On Reporting Durable Patterns in Temporal Proximity Graphs*

PANKAJ K. AGARWAL, Department of Computer Science, Duke University, USA
XIAO HU†, Cheriton School of Computer Science, University of Waterloo, Canada
STAVROS SINTOS, Department of Computer Science, University of Illinois at Chicago, USA
JUN YANG, Department of Computer Science, Duke University, USA

Finding patterns in graphs is a fundamental problem in databases and data mining. In many applications, graphs are *temporal* and evolve over time, so we are interested in finding *durable patterns*, such as triangles and paths, which persist over a long time. While there has been work on finding durable simple patterns, existing algorithms do not have provable guarantees and run in strictly super-linear time. The paper leverages the observation that many graphs arising in practice are naturally *proximity graphs* or can be approximated as such, where nodes are embedded as points in some high-dimensional space, and two nodes are connected by an edge if they are close to each other. We work with an implicit representation of the proximity graph, where nodes are additionally annotated by time intervals, and design near-linear-time algorithms for finding (approximately) durable patterns above a given durability threshold. We also consider an interactive setting where a client experiments with different durability thresholds in a sequence of queries; we show how to compute incremental changes to result patterns efficiently in time near-linear to the size of the changes.

CCS Concepts: • **Theory of computation → Data structures and algorithms for data management**.

Additional Key Words and Phrases: temporal graph, proximity graph, durability, durable pattern, doubling dimension, cover tree

ACM Reference Format:
Pankaj K. Agarwal, Xiao Hu, Stavros Sintos, and Jun Yang. 2024. On Reporting Durable Patterns in Temporal Proximity Graphs. *Proc. ACM Manag. Data* 2, 2 (PODS), Article 81 (May 2024), 26 pages. https://doi.org/10.1145/3651144

1 INTRODUCTION

Finding patterns in large graphs is a fundamental problem in databases and data mining. In many practical applications, graphs evolve over time, and we are often more interested in patterns that are "durable," i.e., persisting over a long time. Here are two examples of finding durable patterns in temporal graphs.

*This work was partially supported by NSF grants CCF-20-07556, CCF-22-23870, IIS-1814493, IIS-2008107, and by US-Israel BSF grant 2022131.
†This work was partially done while the author was visiting Simons Institute for the Theory of Computing.

Authors' addresses: Pankaj K. Agarwal, Department of Computer Science, Duke University, Durham, USA, pankaj@cs.duke.edu; Xiao Hu, Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada, xiaohu@uwaterloo.ca; Stavros Sintos, Department of Computer Science, University of Illinois at Chicago, Chicago, USA, stavros@uic.edu; Jun Yang, Department of Computer Science, Duke University, Durham, USA, junyang@cs.duke.edu.

ACM 2836-6573/2024/5-ART81
https://doi.org/10.1145/3651144

Example 1.1. Consider an online forum with social networking features, where users with similar profiles are connected as friends. Each user may be active on the forum only for a period of time during the day. We are interested in finding cliques of connected users who are simultaneously active for a sufficiently long time period. Such queries are useful to forum administrators who want to understand how the social network influences user interactions and leverage this knowledge to promote more interactions.

Example 1.2. Consider a co-authorship graph where two researchers are connected if they have written at least one paper together. Further suppose that researchers are each associated with a time period when they remain active in research. Besides researchers with direct co-authorship, we might be interested in pairs who co-authored with a set of common researchers over a long period of time. We would not be interested in researchers with a common co-author if the respective collaborations happened at distant times.

The problem of finding (durable) patterns, such as triangles, in general graphs is challenging: known (conditional) lower bounds suggest that it is unlikely to have near-linear algorithms [2, 5, 46]. However, many graphs that arise in practice are naturally *proximity graphs*, or can be approximated as such. In proximity graphs, nodes are embedded as points in some high-dimensional space, and two nodes are connected by an edge if they are close to each other (i.e., their distance is within some threshold). For example, social networks such as Example 1.1 can be embedded (with small error) in the space of user profiles with a low intrinsic dimension [51]. Similarly, in the co-authorship graph, where two authors nodes are connected if they have written at least m papers together for some $m \geq 1$, the nodes can also be embedded in a space with low intrinsic dimension [52]. Generally, for many graphs arising in a wide range of applications (e.g. social network, transportation network, Internet), there exist appropriate node embeddings that preserve the structures and shortest paths in the original graphs [51, 55, 56]. This observation enables us to leverage the properties of proximity graphs to develop efficient algorithms for finding patterns in such graphs, which overcome the hardness of the problem on arbitrary graphs.

This paper hence tackles the problem of finding durable patterns in temporal *proximity* graphs, for which we are not aware of efficient algorithms. For simplicity, we assume in this paper that the embedding of the graph is given – there are efficient algorithms for computing graph embeddings [12, 16, 38, 51, 54]. We work with an implicit representation of the proximity graphs – nodes represented as points and edges defined between pairs of points within a threshold distance in the embedding space. We never construct the graph itself explicitly. We design efficient algorithms whose running time depend on the number of nodes and the intrinsic dimension (*doubling dimension*) of the data. Our approach extends naturally to other classes of graphs including interval graphs, permutation graphs, and grid graphs. Next, we formally define the problems we study. Our notation is summarized in Table 1.

1.1 Problem Definitions

Let (P, ϕ) be a metric space over a set of n points $P \subset \mathbb{R}^d$, for some $d \geq 1$, and a metric ϕ. For a parameter $r > 0$, let $G_\phi(P, r) = (P, E)$ where $E = \{(p, q) \mid \phi(p, q) \leq r\}$ be a *proximity graph*, also called a *unit disk* graph. For simplicity, we assume $r = 1$ and let $G_\phi(P) = G_\phi(P, 1)$. Suppose a function I assigns each point $p \in P$ to a time interval called its *lifespan*, denoted $I_p = [I_p^-, I_p^+]$. We can interpret the lifespan of p as inserting p at time-stamp I_p^- and deleting it at time-stamp I_p^+. We use (P, ϕ, I) to refer to the *temporal* proximity graph, or the underlying metric space with points annotated with interval lifespans. For simplicity, all defined problems assume that the query pattern is triangle. As we point out in Section 1.2, all techniques are extended to more general

P	point set
n	$\lvert P\rvert$
ϕ	distance function
ρ	doubling dimension
ε	distance approximation
τ	durability parameter
$I_p = [I_p^-, I_p^+]$	lifespan (interval) of point p
T_τ	τ-durable triangles
T_τ^ε	τ-durable ε-triangles
K_τ^ε	τ-SUM durable ε-pairs
$K_{\tau,\kappa}^\varepsilon$	(τ,κ)-UNION durable ε-pairs
OUT	Output size

Table 1. Table of Notations.

patterns. For an interval I, we define $|I|$ as the length of I. If I is a set of intervals then $|I|$ is defined as the length of the union of intervals in I.

Durable triangles. A triplet $(p_1, p_2, p_3) \in P \times P \times P$ forms a *triangle* in $G_\phi(P)$ if $\phi(p_1, p_2)$, $\phi(p_2, p_3)$, $\phi(p_1, p_3) \leq 1$. We also introduce an approximate notion of triangles: for a parameter $\varepsilon > 0$, a triplet (p_1, p_2, p_3) forms an $(1+\varepsilon)$-*approximate triangle*, or ε-*triangle* for brevity, if $\phi(p_1, p_2)$, $\phi(p_2, p_3)$, $\phi(p_1, p_3) \leq 1+\varepsilon$. The *lifespan* of (p_1, p_2, p_3) is defined as $I(p_1, p_2, p_3) = I_{p_1} \cap I_{p_2} \cap I_{p_3}$. For a durability parameter $\tau > 0$, (p_1, p_2, p_3) is τ-*durable* if $|I(p_1, p_2, p_3)| \geq \tau$. Let $T_\tau, T_\tau^\varepsilon$ be the set of τ-durable triangles, and τ-durable ε-triangles respectively. Note that $T_\tau \subseteq T_\tau^\varepsilon$. Given a τ-durable triangle with three points, the point that *anchors* the triangle is the one whose lifespan starts the latest among the three. By convention, we will list the anchor first in the triplet; i.e., in a τ-durable triangle (p, q, s), we have $I_p^- \geq \max\{I_q^-, I_s^i\}$.

*Definition 1.3 (**DurableTriangle**).* Given (P, ϕ, I) and $\tau \geq 0$, it asks to report all τ-durable triangles (or ε-triangles).

