values of $S'[p,i]$. See Algorithm 11 for the preprocessing step. For each row $p = 1, \ldots, m$, we build the range maximum data structure RM_p using linear space [10], so that for any query interval $[i,j]$, with $1 \leqslant i < j \leqslant n$, it can report the maximum value stored in $S'[p,i], \ldots, S'[p,j]$ in $O(1)$ time. The total preprocessing requires $O(mn)$ time.

To report all common elements that satisfy the condition, we build the data structure described as follows. For $p = 1, \ldots, m$ and for $i = 1, \ldots, n$, we map the values $S'[p,i]$ to a point $(i, S'[p,i]) \in \mathbb{R}^2$. Let X_p be the set of points obtained in this way for the row p of S'. For $p = 1, \ldots, m$ we create a priority search tree (PST) (see [27]) T_p for the points in X_p. Hence the first level of T_p is based on the values of i. For each canonical node of this tree, we build a second level on the sorted values of $S'[p,i]$. For each row p, this can be done in $O(n \log n)$ time and using $O(n)$ space. So the total space requirement becomes $O(mn)$.

Answering Queries: Given query indices p, q, i and j, we query the range maximum data structure RM_p with interval $[i,j]$. Suppose we obtain the maximum value and denote it by $\mathcal{M}_{p,i,j}$. If $\mathcal{M}_{p,i,j} \geqslant q - p + 1$ then there exists at least an element k in $[i,j]$ such that $\prod_{r=p}^{q} \alpha_{rk} \geqslant \tau$. Otherwise the answer is negative. To report the common elements for the same query indices i, j, p

	1	2	3	4	5	6	7
1	.2	.3	.7	.5	.6	.3	.8
2	.6	.1	.9	.2	.5	.8	.9
3	.8	.5	.6	.7	.9	.5	.5
4	.6	.1	.8	.5	.7	.5	.9
5	.7	.4	.4	.6	.9	.6	.8

(a)

	1	2	3	4	5	6	7
1	0	1	1	4	2	1	4
2	2	0	3	0	3	2	4
3	3	1	2	2	3	1	3
4	2	0	2	2	2	2	2
5	1	1	1	1	1	1	1

(b)

Figure 5.3: (a) $\mathcal{S}$ is a matrix of size 5×7 and $\tau = 0.3$. (b) The data structure $\mathcal{S}'$. An example illustrating the query with indices 1, 7, 1 and 3 (corresponding elements are blue in $\mathcal{S}$ and $\mathcal{S}'$). The range maximum data structure for row 1 of $\mathcal{S}'$ returns the maximum value 4, which is greater than $3 - 1 + 1 = 3$, so there exists some element (4 & 7) in $[1, 7]$ for which the answer is positive. Whereas, the query with indices 3, 5, 2 and 5 returns a negative result (highlighted with green). In row 2 of $\mathcal{S}'$ the maximum value $M_{(}(x)_{3,5}) = 3$, which is less than $5 - 2 + 1 = 4$.

Algorithm 11: Compute $\mathcal{S}'$

Input : A matrix $\mathcal{S}$ of size $m \times n$ and a fixed value $\tau \in [0, 1]$

```
for p = 1 to m do
    for i = 1 to n do
        Initialize S'[p, i] with value −1.
for i = 1 to n do
    Set p ← 1 and top ← 0.
    for ℓ = 1 to m do
        case ℓ = 1 : do
            if S[ℓ, i] < τ then
                Set S'[ℓ, i] = 0.
            else
                Set p = p ∗ S[ℓ, i].
                Set top = 1.
            Set ℓ++.
        case ℓ ⩾ 2 : do
            if p ∗ S[ℓ, i] ⩾ τ then
                Set p ← p ∗ S[ℓ, i] and ℓ++.
            else
                Set S'[top, i] = ℓ − top.
                Set p ← (p/S[top, i]).
                Set top++.
```

and q, we query T_p using a query range $[i, j] \times [q - p + 1, +\infty)$. The output of this query is the set of all those elements k in $\mathcal{S}[p, i], \ldots, \mathcal{S}[q, j]$ such that $\prod_{r=p}^{q} \alpha_{rk} \geqslant \tau$. Each query can be answered in $O(\log n + w)$ time, where w is the size of the output. Therefore, we obtain the following.