Suppose we have embedded the social network in Example 1.1 as a proximity graph where nodes represent users. The goal is to find triplets (or generally cliques) of users who are simultaneously active on the forum.[1]

In some use cases, we do not have a clear choice of the durability parameter τ in mind, and we may want to explore with different settings. Supporting this mode of querying motivates the problem of incrementally reporting τ-durable triangles. Here, queries arrive in an online fashion, each specifying a different durability parameter $\tau_1, \tau_2, \ldots$. Instead of computing each query τ_{i+1} from scratch, we want to leverage the previous query result T_{τ_i} and only incrementally compute what is new. Note that every τ-durable triangle must also be τ'-durable for every $\tau' \leq \tau$. Therefore, if $\tau_{i+1} \geq \tau_i$, we have $T_{\tau_{i+1}} \subseteq T_{\tau_i}$ so we simply need to filter the old results to obtain new ones (assuming we remember results together with their lifespans). The more interesting case is when $\tau_{i+1} < \tau_i$, so $T_{\tau_{i+1}} \supseteq T_{\tau_i}$, and we need to incrementally report new results.

*Definition 1.4 (**IncrDurableTriangle**).* Given (P, ϕ, I) and $\tau_\prec > \tau > 0$, it asks to report all τ-durable triangles (or ε-triangles) that are not $\tau_\prec$-durable, along with their lifespans.

[1]While for simplicity of exposition we assume that each node has a single-interval lifespan, it is straightforward to extend our temporal model consider multiple-interval lifespans, with the complexities of our solutions in the following sections increased by a factor equal to the maximum number of intervals per lifespan.

Problem	Time complexity in $\tilde{O}(\cdot)$
DurableTriangle	$n\varepsilon^{-O(\rho)} + \text{OUT}$
IncrDurableTriangle	$\varepsilon^{-O(\rho)} \cdot \text{OUT}$
AggDurablePair–SUM	$\varepsilon^{-O(\rho)} \cdot (n + \text{OUT})$
AggDurablePair–UNION	$\kappa\varepsilon^{-O(\rho)} \cdot (n + \text{OUT})$

Table 2. Summary of our main results. Here, n is the input size, i.e., the number of points in P; ρ is the doubling dimension of (P, ϕ); τ is the durability parameter; OUT is the output size for the respective problem (different for each problem); ε is the approximation ratio; and k is the parameter used for (τ, κ)-UNION durability. In the complexities reported above, $\tilde{O}(\cdot)$ hides a polylog n factor, and the hidden constants in $O(\cdot)$ and $\tilde{O}(\cdot)$ may depend on ρ, which is assumed to be a constant.

Aggregate-durable pairs. Given a pair $(p_1, p_2) \in P \times P$, we consider the set U of nodes incident to both p_1 and p_2 and aggregate the lifespans of triplets (u, p_1, p_2). We call U the *witness* of (p_1, p_2). There are two natural ways of aggregating over U: SUM and UNION. For SUM, we aggregate by summing up the durabilities of triplet lifespans, i.e., $\text{AGG}(p_1, p_2, U) = \sum_{u \in U} |I(u, p_1, p_2)|$. For UNION, we aggregate by first taking the union of the triplet lifespans and then considering its length, i.e., $\text{AGG}(p_1, p_2, U) = |\bigcup_{u \in U} I(u, p_1, p_2)|$. Intuitively, SUM gives higher weights to time periods when multiple simultaneous connections exist, while UNION only cares about whether a period is covered at all by any connection. Given durability parameter $\tau > 0$, a pair $(p_1, p_2) \in P \times P$ is *τ-aggregate-durable* if $\phi(p_1, p_2) \le 1$ and $\text{AGG}(p_1, p_2, U) \ge \tau$ for $U = \{u \in P \mid \phi(p_1, u), \phi(p_2, u) \le 1\}$. We also define *$\tau$-aggregate-durable ε-pairs* by relaxing the distance thresholds for $\phi(p_1, p_2)$, $\phi(p_1, u)$, and $\phi(p_2, u)$ from 1 to $1+\varepsilon$. Let $K_\tau, K_\tau^\varepsilon$ be the set of all τ-aggregate-durable pairs, ε-pairs respectively. Notice that $K_\tau \subseteq K_\tau^\varepsilon$.

*Definition 1.5 (**AggDurablePair**).* Given (P, ϕ, I) and $\tau \ge 0$, it asks to report all τ-aggregate-durable pairs (or ε-pairs).

Suppose we have embedded the co-authorship graph in Example 1.2 as a proximity graph where nodes represent authors. The goal is to find pairs of coauthors p_i, p_j who have collaborated sufficiently with various others, either in terms of total time over all collaborators (SUM), or over a large portion of p_i and p_j's shared active lifespan (UNION).

1.2 Our Results and Approach

We present algorithms for ε-approximate versions of all three problems, whose time complexity are summarized in Table 2. In all cases, we report all durable triangles along with some durable ε-triangles. The running time is always near-linear in terms of the input and output size, which is almost the best one could hope for. Our solutions leverage the observation that proximity graphs in practice often have bounded *spread* Δ and *doubling dimension* ρ — Section 2 further reviews these concepts and the associated assumptions. The complexities in Table 2 assume spread to be $n^{O(1)}$ (hence Δ is omitted) and doubling dimension to be constant, but our algorithms also work for more general cases.

For **DurableTriangle (Section 3)**, our main approach is to construct a hierarchical space decomposition consisting of a canonical set of balls, via a *cover tree*. We use the canonical set of balls to obtain a compact, implicit representation of points within unit distance from each point, and then use an *interval tree* along with auxiliary data structures to report durable triangles in linear time. The algorithm runs in $\tilde{O}(n\varepsilon^{-O(\rho)} + \text{OUT})$ time, where $\text{OUT} \in [|T_\tau|, |T_\tau^\varepsilon|]$ is the result size. (The $\tilde{O}(\cdot)$ notation hides polylogarithmic factors). For the ℓ_∞ metric, this result can be improved to $\tilde{O}(n + |T_\tau|)$.

Moreover, our data structures can be extended to support delay-guaranteed enumeration as well as dynamic settings where nodes are inserted or deleted according to their lifespan.

For **IncrDurableTriangle (Section 4)**, to support incremental computation of queries arriving in an online setting, we additionally maintain an activation threshold for each point with respect to different durability parameters. In more detail, for each durability parameter τ, we design an oracle that can efficiently find the largest value $\beta < \tau$ such that p participates in a β-durable triangle that is not τ-durable, which is key to achieve near-linear time complexity. Our algorithm constructs an $\tilde{O}(n)$-size data structure in $\tilde{O}(n\varepsilon^{-O(\rho)})$ time, such that given the previous query parameter $\tau_<$ and current query parameter $\tau < \tau_<$, it can report the delta results in $\tilde{O}(\varepsilon^{-O(\rho)} \cdot \mathsf{OUT})$ time, where $\mathsf{OUT} \in \left[|I_\tau - I_{\tau_<}|, |I_\tau^\varepsilon - I_{\tau_<}^\varepsilon|\right]$ is the delta result size. Specifically, for the ℓ_∞ metric, IncrDurableTriangle can be solved exactly in $\tilde{O}(|I_{\tau_{i+1}} \setminus I_{\tau_i}|)$ time.

For **AggDurablePair (Section 5)**, recall that the problem requires aggregating lifespans over witness set U. We build auxiliary data structures to compute the sum or union of intervals intersecting any given interval. Additionally, we identify a special ordering of P such that only a bounded number of pairs that are not aggregate-durable will be visited, so the linear-time complexity can be guaranteed. For the SUM version of the problem, we present an $\tilde{O}((n + \mathsf{OUT}) \cdot \varepsilon^{-O(\rho)})$-time algorithm, where $\mathsf{OUT} \in \left[K_\tau, K_\tau^\varepsilon\right]$ is the output size. The UNION version is more challenging because of the inherent hardness of computing the union of intervals that intersect a query interval. However, as shown in Section 5.2, we can still get an near-linear-time and output-sensitive algorithm that reports all τ-UNION-durable pairs along with some $(1 - 1/e)\tau$-UNION-durable ε-pairs.

Extensions. Our algorithms also work for every ℓ_α-metric[2] or metric with bounded expansion constant[3]. Moreover, all our results for reporting triangles can be extended to reporting cliques, paths, and star patterns of constant size. See details in Appendix C.

Connection with triangle listing algorithms in general graphs. Consider simple directed or undirected graphs with n vertices and m edges. The trivial algorithm by listing all triples of vertices runs in $O(n^3)$ time. This is worst-case optimal in terms of n, since a dense graph may contain $\Theta(n^3)$ triangles. A graph with m edges contains $\Theta(m^{3/2})$ triangles. It has been shown that all triangles in a graph of m edges can be enumerated in $\tilde{O}(m^{3/2})$ time [35, 43, 50]. This is also worst-case optimal, since a graph of m edges may contain $\Theta(m^{3/2})$ triangles. Later, output-sensitive algorithms for listing triangles were developed using fast matrix multiplication, which run in $\tilde{O}\left(n^\omega + n^{\frac{3(\omega-1)}{5-\omega}} \cdot \mathsf{OUT}^{\frac{2(3-\omega)}{5-\omega}}\right)$ or $\tilde{O}\left(m^{\frac{2\omega}{\omega+1}} + m^{\frac{3(\omega-1)}{\omega+1}} \cdot \mathsf{OUT}^{\frac{3-\omega}{\omega+1}}\right)$ time, where $O(n^\omega)$ is the running time of $n \times n$ matrix multiplication and OUT is the number of triangles in the graph [9]. In contrast, it has been shown [46] that listing m triangles in a graph of m edges requires $m^{4/3-o(1)}$ time, assuming the 3SUM conjecture[4]. A careful inspection of this lower bound construction reveals that listing $n^{3/2}$ triangles in a graph of n vertices requires $n^{2-o(1)}$ time, assuming the 3SUM conjecture. These lower bounds together rule out the possibility of listing triangles in general graphs within $O(m + n + \mathsf{OUT})$ time, unless the 3SUM conjecture is refuted.

Existing techniques for listing triangles in general graphs do not yield efficient algorithms for our setting and several new ideas are needed to obtain the results of this paper. First, most traditional techniques for listing triangles do not handle temporal constraints on vertices or edges.

---

[2]If ϕ is the *ℓ_α-metric* then $\phi(p, q) = \left(\sum_{j=1}^d |p_j - q_j|^\alpha\right)^{1/\alpha}$, where p_j, q_j are the j-th coordinates of points p and q, respectively.

[3]A metric space (P, ϕ) has *expansion constant* D if D is the smallest value such that for every $p \in P$ and $r > 0$ $|P \cap \mathcal{B}(p, 2r)| \leq D \cdot |P \cap \mathcal{B}(p, r)|$, where $\mathcal{B}(p, r)$ is the ball with center p and radius r.

[4]The 3SUM conjecture states that: Given three sets A, B, C of n elements, any algorithm requires $n^{2-o(1)}$ time to determine whether there exists a triple $(a, b, c) \in A \times B \times C$ such that $a + b + c = 0$.

Recently, efficient algorithms for durable–join with temporal constraints on edges are proposed in [33]. However, their algorithm requires $\Omega(m^{3/2})$ time for listing durable triangles, even if the number of durable triangles is much smaller. Even without temporal constraints, all the worst-case optimal join algorithms run in super-linear time, in terms of n and OUT, for listing triangles in proximity graphs. Furthermore, in our setting, the input is an implicit representation of a proximity graph (P, ϕ). To feed (P, ϕ) as input to these algorithm, the number of edges m can be quadratic in terms of $|P|$, which already requires $\Omega(n^2)$ time for processing the input, not to mention the time for identifying triangles. Our algorithms use novel geometric data structures to identify all triangles that a point belongs to in time which is linear (ignoring $\log n$ factors) to both input size n and output size OUT. Finally, the known algorithms do not handle the incremental or the aggregate versions of our problem, and we need completely new techniques to further exploit the structure of proximity graphs.

2 PRELIMINARIES

We start by reviewing some basic concepts and data structures. Building on the basic data structures, we introduce an oracle that will be frequently used by our algorithms in the ensuing sections.

2.1 Basic concepts and data structures

Spread. The *spread* of a set P under distance metric ϕ is the ratio of the maximum and minimum pairwise distance in P. For many data sets that arise in practice, the spread is polynomially bounded in n, and this assumption is commonly made in machine learning and data analysis [8, 10, 17, 37].

Doubling dimension. For $x \in \mathbb{R}^d$ and $r \geq 0$, let $\mathcal{B}(x, r) = \{y \in \mathbb{R}^d \mid \phi(x, y) \leq r\}$ denote the ball (under the metric ϕ) centered at point x with radius r. A metric space (P, ϕ) has *doubling dimension* ρ if for every $p \in P$ and $r > 0$, $\mathcal{B}(p, r) \cap P$ can be covered by the union of at most 2^ρ balls of radius $r/2$. For every $\alpha > 0$, let ℓ_α be the α norm. The metric space (P, ℓ_α) has doubling dimension d for every $P \subset \mathbb{R}^d$, but for specific $P \subset \mathbb{R}^d$, the doubling dimension can be much smaller—e.g., points in 3d lying on a 2-dimensional plane or sphere has doubling dimension 2. Doubling dimensions and their variants are popular approaches for measuring the intrinsic dimension of a data set in high dimension; see, e.g., [11, 24, 47, 47]. It has been widely shown that graphs arising in practice have low doubling dimension [18, 25, 42, 49, 51]. Empirical studies in these papers and other sources (e.g., [1]) show that the doubling dimension of router graphs, internet latency graphs, citation graphs, and movie database graphs are less than 15.

Interval tree. Let $\mathcal{I}$ be a set of intervals. An interval tree [41] is a tree-based data structure that can find intersections of a query interval I with the set of intervals $\mathcal{I}$ stored in the interval tree. For example, it can report or count the number of intervals in $\mathcal{I}$ intersected by I visiting only $O(\log n)$ nodes. It has $O(n)$ space and it can be constructed in $O(n \log n)$ time.

Cover tree. A cover tree $\mathcal{T}$ is a tree-based data structure where each node u of $\mathcal{T}$ is associated with a representative point $\text{Rep}_u \in P$ and a ball $\mathcal{B}_u$. Each node belongs to an integer-numbered level; if a node u is at level i then its children are at level $i - 1$. Let C_i be the set of balls associated with nodes at level i. The radius of each ball $\mathcal{B}_u$ at level i is 2^i (notice that our definition allows level numbers to be positive or negative). Each point $p \in P$ is stored in one of the leaf nodes. The root consists of a ball that covers the entire data set and its representative point is any point in P. A cover tree satisfies the following constraints:

- **(Nesting)** If there is a node u at level i with a representative point $\text{Rep}_u \in P$, then Rep_u is also a representative point in a node at level $i - 1$.
- **(Covering)** For every representative point Rep_u at level $i - 1$, there exists at least one representative Rep_v at level i such that $\phi(\text{Rep}_v, \text{Rep}_u) < 2^i$. We designate v as the parent of u.

- **(Separation)** For every u, v at level i, $\phi(\text{Rep}_u, \text{Rep}_v) > 2^i$.

Traditionally, a cover tree is used mostly for approximate nearest-neighbor queries [8, 31]. We modify the construction of the cover tree to use it for *ball-reporting queries* in bounded doubling spaces. Given a point p, let $\mathcal{B}(p, r) = \{x \in \mathbb{R}^d \mid \phi(x, p) \leq r\}$ be a ball with radius r centered at p, and let $\mathcal{B}(p) := \mathcal{B}(p, 1)$. Given a query point p, the goal is to report $B(p) \cap P$ efficiently. We modify the cover tree to answer ball-reporting queries approximately. In each node u of the cover tree, we (implicitly) store P_u, i.e., the points that lie in the leaf nodes of the subtree rooted at u. Let p be a query point. We find a set of nodes in the cover tree whose associated balls entirely cover $\mathcal{B}(p)$ and might cover some region outside $\mathcal{B}(p)$ within distance $(1 + \varepsilon)$ from the center of $\mathcal{B}$. The set of nodes we find in the query procedure are called *canonical nodes*, their corresponding balls are called *canonical balls*, and the subsets of points stored in the canonical balls are called *canonical subsets*. In the end, we report all points stored in the canonical nodes. More formally, in Appendix A, we show how to construct a data structure with space $O(n)$ in $O(n \log n)$ time, while achieving the following guarantees when the spread is bounded. For a query point $q \in \mathbb{R}^d$, in $O(\log n + \varepsilon^{-O(\rho)})$ time, it returns a set of $O(\varepsilon^{-O(\rho)})$ canonical balls (corresponding to nodes in the modified cover tree) of diameter no more than ε, possibly intersecting, such that each point of $\mathcal{B}(q) \cap P$ belongs to a unique canonical ball. Each canonical ball may contain some points of $\mathcal{B}(q, 1 + \varepsilon) \cap P$.