THEOREM 5.9. *Let $\mathcal{S}$ be a matrix of size $m \times n$, where m and n are positive integers. Suppose for $p = 1, \ldots, m$ and $i = 1, \ldots, n$, each element of $\mathcal{S}[p, i]$ is a real number $\alpha_{pi} \in (0, 1)$. Let τ be a fixed threshold in $[0, 1]$. $\mathcal{S}$ can be preprocessed into data structures of size $O(mn)$ so that given query indices i, j, p and q, with $1 \leqslant i < j \leqslant n$ and $1 \leqslant p < q \leqslant m$, it can answer the MLCE queries in $O(1)$ time and report these common elements in $O(\log n + w)$ time, where w is the size of the output.*

5.3.1 *A Special Case*

We consider a special case of MLCE problem where $\mathcal{S}$ is an $m \times n$ matrix and each entry $\mathcal{S}[p, i]$ is $\alpha_{pi} \in \{0, 1\}$ and $\tau = 1$. We want to preprocess $\mathcal{S}$ into a data structure such that for query values p, q, i and j, with $1 \leqslant p \leqslant q \leqslant m$ and $1 \leqslant i \leqslant j \leqslant n$, it can decide if there exists some integer k for which $i \leqslant k \leqslant j$ and $\prod_{r=p}^{q} \alpha_{rk} = \tau$.

We remove all entries of $\mathcal{S}$ that contain zero. Let N be the total number of elements in $\mathcal{S}$ that have value 1, that is $\sum_{p=1}^{m} \sum_{i=1}^{n} \alpha_{p,i} = N$. Figure 5.4 illustrates an example. After removal of all zeros the remaining elements in $\mathcal{S}$ are represented by x_k (see Figure 5.4(b)), where k represents its original column index.

Preprocessing step: For each $p = 1$ to m and $x_k \in \mathcal{S}[p]$, we define $\beta_p(k) = r - p$, where r is the smallest integer such that $r \geqslant p + 1$ and $x_k \notin \mathcal{S}[r]$. Let $\mathcal{S}'$ be a matrix of the same size as $\mathcal{S}$. For each row p and for each $x_k \in \mathcal{S}[p]$, let $\mathcal{S}'[p]$ store the value of $\beta_p(k)$. As before, for each row p of $\mathcal{S}'[p]$ we build the range maximum data structure RM_p so that for any query interval $[i, j]$, with $1 \leqslant i < j \leqslant |\mathcal{S}'[p]|$, it can report the maximum value stored in $\mathcal{S}'[p, i], \ldots, \mathcal{S}'[p, j]$ in $O(1)$ time using linear space [10]. See Figure 5.4(c).

Algorithm for computing $\beta_p(k)$: For each element $k \in \{1, 2, \ldots, n\}$, we maintain an array X_k that contains the sorted indices (in the non-decreasing order) of the rows p such that $x_k \in \mathcal{S}[p]$. For $k = 1$ to n, we scan each element of the array X_k sequentially to find the maximum subsequence of sets that contain element x_k. Then for each of these sets we compute $\beta_p(k)$ according to Algorithm 12. Initialization of m empty lists $\mathcal{S}'[p]$, with $1 \leqslant p \leqslant m$, (lines 1 and 2) takes $O(N)$ time. From line 3 to 13, every element of lists $X_1 \ldots X_n$ is scanned exactly once to find the maximum subsequence of sets as discussed above. Therefore the total time required for this algorithm is $O(N)$. The total space required for $\mathcal{S}'$ is also $\sum_{k=1}^{n} |X_k| = N$. Thus the total space requirement is linear.

Answering queries: The query process is similar to the previous section. Since $\mathcal{S}'$ is not a matrix in this case, we use binary searches before querying for the range maximum value. For query indices p, q, i and j, we query $\mathcal{S}'[p]$ using the interval $[i, j]$ to find the range $x_{i'j'} = x_{i'}, \ldots, x_{j'}$ with the smallest index $i' \geqslant i$ and the largest index $j' \leqslant j$. We query the range maximum data structure RM_p for the maximum value in the range $x_{i'j'}$ and follow the same process. The query time is dominated by the binary searches that require $O(\log N)$ time.