2.2 Durable ball query

In this subsection, we describe an extension of the ball-reporting query that will be frequently used by our algorithms. Given (P, ϕ, I), $\tau > 0$, and a point p with interval I_p, a *τ-durable ball query* finds all points $q \in P$ such that $\phi(p, q) \leq 1$, $|I_p \cap I_q| \geq \tau$, and $I_p^- \in I_q$. Answering such a query exactly is inherently expensive even in the Euclidean space, since a near-linear space data structure has $\Omega(n^{1-1/d} + \text{OUT})$ query time [13], where OUT is the output size. If we use such a data structure for our problem in metrics with bounded doubling dimension, it would lead to a near-quadratic time algorithm for the DurableTriangleproblem. Instead, we consider the following relaxed version:

Definition 2.1 (ε-approximate τ-durable ball query). Given (P, ϕ, I), $\tau \geq 0$, $\varepsilon \in (0, 1)$, and a point p with interval I_p, find a subset $Q \subseteq P$ of points such that $\mathcal{B}(p) \cap P \subseteq Q \subseteq \mathcal{B}(p, 1 + \varepsilon) \cap P$, and for every $q \in Q$, $|I_p \cap I_q| \geq \tau$ and $I_p^- \in I_q$.

We note that the condition $I_p^- \in I_q$ is needed to avoid reporting duplicate results, as we will see in the next sections.

Data structure. Intuitively, we use a multi-level data structure $\mathcal{D}$ to handle this query. At the first level, we construct a cover tree CT on P to find a small number of canonical nodes that contain the points of $\mathcal{B}(p) \cap P$ (but may also contain some point of $\mathcal{B}(p, 1 + \varepsilon) \cap P$). At each node u of the cover tree, we construct an interval tree IT_u over the temporal intervals of P_u. Using the cover tree along with interval trees, we can find a set of $O(\varepsilon^{-d})$ canonical nodes that contain all points q within distance 1 from p and $I_p^- \in I_q$, but they may also contain points within distance $1 + \varepsilon$ from p. $\mathcal{D}$ uses $O(n \log n)$ space and can be constructed in $O(n \log^2 n)$ time.

Let $\text{durableBallQ}(p, \tau, \varepsilon)$ denote the query procedure of $\mathcal{D}$ with parameters p, τ, ε. It answers the query as follows:

- **(step 1)** We query CT with point p and radius 1, and obtain a set of canonical nodes $C = \{u_1, u_2, \ldots, u_k\}$ for $k = O(\varepsilon^{-O(\rho)})$. From Appendix A, each node in C corresponds to a ball with diameter no more than ε. For each $u_j \in C$, $\phi(p, \text{Rep}_j) \leq 1 + \varepsilon/2$.[5]
- **(step 2)** For each canonical node $u_j \in C$, we query IT_{u_j} with I_p^-, and obtain all $q \in \mathsf{IT}_u$ such that $I_q^- + \tau \leq I_p^- + \tau \leq I_q^+$.

[5]For simplicity, we denote the representative point of a node u_i by Rep_i.

Algorithm 1: ReportTriangle($\mathcal{D}, p, \tau, \varepsilon$)

```
C_p : {C_{p,1}, C_{p,2}, ..., C_{p,k}} ← durableBallQ(p, τ, ε/2), with Rep_i denoting the representative
  point of the ball for C_{p,i};
foreach j ∈ [k] do
    foreach q, s ∈ C_{p,j} where q precedes s do
        report (p, q, s);
foreach i, j ∈ [k] where i < j do
    if φ(Rep_i, Rep_j) ≤ 1 + ε/2 then
        foreach (q, s) ∈ C_{p,i} × C_{p,j} do
            report (p, q, s);
```

In the end, durableBallQ returns $O(k)$ disjoint result point sets, whose union is the answer to the ε-approximate τ-durable ball query. The grouping of result points into subsets and the implicit representation of these subsets is an important feature of durableBallQ that we shall exploit in later sections. Note that durableBallQ might return a point q such that $\phi(p, q) > 1$, but $\phi(p, q) \leq 1 + \varepsilon$ always holds. Together, we obtain:

Lemma 2.2. *Given a set P of n points, a data structure can be built in $O(n \log^2 n)$ time with $O(n \log n)$ space, that supports an ε-approximate τ-durable ball query, computing a family of $O(\varepsilon^{-d})$ canonical subsets in $O(\varepsilon^{-d} \log n)$ time.*

Extended data structure with refined result partitioning. We define the more involved query procedure durableBallQ$'(p, \tau, \tau', \varepsilon)$, which will be used by our algorithms for IncrDurableTriangle in Section 4. The goal is to return a subset $Q \subseteq P$ such that $\mathcal{B}(q) \cap P \subseteq Q \subseteq \mathcal{B}(q, 1 + \varepsilon) \cap P$, and for every $q \in Q$, $I_q^- + \tau \leq I_p^- + \tau \leq I_q^+$ (just as for durableBallQ), with the additional constraint that $I_q^+ \geq I_p^- + \tau'$. Let $\mathcal{D}'$ be the extended version of $\mathcal{D}$ to answer durableBallQ$'$. It consists of a cover tree along with two levels of interval trees, one to handle the first linear constraint, and the second to handle the additional linear constraint. The space, construction time and query time of $\mathcal{D}'$ are increased only by a $\log n$ factor compared with $\mathcal{D}$.

3 REPORTING DURABLE TRIANGLES

This section describes our near-linear time algorithm for the ε-approximate DurableTriangle problem. As mentioned, our algorithm works for every general metric with constant doubling dimension. In the full version of the paper [4] we show how to solve the problem exactly for ℓ_∞ metric.

High-level Idea. We visit each point $p \in P$ with $|I_p| \geq \tau$ (a prerequisite for p to be in a τ-durable triangle), and report all τ-durable triangles that p anchors. To find all τ-durable triangles anchored by p, we run a τ-durable ball query around p on $\mathcal{D}$ (Section 2.2) to get an implicit representation (as a bounded number of canonical subsets) of all points within distance 1 from p, where I_p^- is the largest left endpoint among their lifespans (p should be the newest point among the three points of each triangle we report). Recall that each canonical subset returned consists of points within a ball of a small diameter, so we can approximate inter-ball distances among points by the distances among the ball centers. For every pair of balls, if their centers are within distance 1 (plus some slack), we report all τ-durable triangles consisting of p and the Cartesian product of points in the two balls.

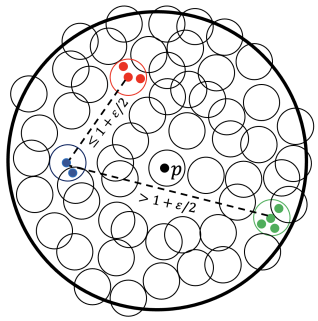

Fig. 1. Illustration of Algorithm 1: p is visited. The small (possibly overlapping) balls represent the canonical nodes returned from $\mathcal{D}$. Each point within distance 1 from p lies in exactly one such ball. We report the triangles formed by p and the points in red and blue balls that satisfy the durability constraint. We do not report triangles formed by p and the points in blue and green balls because they are well separated.

Algorithm. As a preprocessing step, we construct the data structure $\mathcal{D}$ as described in Section 2.2 over P. Our algorithm invokes ReportTriangle (Algorithm 1) for each point $p \in P$. ReportTriangle runs a τ-durable ball query durableBallQ$(p, \tau, \varepsilon/2)$—note the use of $\varepsilon/2$ here for technical reasons—and obtains a family of disjoint result point sets $C_p = \{C_{p,1}, C_{p,2}, \ldots, C_{p,k}\}$ for some $k = O(\varepsilon^{-\rho})$ (see also Figure 1). Each $C_{p,j}$ is covered by a cover tree ball in $\mathcal{D}$ with diameter of no more than $\varepsilon/2$, and contains all points therein whose intervals "sufficiently intersect" I_p, i.e., any q in the ball satisfying $I_q^- + \tau \le I_p^- + \tau \le I_q^+$, as explained in Section 2.2.

Given p, all τ-durable triangles (p, q, s) anchored by p can be classified into two types: (1) q and s belong to the same result point set $C_{p,j}$ for some j; and (2) q and s belong to different sets $C_{p,i}$ and $C_{p,j}$ (where $i \neq j$) that are sufficiently close. To report triangles of the first type, we simply enumerate all pairs of q and s within $C_{p,j}$, for each j. We avoid duplicate reporting of (p, q, s) and (p, s, q) by always picking q as the point with the smaller index in $C_{p,j}$. To report triangles of the second type, we consider (i, j) pairs where Rep_i and Rep_j, the representative points of the balls containing $C_{p,i}$ and $C_{p,j}$, are within distance $1 + \varepsilon/2$. We simply enumerate the Cartesian product of $C_{p,i}$ and $C_{p,j}$. We avoid duplicate reporting of (p, q, s) and (p, s, q) by imposing the order $i < j$.

Correctness. Let (p, q, s) be a triangle reported by our algorithm. We show that (p, q, s) is a τ-durable ε-triangle. From Section 2.2, we know that $\phi(p, q) \le 1 + \varepsilon/2$ and $\phi(p, s) \le 1 + \varepsilon/2$, because q and s belong in one or two canonical subsets in C_p. If q and s belong to the same canonical subset $C_{p,i}$ then by definition $\phi(q, s) \le \varepsilon/2$. If $q \in C_{p,i}$ and $s \in C_{p,j}$ for $i \neq j$, then $\phi(q, s) \le \phi(q, \text{Rep}_i) + \phi(s, \text{Rep}_j) + \phi(\text{Rep}_i, \text{Rep}_j) \le \varepsilon/4 + \varepsilon/4 + (1 + \varepsilon/2) \le 1 + \varepsilon$. In every case, it is true that $\phi(p, q), \phi(p, s), \phi(s, q) \le 1 + \varepsilon$. Hence, (p, q, s) is an ε-triangle. Next, we show that $|I_p \cap I_q \cap I_s| \ge \tau$. Recall that by definition, $|I_p| \ge \tau$. Using the τ-durable ball query durableBallQ$(p, \tau, \varepsilon/2)$, we have that $|I_p \cap I_q| \ge \tau$, $|I_p \cap I_s| \ge \tau$, $I_p^- \in I_q$, and $I_p^- \in I_s$ (see also Definition 2.1). We can rewrite these inequalities as $I_q^- + \tau \le I_p^- + \tau \le I_q^+$ and $I_s^- + \tau \le I_p^- + \tau \le I_s^+$. Hence, $I_p^- + \tau \le \min\{I_q^+, I_s^+\}$, concluding that $|I_p \cap I_q \cap I_s| \ge \tau$. Each triangle (p, q, s) is reported only once, in a specific vertex order: the temporal conditions ensure that p anchors the triangle, and the ordering of q and s is consistently enforced by ReportTriangle. Overall, we showed that if (p, q, s) is reported, then it is a τ-durable ε-triangle and it is reported exactly once.

Next, we prove that we do not miss any τ-durable triangle. Let (p, q, s) be a τ-durable triangle. Without loss of generality, assume that $I_p^- \ge \max\{I_q^-, I_s^-\}$. By definition, $\phi(p, q) \le 1$, $\phi(p, s) \le 1$, and $\phi(q, s) \le 1$. Hence, after visiting p, by the definition of the τ-durable ball query durableBallQ$(p, \tau, \varepsilon/2)$, there exist indexes i, j such that $q \in C_{p,i}$ and $s \in C_{p,j}$. If $i = j$, then (p, q, s) must be reported by Line 4 of ReportTriangle. If $i \neq j$, note that $\phi(\text{Rep}_i, \text{Rep}_j) \le$

$\phi(\text{Rep}_i, q) + \phi(q, s) + \phi(s, \text{Rep}_j) \le \varepsilon/4 + 1 + \varepsilon/4 \le 1 + \varepsilon/2$; therefore (p, q, s) must be reported by Line 8. Overall, we showed that every τ-durable triangle is reported by Algorithm 1.

Time complexity. By Lemma 2.2, we can construct $\mathcal{D}$ in time $O(n \log^2 n)$. For each $p \in P$, we run a τ-durable ball query on $\mathcal{D}$ in $O(\varepsilon^{-O(\rho)} \log n)$ time. Moreover, $|C_p| = O(\varepsilon^{-O(\rho)})$. Checking pairs in C in which to search for triangles of Type (2) takes additional $O(\varepsilon^{-2 \cdot O(\rho)})$ time. All pairs of points examined by ReportTriangle are indeed returned, and together they correspond to all τ-durable triangles plus some ε-triangles involving p. Hence, the overall additional time incurred is $O(\text{OUT})$ where $|T_\tau| \le \text{OUT} \le |T_\tau^\varepsilon|$.

Theorem 3.1. *Given (P, ϕ, I), $\tau > 0$, and $\varepsilon > 0$, ε-approximate DurableTriangle can be solved in $O\left(n(\varepsilon^{-O(\rho)} \log n + \log^2 n) + OUT\right)$ time, where $n = |P|$, ρ is the doubling dimension of P, and OUT is the number of triangles reported.*

Remark 1. For every ℓ_α norm in $\mathbb{R}^d$, we can simplify the data structure $\mathcal{D}$ using a quadtree instead of a cover tree. The running time and approximation with respect to the overall number of reported triangles remain the same.

Remark 2. Our algorithm can be extended to support *delay-guaran-teed enumeration* [3, 7, 34] of durable patterns, i.e., the time between reporting two consecutive patterns is bounded. After spending $O(n(\varepsilon^{-O(\rho)} \log n + \log^2 n))$ preprocessing time, we can support $O(\varepsilon^{-O(\rho)} \log n)$-delay enumeration for ε-approximate DurableTriangle.

Remark 3. Using a dynamic cover tree, we can extend our algorithm to the dynamic setting where we do not have all points upfront. If points are inserted or deleted according to their lifespans, we support $O(\log^3 n)$ amortized update time. After inserting a point p, we can report the new (if any) triangles that p participates in using Algorithm 1) in time near linear to the number of new triangles reported. We show the details in Appendix B.

4 INCREMENTAL REPORTING WHEN VARYING τ

We next consider reporting durable triangles when queries with different durability parameters arrive in an online fashion. As discussed in Section 1.1, the problem, IncrDurableTriangle, boils down to reporting any new result triangles in $T_\tau \setminus T_{\tau_\prec}$, where $\tau_\prec > \tau$ are the previous and current durability parameters, respectively.

As a starter, we can proceed similarly as in Section 3, reporting durable triangles for each anchor point p, but taking care to ensure that we report only τ-durable triangles that are not $\tau_\prec$-durable. Doing so entails retrieving candidate pairs (q, s) as in ReportTriangle, but additionally guaranteeing that *at least one* of q and s ends between $I_p^- + \tau$ and $I_p^- + \tau_\prec$, which leads to $I(p, q, s)$ having durability between τ and $\tau_\prec$. This additional search condition necessitates the modified data structure $\mathcal{D}'$ discussed in Section 2.

However, the naive approach above has the following problem. It is possible that we carry out the search on $\mathcal{D}'$ for p, only to realize that in the end no new result triangle needs to be reported. Ideally, we instead want an *output-sensitive* algorithm whose running time depends only on the output size. To this end, we need an efficient way to test whether p should be *activated* for output; i.e., there is at least one triangle in $T_\tau \setminus T_{\tau_\prec}$ anchored by p. This test motivates the idea of activation thresholds below.

Definition 4.1 (Activation threshold). Given (P, ϕ, I) and $\tau > 0$, the *activation threshold* of $p \in P$ with respect to τ is defined as:

$$\beta_p^\tau = \max\{\tau' < \tau \mid \exists q, s \in P : I_q^- \le I_p^-,\ I_s^- \le I_p^-,\ \text{and}(p, q, s) \text{ is } \tau'\text{-durable but not } \tau\text{-durable}\}.$$

We set $\beta_p^\tau = -\infty$ if no such τ' exists. We call $\beta_p^{+\infty}$ the *maximum activation threshold* of p.

Algorithm 2: ReportDeltaTriangle($\mathcal{D}', p, \tau, \tau_\prec, \varepsilon$)

$C_p : \{C_{p,1}, C_{p,2}, \cdots, C_{p,k}\} \leftarrow$ durableBallQ$'(p, \tau, \tau_\prec, \varepsilon/2)$, with Rep_i as the representative point of the ball for $C_{p,i}$ and $C_{p,i} = \Lambda_{p,i} \cup \overline{\Lambda}_{p,i}$;
foreach $j \in [k]$ **do**
  foreach $q, s \in \Lambda_{p,j}$ *where* q *precedes* s **do**
    report (p, q, s);
  foreach $(q, s) \in \Lambda_{p,j} \times \overline{\Lambda}_{p,j}$ **do report** (p, q, s) ;
foreach $i, j \in [k]$ *where* $i < j$ **do**
  if $\phi(Rep_i, Rep_j) \leq 1 + \frac{\varepsilon}{2}$ **then**
    foreach $(q, s) \in \Lambda_{p,i} \times \Lambda_{p,j}$ **do report** (p, q, s) ;
    foreach $(q, s) \in \Lambda_{p,i} \times \overline{\Lambda}_{p,j}$ **do report** (p, q, s) ;
    foreach $(q, s) \in \overline{\Lambda}_{p,i} \times \Lambda_{p,j}$ **do report** (p, q, s) ;

With activation thresholds, we can easily determine whether to activate p: the condition is precisely $\beta_p^{\tau_\prec} \geq \tau$. If $\beta_p^{\tau_\prec} < \tau$, by definition of $\beta^{\tau_\prec}$, any τ-durable triangle anchored by p is already $\tau_\prec$-durable and hence does not need to be reported; otherwise, we need to at least report $\beta^{\tau_\prec}$-durable triangles anchored by p.