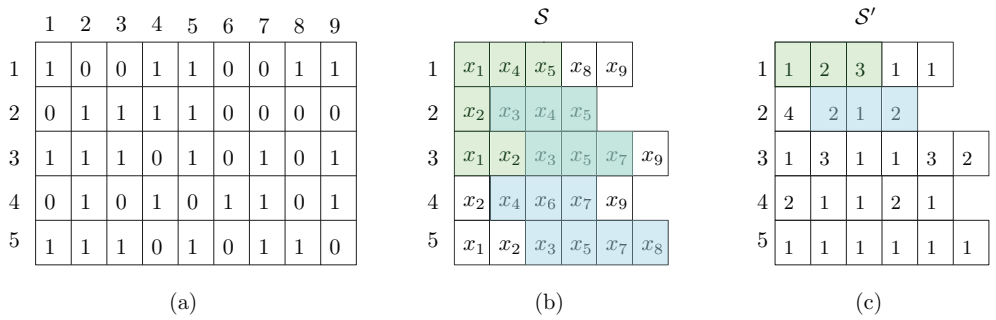


Figure 5.4: (a) A matrix $\mathcal{S}$ of size 5×9. (b) $\mathcal{S}$ after removing all zero's. (c) The data structure $\mathcal{S}'$. Suppose, we query $\mathcal{S}$ with query indices 1, 7, 1 and 3. All corresponding elements are highlighted with green. The query range $[1,7]$ in $\mathcal{S}'$ is also highlighted with green. We query the range maximum data structure in $\mathcal{S}'[1]$ for the maximum element within $[1,7]$. Since $M_{(}(x)_{1,7}) = 3$ is equal to $3-1+1=3$, we answer: Yes, there exists an element x_5 such that $\prod_{r=1}^{7} \alpha_{r,5} = 1$. Similarly the answer to the query with indices 3, 8, 2 and 5 (corresponding elements are similarly highlighted with blue in $\mathcal{S}$ and in $\mathcal{S}'$) is negative. The maximum value $M_{(}(x)_{3,8})$ in $\mathcal{S}'[2]$ is 2 that is less than $5-2+1=4$, so there does not exist any element in $[3,8]$ that satisfies the condition.

Algorithm 12: Compute $\beta_p(k)$

Input : A matrix $\mathcal{S}$ of size N

```
for ℓ = 1 to m do
    Initialize empty rows S'[ℓ] of size |S[ℓ]|
for k = 1 to n do
    for p = 1 to |X_k| do
        Let z ← X_k[p]
        Let top ← X_k[p]
        while X_k[p+1] == z+1 do
            Increase p++
            Increase z++
        for ℓ = top to z do
            Insert z − ℓ + 1 to the next empty cell of row S'[ℓ]
            Increase ℓ++
        Increase p++
```

THEOREM 5.10. *Let $\mathcal{S}$ be a matrix of size $m \times n$, where m and n are positive integers. Suppose for $p = 1, \ldots, m$ and $i = 1, \ldots, n$, each element of $\mathcal{S}[p,i]$ is a number $\alpha_{pi} \in \{0,1\}$ and $\tau = 1$. Let N be the total number of elements in $\mathcal{S}$ that have value 1. $\mathcal{S}$ can be preprocessed into data structures of size $O(N)$ so that given query indices i, j, p and q, with $1 \leqslant i < j \leqslant n$ and $1 \leqslant p < q \leqslant m$, it can answer the MLCE queries in $O(\log N)$ time and report these common elements in $O(\log N + w)$, where w is the size of the output.*

5.4 CONCLUSION

We present data structures to answer two types of window queries for stochastic input sequences. The first problem is a c-colored most likely maximal point problem (MLMP) where given an instance of $\mathrm{MLMP}(P_1, \alpha_1; P_2, \alpha_2; \ldots; P_c, \alpha_c)$, with $P = P_1 \cup \ldots \cup P_c$, and a pair of query indices our data structures can report the most likely maximal point within a query interval in P in $O(1)$ time using $O(n \log n)$ space, where n is the number of points in the input sequence. Secondly, we solve the most likely common element (MLCE) problem, where a matrix of size $m \times n$, with all entries being real numbers in $[0, 1]$ and a fixed real number τ in $[0, 1]$ are given. For query indices p, q, i and j, our data structure can decide if there exists some integer k for which $i \leqslant k \leqslant j$ and $\prod_{r=p}^{q} \alpha_{rk} \geqslant \tau$ in $O(1)$ time using $O(mn)$ space. The reporting version takes $O(\log n + w)$ time, where w is the size of the output.