In the following subsections, we first describe the algorithm for processing each activated point (Section 4.1), and then address the problem of computing activation thresholds efficiently (Section 4.2), which requires maintaining additional data structures across queries to help future queries. Finally, we summarize our solution and discuss its complexity (Section 4.3). In [4], we describe the specialized solution for the ℓ_∞-metric.

4.1 Reporting for each activated point

Given an activated point $p \in P$, for which we have already determined that $\beta_p^{\tau_\prec} \geq \tau$, we report all τ-durable triangles anchored by p that are not $\tau_\prec$-durable, using ReportDeltaTriangle (Algorithm 2) explained further below.

As discussed at the beginning of this section, reporting triangles (p, q, s) that are τ-durable but not $\tau_\prec$-durable entails ensuring that at least one of q and s ends between $I_p^- + \tau$ and $I_p^- + \tau_\prec$. To this end, we use the modified data structure $\mathcal{D}'$ discussed in Section 2. We query $\mathcal{D}'$ using durableBallQ$'(p, \tau, \tau_\prec, \varepsilon/2)$ to get $k = O(\varepsilon^{-O(\rho)})$ canonical balls of the cover tree in $\mathcal{D}'$ with representative points $\text{Rep}_1, \text{Rep}_2, \ldots, \text{Rep}_k$; durableBallQ$'$ further partitions the result point set $C_{p,j}$ associated with each ball centered at Rep_j into two subsets

$$\Lambda_{p,j} = \left\{q \in C_{p,j} \mid I_q^+ < I_p^- + \tau_\prec\right\}, \overline{\Lambda}_{p,j} = \left\{q \in C_{p,j} \mid I_q^+ \geq I_p^- + \tau_\prec\right\}.$$

By definition, if $q \in \Lambda_{p,j}$, then $I_q^- \leq I_p^-$ and $I_p^- + \tau \leq I_q^+ < I_p^+ + \tau_\prec$, while if $q \in \overline{\Lambda}_{p,j}$ then $I_q^- \leq I_p^-$ and $I_q^+ \geq I_p^- + \tau_\prec$. See Figure 2 for an illustration. Recall that durableBallQ$'$ does not explicitly construct $\Lambda_{p,j}$ and $\overline{\Lambda}_{p,j}$; instead, these subsets correspond to canonical subsets of nodes in the interval trees within $\mathcal{D}'$.

For (p, q, s) to be *not* $\tau_\prec$-durable, at least one of q and s must belong to some $\Lambda_{p,j}$ instead of $\overline{\Lambda}_{p,j}$. Therefore, we can divide all triangles (p, q, s) that are τ-durable but not $\tau_\prec$-durable into four types, which can be computed with the help of the above partitioning: (1) $q, s \in \Lambda_{p,j}$ for some j; (2) $q \in \Lambda_{p,j}$ and $s \in \overline{\Lambda}_{p,j}$ for some j; (3) $q \in \Lambda_{p_i}$ and $s \in \Lambda_{p,j}$ for some $i \neq j$ where Rep_i and Rep_j are sufficiently close; (4) $q \in \Lambda_{p_i}$ and $s \in \overline{\Lambda}_{p,j}$ for some $i \neq j$ where Rep_i and Rep_j are sufficiently close. ReportDeltaTriangle (Algorithm 2) covers all these cases. As with ReportTriangle in

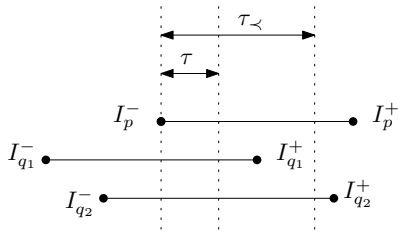

Fig. 2. An illustration of $C_{p,j} = \Lambda_{p,j} \cup \overline{\Lambda}_{p,j}$. Here $q_1 \in \Lambda_{p,j}$ and $q_2 \in \overline{\Lambda}_{p,j}$.

Section 3, we enforce an ordering between q and s to ensure that only one of (p, q, s) and (p, s, q) is reported. Thanks to the implicit representation of $\Lambda_{p,j}$'s and $\overline{\Lambda}_{p,j}$'s, ReportDeltaTriangle avoids enumerating points in a subset if they do not contribute to any result triangle. For example, if $\overline{\Lambda}_{p,i} = \emptyset$ (line 10 of Algorithm 2), we short-circuit the computation and avoid enumerating $\Lambda_{p,j}$.

Remark. Note that ReportTriangle (Section 3) can be seen as as special case of ReportDeltaTriangle whenever $\tau_\prec > \max_{p \in P} \beta_p^{+\infty}$.

4.2 Computing activation thresholds

We turn to the question of how to compute the activation threshold $\beta^{\tau_\prec}$ given p and $\tau_\prec$, required for determining whether to activate p. Naively, we can find all triangles anchored by p (regardless of durability) and build a map of all activation thresholds for p. However, we have a more efficient solution that builds on two ideas. First, consider a sequence of queries with durability parameters $\tau_1, \tau_2, \ldots$. After answering the current query, say τ_i, we compute and remember $\beta_p^{\tau_i}$ for each (relevant) p, so they are available to help the next query τ_{i+1}. Second, we use a binary search procedure to look for activation thresholds within a desired range, by exploiting the extended data structure $\mathcal{D}'$ to quickly test existence of thresholds in a range without enumerating result triangles therein. The second idea is implemented by ComputeActivation (Algorithm 3). It runs a binary search making guesses for the value of β_p^τ. For each guess τ' of β_p^τ, we use a primitive called DetectTriangle to test whether there exists any triangle anchored by p that is τ'-durable but not τ-durable—in other words, whether $\beta_p^\tau \in [\tau', \tau)$. DetectTriangle mirrors Algorithm 2, except that it merely checks the existence of triangles for each type returning true or false instead of reporting them. Using durableBallQ′ for returning implicit representations for $\Lambda_{p,j}$'s and $\overline{\Lambda}_{p,j}$'s, it is quick to check whether their combinations yield a non-empty result set. Given p, the search space of activation thresholds has only $O(n)$ possibilities: The lifespan of every triangle (p, q, s) anchored by p is either in $[I_p^-, I_q^+]$ or $[I_p^-, I_s^+]$, thus the durability of any triangle anchored by p falls into the set $\{I_q^+ - I_p^- \mid q \in P, I_q^+ \geq I_p^-\}$. The number of steps in the binary search and the number of invocations of DetectTriangle is $O(\log n)$.

We are now ready to put together the data structures and procedure for computing and maintaining activation thresholds. We use two simple binary search trees $\mathcal{S}_\alpha$ and $\mathcal{S}_\beta$. $\mathcal{S}_\alpha$ indexes all points $p \in P$ by their maximum activation thresholds $\beta_p^{+\infty}$. We precompute $\mathcal{S}_\alpha$ by calling ComputeActivation for each $p \in P$. Once constructed, $\mathcal{S}_\alpha$ remains unchanged across queries.

$\mathcal{S}_\beta$ indexes points by their activation thresholds with respect to the durability parameter. Suppose the current query parameter is τ and the previous one is $\tau_\prec$. Before executing the current query, $\mathcal{S}_\beta$ indexes each point p by $\beta_p^{\tau_\prec}$, so the current query can use $\mathcal{S}_\beta$ to find p's with $\beta_p^{\tau_\prec} \geq \tau$ to activate. After completing the current query, we update $\mathcal{S}_\beta$ for the next round: as long as there exists a τ-durable triangle anchored by p, $\mathcal{S}_\beta$ indexes p by the value of β_p^τ. Initially, $\mathcal{S}_\beta$ starts out as an empty tree, which can be interpreted as having completed an initial query with durability parameter $+\infty$.

Algorithm 3: ComputeActivation($\mathcal{D}', p, \tau, \varepsilon$)

```
1  I⁺ ← {I_q⁺ | q ∈ P}, R ← n − 1, L ← 0, τ_ret ← −∞;
2  while L ≤ R do
3      m ← ⌊(L + R)/2⌋;
4      if I⁺[m] > I_p⁻ + τ then R ← m − 1;
5      else if I⁺[m] < I_p⁻ then L ← m + 1;
6      τ′ ← I⁺[m] − I_p⁻;
7      B ← DetectTriangle(τ′, τ);
8      if B = true then τ_ret ← τ′, L ← m + 1;
9      else R ← m − 1;
10 return τ_ret;
11 Subroutine DetectTriangle(τ_1, τ_2) begin
12     C_p : {C_{p,1}, C_{p,2}, ··· , C_{p,k}} ← durableBallQ′(p, τ_1, τ_2, ε/2), with Rep_i denoting the
         representative point of the ball for C_{p,i} and C_{p,i} = Λ_{p,i} ∪ Λ̄_{p,i};
13     foreach j ∈ [k] do
14         if |Λ_{p,j}| ≥ 2 then return true;
15         if |Λ_{p,j}| ≥ 1 and |Λ̄_{p,j}| ≥ 1 then return true;
16     foreach i, j ∈ [k] where i < j do
17         if φ(Rep_i, Rep_j) ≤ 1 + ε/2 then
18             if |Λ_{p,i}| ≥ 1 and |Λ_{p,j}| ≥ 1 then return true;
19             if |Λ_{p,i}| ≥ 1 and |Λ̄_{p,j}| ≥ 1 then return true;
20             if |Λ̄_{p,i}| ≥ 1 and |Λ_{p,j}| ≥ 1 then return true;
21     return false;
```

Maintenance of $\mathcal{S}_\beta$ has two cases depending on the current query. First, consider the more interesting case of $\tau_\prec > \tau$, where we need to potentially report new result triangles. For each p activated, i.e., $\beta^{\tau_\prec} \geq \tau$, we call ComputeActivation($\mathcal{D}', p, \tau, \varepsilon$) to obtain β_p^τ and update p's entry in $\mathcal{S}_\beta$. This is all we need to do to maintain $\mathcal{S}_\beta$ because, if p were not activated for the current query, we would have $\beta^{\tau_\prec} < \tau$, and therefore $\beta_p^\tau = \beta_p^{\tau_\prec}$.

In the less interesting case of $\tau \geq \tau_\prec$, there are no new result triangles to report, but some old ones may need to be invalidated. Strategies for maintaining $\mathcal{S}_\beta$ differ depending on the usage scenario. In the first scenario, suppose that the client issuing the query sequence incrementally maintains the query result as lists of triangles grouped by anchor points, and triangles within each list are sorted by durability. When $\tau \geq \tau_\prec$, the client can simply trim its lists according to τ. During this process, it can easy obtain and pass information to the server for updating $\mathcal{S}_\beta$: for each anchor p, β_p^τ simply takes on the highest durability value removed from p's list, or it remains unchanged if no triangle is removed. In the alternative (and less likely) scenario where the client does not remember anything, the server can simply rebuild $\mathcal{S}_\beta$ by running ComputeActivation for each $p \in \mathcal{S}_\alpha$ with maximum activation threshold no less than τ.

Correctness. We first show that the values β_p^τ are updated correctly in Algorithm 3. Let τ' be the parameter in the binary search that we checked in Algorithm 3. Point p can only form a τ'-durable ε-triangle with points q whose intervals I_q intersect I_p^- and either $I_q^+ < I_p^- + \tau$ or $I_q^+ \geq I_p^- + \tau$. $\bigcup_j \Lambda_{p,j}$ is the set of points satisfying the first inequality, and $\bigcup_j \overline{\Lambda}_{p,j}$ the set of points in the second inequality.

For every pair $q, s \in \overline{\Lambda}_{p,j}$, we do not activate point p with durability τ'. If indeed $\phi(q,s) \leq 1$ and $q, s \in \overline{\Lambda}_{p,j}$, then (p,q,s) is a τ-durable triangle. So our algorithm does not activate a point p because of a previously reported τ-durable triangle. By definition, it is also straightforward to see that p should be activated at durability τ' if there is a pair of points $q, s \in \Lambda_{p,j} \cup \overline{\Lambda}_{p,j}$ such as either q or s belongs in $\Lambda_{p,j}$. This is because either I_q or I_s does not overlap with I_p for more than τ and overlaps more than τ', so (p,q,s) was not a τ-durable ε'-triangle for every $\varepsilon' > 0$. Next, let p be a point that is activated because of a triangle (p,q,s). We show that any (p,q,s) is a τ'-durable ε-triangle. As we mentioned, either I_q or I_s does not intersect I_p for more than τ but intersects I_p for more than τ' so it remains to show that $\phi(p,q), \phi(p,s), \phi(q,s) \leq 1+\varepsilon$. By the definition of $\mathcal{D}'$ we have that $\phi(p,q) \leq 1+\varepsilon/2$ and $\phi(p,s) \leq 1+\varepsilon/2$. If q, s belong in the same subset $C_{p,j}$ then it also follows that $\phi(q,s) \leq \varepsilon/4 \leq 1+\varepsilon$. If $q \in C_{p,j}$ and $s \in C_{p,i}$ for $i < j$ then in Algorithm 3 we only consider this triangle if and only if $\phi(\mathsf{Rep}_j, \mathsf{Rep}_i) \leq 1+\varepsilon/2$. We have $\phi(q,s) \leq \phi(q,\mathsf{Rep}_j) + \phi(\mathsf{Rep}_j,\mathsf{Rep}_i) + \phi(\mathsf{Rep}_i,s) \leq 1+\varepsilon/2+\varepsilon/4+\varepsilon/4 = 1+\varepsilon$. So (p,q,s) is a τ'-durable ε-triangle that is not τ-durable.

The correctness of Algorithm 2 follows from the same arguments we used to prove the correctness of Algorithm 3. Overall, Algorithm 2 reports all τ_{i+1}-durable triangles along with some τ_{i+1}-durable ε-triangles, that are not τ_i-durable ε-triangles. Hence, $|T_{\tau_{i+1}} \setminus T_{\tau_i}| \leq \mathsf{OUT} \leq |T^{\varepsilon}_{\tau_{i+1}} \setminus T^{\varepsilon}_{\tau_i}|$.

4.3 Solution summary and complexity

In summary, we build the data structure $\mathcal{D}'$ as described in Section 2.2; its size is $O(n\log^2 n)$, and it can be constructed in $O(n\log^3 n)$ time. We also build the index $\mathcal{S}_\alpha$ of maximum activation thresholds, which has size $O(n)$. To construct $\mathcal{S}_\alpha$, as mentioned, we perform at most $O(\log n)$ guesses for each point, and each guess invokes DetectTriangle once, which takes $O(\varepsilon^{-O(\rho)}\log^2 n)$ time; therefore, the total construction time for $\mathcal{S}_\alpha$ is $O(n\varepsilon^{-O(\rho)}\log^3 n)$. Finally, we maintain the index $\mathcal{S}_\beta$ of activation thresholds for the current durability parameter; its size is $O(n)$, its initial construction time is $O(1)$, and its maintenance time will be further discussed below.

To report new result triangles when the durability parameter changes from $\tau_\prec$ to τ, we use $\mathcal{S}_\beta$ to search for points p with $\beta_p^{\tau_\prec} \geq \tau$ to activate. Each activated point p requires $O(\mathsf{OUT}_p + \varepsilon^{-O(\rho)}\log^2 n)$ time for ReportDeltaTriangle to report all new durable triangles anchored by p, where OUT_p denotes the number of them. Then, to maintain $\mathcal{S}_\beta$, we need $O(\varepsilon^{-O(\rho)}\log^3 n)$ time for ComputeActivation, and $O(\log n)$ time to update $\mathcal{S}_\beta$ for each point p activated. For each activated p, at least one new durable triangle is reported, so the number of calls to ComputeActivation is bounded by the output size. Overall, we spend $O\left(\mathsf{OUT} + \varepsilon^{-O(\rho)}\log^2 n\right)$ time for reporting and $O(\mathsf{OUT}\cdot\varepsilon^{-O(\rho)}\log^3 n)$ time for maintenance, where OUT is the number of results reported.