The following are natural problems that require further explorations.

OPEN PROBLEM 6. *Given an instance of* $(B, \alpha_1; R, \alpha_2)$*, construct a data structure that can report the most likely maximal point in a given query interval in logarithmic query time using linear space.*

OPEN PROBLEM 7. *Given a sequence* $P = (p_1, \ldots, p_n)$ *of* n *stochastic points in* $\mathbb{R}^d$*, with* $d \geqslant 1$*, such that for* $k = 1 \ldots n$*, each point* p_k *exists with probability* $\gamma_k \in [0, 1]$*, report the most likely maximal point in a given query interval.*

OPEN PROBLEM 8. *Given a sequence of random subsets* $S_1, S_1, \ldots, S_m$ *of* $\mathcal{U} = \{1, \ldots, n\}$ *and query indices* p*,* q*,* i *and* j*, with* $1 \leqslant p \leqslant q \leqslant m$ *and* $1 \leqslant i \leqslant j \leqslant n$*, find the element* $k \in \{i, \ldots, j\}$ *that is most likely to be in* $S_p \cap S_{p+1} \cap \ldots, \cap S_q$.

BIBLIOGRAPHY

[1] Abdullah, A., Daruki, S., Phillips, J.M.: Range counting coresets for uncertain data. In: Symposuim on Computational Geometry 2013, SoCG '13, Rio de Janeiro, Brazil, June 17-20, 2013. pp. 223–232. ACM (2013), https://doi.org/10.1145/2462356.2462388

[2] Afshani, P., Agarwal, P.K., Arge, L., Larsen, K.G., Phillips, J.M.: (approximate) uncertain skylines. Theory Comput. Syst. 52(3), 342–366 (2013), https://doi.org/10.1007/s00224-012-9382-7

[3] Agarwal, P.K., Cheng, S., Tao, Y., Yi, K.: Indexing uncertain data. In: Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2009, June 19 - July 1, 2009, Providence, Rhode Island, USA. pp. 137–146 (2009), https://doi.org/10.1145/1559795.1559816

[4] Agarwal, P.K., Har-Peled, S., Suri, S., Yildiz, H., Zhang, W.: Convex hulls under uncertainty. Algorithmica 79(2), 340–367 (2017), https://doi.org/10.1007/s00453-016-0195-y

[5] Agarwal, P.K., Kumar, N., Sintos, S., Suri, S.: Range-max queries on uncertain data. J. Comput. Syst. Sci. 94, 118–134 (2018), https://doi.org/10.1016/j.jcss.2017.09.006

[6] Aggarwal, C.C. (ed.): Managing and Mining Sensor Data. Springer (2013), https://doi.org/10.1007/978-1-4614-6309-2

[7] Agrawal, A., Li, Y., Xue, J., Janardan, R.: The most-likely skyline problem for stochastic points. In: Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017, July 26-28, 2017, Carleton University, Ottawa, Ontario, Canada. pp. 78–83 (2017)

[8] Bannister, M.J., Devanny, W.E., Goodrich, M.T., Simons, J.A., Trott, L.: 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), http://www.cccg.ca/proceedings/2014/papers/paper02.pdf

[9] Bannister, M.J., DuBois, C., Eppstein, D., Smyth, P.: 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. pp. 856–864 (2013), https://doi.org/10.1137/1.9781611973105.61

[10] Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings. pp. 88–94 (2000), https://doi.org/10.1007/10719839_9

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

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

[13] Carmi, P., Chanchary, F., Maheshwari, A., Smid, M.: The most likely object to be seen through a window. Submitted to International Journal of Computational Geometry & Applications (2019)

[14] Chan, T.M., Pratt, S.: 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. LIPIcs, vol. 51, pp. 28:1–28:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016), https://doi.org/10.4230/LIPIcs.SoCG.2016.28

[15] Chanchary, F., Maheshwari, A.: Counting subgraphs in relational event graphs. In: WALCOM: Algorithms and Computation - 10th International Workshop, WALCOM 2016, Kathmandu, Nepal, March 29-31, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9627, pp. 194–206. Springer (2016), https://doi.org/10.1007/978-3-319-30139-6_16

[16] Chanchary, F., Maheshwari, A., Smid, M.H.M.: Querying relational event graphs using colored range searching data structures. In: Algorithms and Discrete Applied Mathematics -

Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10156, pp. 83–95. Springer (2017), https://doi.org/10.1007/978-3-319-53007-9_8

[17] Chanchary, F., Maheshwari, A., Smid, M.H.M.: 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. Lecture Notes in Computer Science, vol. 10743, pp. 199–213. Springer (2018), https://doi.org/10.1007/978-3-319-74180-2_17

[18] De Caluwe, R.: Fuzzy and uncertain object-oriented databases: concepts and models, vol. 13. World Scientific (1997)

[19] Goodchild, M.F.: Uncertainty in Remote Sensing and GIS. Edited by Giles M. Foody and Peter M. Atkinson (Chichester, UK: John Wiley, 2002). [Pp. xviii + 307 pages]. ISBN 0-470-84408-6. Hardback, vol. 18 (2004), https://doi.org/10.1080/13658810310001620861

[20] Held, M., Mitchell, J.S.B.: Triangulating input-constrained planar point sets. Inf. Process. Lett. 109(1), 54–56 (2008), https://doi.org/10.1016/j.ipl.2008.09.016

[21] Jørgensen, A., Löffler, M., Phillips, J.M.: Geometric computations on indecisive and uncertain points. CoRR abs/1205.0273 (2012), http://arxiv.org/abs/1205.0273

[22] Kamousi, P., Chan, T.M., Suri, S.: Stochastic minimum spanning trees in euclidean spaces. In: Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011. pp. 65–74. ACM (2011), https://doi.org/10.1145/1998196.1998206

[23] Kamousi, P., Chan, T.M., Suri, S.: Closest pair and the post office problem for stochastic points. Comput. Geom. 47(2), 214–223 (2014), https://doi.org/10.1016/j.comgeo.2012.10.010

[24] Li, J., Wang, H.: Range queries on uncertain data. Theor. Comput. Sci. 609, 32–48 (2016), https://doi.org/10.1016/j.tcs.2015.09.005

[25] Löffler, M., Snoeyink, J.: Delaunay triangulation of imprecise points in linear time after preprocessing. Comput. Geom. 43(3), 234–242 (2010), https://doi.org/10.1016/j.comgeo.2008.12.007

[26] MacKeown, P.K.: Stochastic Simulation in Physics. Springer-Verlag New York, Inc. (2001)

[27] McCreight, E.M.: Priority search trees. SIAM J. Comput. 14(2), 257–276 (1985), https://doi.org/10.1137/0214021

[28] Mena, R.H.: Book Review - J.B. Kadane, Principles of Uncertainty, Texts in Statistical Science Series, Chapman & Hall/CRC, 2011. pp. xxviii+475, ISBN 978-1-4398-6161-5, vol. 31 (2014), https://doi.org/10.1007/s00357-014-9158-7

[29] Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria,

September 23-27, 2007. pp. 15–26. ACM (2007), http://www.vldb.org/conf/2007/papers/research/p15-pei.pdf

[30] Suri, S., Verbeek, K., Yildiz, H.: On the most likely convex hull of uncertain points. In: Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings. pp. 791–802. Springer (2013), https://doi.org/10.1007/978-3-642-40450-4_67

[31] Vazirgiannis, M., Halkidi, M., Gunopulos, D.: Uncertainty Handling and Quality Assessment in Data Mining. Advanced Information and Knowledge Processing, Springer (2003), https://doi.org/10.1007/978-1-4471-0031-7

[32] Xue, J., Li, Y.: Colored stochastic dominance problems. CoRR abs/1612.06954 (2016), http://arxiv.org/abs/1612.06954

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of April 29, 2019 (`classicthesis` version 4.1).