Theorem 4.2. *Given (P,ϕ,I), and $\varepsilon > 0$, a data structure of size $O(n\log^2 n)$ can be constructed in $O(n\varepsilon^{-O(\rho)}\log^3 n)$ time such that, the ε-approximate IncrDurableTriangle problem can be solved in $O\left(OUT\cdot\varepsilon^{-O(\rho)}\log^3 n\right)$ time, where $n = |P|$, ρ is the doubling dimension of P, and OUT is the number of results reported.*

5 REPORTING AGGREGATE-DURABLE PAIRS

5.1 SUM

We start by describing a data structure that allows us to efficiently compute the total length of all intersections between a query interval with a given set of intervals. Then we show how to use this primitive to report all τ-SUM-durable pairs for AggDurablePair-SUM, along with some τ-SUM-durable ε-pairs (but no other pairs).

Interval-SUM-durability. Given a set of intervals $\mathcal{I}$, we want a primitive that can efficiently decide, given any query interval J and $\tau > 0$, whether $\sum_{I \in \mathcal{I}} |I \cap J| \geq \tau$. To this end, we construct a data structure IT^{Σ} over $\mathcal{I}$, which is a variant of an interval tree where each tree node v is annotated with the following information:

- $|v|$, the total number of intervals stored at v;
- $\sum_{I \in v} |I|$, the total length of intervals stored at v;
- $\sum_{I \in v} I^+$, the sum of right endpoints of intervals stored at v;
- $\sum_{I \in v} I^-$, the sum of left endpoints of intervals stored at v.

Given a query interval J, we obtain $O(\log^2 n)$ canonical set of nodes in IT^{Σ}, where each node v falls into: (1) every $I \in v$ completely covers J; (2) J completely covers every $I \in v$; (3) every $I \in v$ partially intersects J with $I^+ \in J$; (4) every $I \in v$ partially intersects J with $I^- \in J$. Then, we can rewrite the SUM-durability of intervals with respect to J as follows:

$$\sum_{I \in \mathcal{I}} |I \cap J| = \sum_{v} \sum_{I \in v} |I \cap J| = \sum_{v} \begin{cases} |v| \cdot |J| & \text{if (1);} \\ \sum_{I \in v} |I| & \text{if (2);} \\ \sum_{I \in v} I^+ - |v| \cdot J^- & \text{if (3);} \\ |v| \cdot J^+ - \sum_{I \in v} I^- & \text{if (4).} \end{cases}$$

Note that IT^{Σ} can be constructed in $O(n \log^2 n)$ time and uses $O(n \log n)$ space. This way, we have a procedure ComputeSumD which, given IT^{Σ} and J, returns $\sum_{I \in \mathcal{I}} |I \cap J|$ in $O(\log^2 n)$ time. The interval tree IT^{Σ} can also be used to find C_p.

Data structure. While IT^{Σ} makes it efficient to sum durabilities over a set of intervals given $I_p \cap I_q$ for a candidate pair (p, q), we cannot afford to check all possible pairs, and we have not yet addressed the challenge of obtaining the intervals of interest (which must come from witness points incident to both p and q) in the first place. The high-level idea is to leverage the same space decomposition from the previous sections to efficiently obtain canonical subsets of witness points, in their implicit representation. These canonical subsets of intervals serve as the basis for building IT^{Σ} structures. In more detail, we construct $\mathcal{D}^{\Sigma}$ in a similar way as $\mathcal{D}$ in Section 2.2. Like $\mathcal{D}$, $\mathcal{D}^{\Sigma}$ is a two-level data structure consisting of a cover tree and an interval tree variant (as described above) for every node of the cover tree. For each cover tree node u, let C_u denote the subset of points in P within the ball of u centered at Rep_u. We build IT^{Σ}_u over C_u with SUM annotations, and for each node in IT^{Σ}_u, we also store points in decreasing order of their right interval endpoints. Overall, we can construct $\mathcal{D}^{\Sigma}$ in $O(n \log^3 n)$ time having $O(n \log^2 n)$ space.

Algorithm. We report all τ-SUM-durable pairs (p, q) where $I_p^- \geq I_q^-$ (to avoid duplicates); we say p *anchors* the pair. For each $p \in P$, we invoke ReportSUMPair (Algorithm 4) to report τ-SUM-durable ε-pairs (p, q) anchored by p. To this end, ReportSUMPair runs the τ-durable ball query $\mathsf{durableBallQ}(p, \tau, \varepsilon/2)$ over $\mathcal{D}^{\Sigma}$, and obtain a family of result point sets $C_{p,1}, C_{p,2}, \ldots, C_{p,k}$ for some $k = O(\varepsilon^{-\rho})$. Each $C_{p,j}$ is covered by a cover tree ball in $\mathcal{D}^{\Sigma}$ with diameter of no more than $\varepsilon/2$, and contains all points q within the ball where $I_q^- + \tau \leq I_p^- + \tau \leq I_q^+$, as explained in Section 2.2, sorted with respect to I_q^+. Let $\mathsf{IT}^{\Sigma}_{p,j}$ denote the interval tree for the cover tree node corresponding to $C_{p,j}$. For each j, we go through each point $q \in C_{p,j}$ in decreasing order of right endpoints to check whether (p, q) is τ-SUM-durable. To do this check, we consider witnesses from point sets $C_{p,1}, C_{p,2}, \ldots, C_{p,k}$. We can skip an entire set $C_{p,i}$ if its ball center Rep_i is too far from Rep_j, because all points in $C_{p,i}$ would be too far from q. Otherwise, we query $\mathsf{IT}^{\Sigma}_{p,i}$ using interval $I_p \cap I_q$ to obtain the sum of durabilities over all witnesses in $C_{p,i}$. We compute these partial sums together and compare the total with $\tau + 2 \cdot |I_p \cap I_q|$ (note that the second term accounts for the fact that the partial sums include the contributions of p and q themselves, which should be discounted). If the total passes the threshold, we report (p, q). If not, we stop consider any remaining point $q' \in C_{p,j}$

Algorithm 4: ReportSUMPair($\mathcal{D}^{\Sigma}, p, \tau, \varepsilon$)

1 $C_p : \{C_{p,1}, C_{p,2}, \cdots, C_{p,k}\} \leftarrow$ durableBallQ$(p, \tau, \varepsilon/2)$, with Rep_i as the representative point of the cover tree node for $C_{p,i}$, and $\text{IT}^{\Sigma}_{p,i}$ as the annotated interval tree for the cover tree node;
2 **foreach** $j \in [k]$ **do**
3 **foreach** $q \in C_{p,j}$ *in descending order of* I^+_q **do**
4 $t \leftarrow 0$;
5 **foreach** $i \in [k]$ **do**
6 **if** $\phi(Rep_i, Rep_j) \leq 1 + \frac{\varepsilon}{2}$ **then**
7 $t \leftarrow t +$ ComputeSumD$(\text{IT}^{\Sigma}_{p,i}, I_p \cap I_q)$;
8 **if** $t \geq \tau + 2 \cdot |I_p \cap I_q|$ **then report** (p, q) ;
9 **else break**;

(which has $I^+_{q'} < I^+_q$), since $I_p \cap I_{q'} \subset I_p \cap I_q$ and will surely yield a lower total durability. This is the key for output-sensitive time.

Correctness. First, each pair (p, q) is reported at most once, as (p, q) is reported if $I^-_p \geq I^-_q$. Next, we show that every pair reported must be a τ-SUM-durable ε-triangle. Consider a pair (p, q) that is reported. Note that our algorithm considers a node u_i from $\mathcal{D}^{\Sigma}$ with radius $\varepsilon/4$ if and only if $\phi(p, \text{Rep}_i) \leq 1 + \varepsilon/4$. If $q \in u_i$, we have $\phi(p, q) \leq \phi(p, \text{Rep}_i) + \phi(\text{Rep}_i, q) \leq 1 + \varepsilon/2$. Hence, in any pair (p, q) we return it holds that $\phi(p, q) \leq 1 + \varepsilon$. Then we only consider points within distance $1 + \varepsilon$ from both p, q to find the sum of their corresponding intervals. Indeed, we only consider the pairs $C_{p,i}, C_{p,j}$ with $\phi(\text{Rep}_i, \text{Rep}_j) \leq 1 + \varepsilon/2$. Let q' be any point from $C_{p,i}$. We have $\phi(p, q') \leq 1 + \varepsilon/2$, and $\phi(q, q') \leq \phi(\text{Rep}_i, q) + \phi(\text{Rep}_i, \text{Rep}_j) + \phi(\text{Rep}_j, q') \leq 1 + \varepsilon$. Overall, by showing i) $\phi(p, q) \leq 1$, ii) that we only take the sum of intervals in $I_p \cap I_q$ among points (witness points) within distance $1 + \varepsilon$ from both p, q, and iii) the correctness of the IT^{Σ} data structure, we conclude that the reporting pair (p, q) is a τ-durable ε-pair.

Finally, we show that every τ-SUM-durable pair will be reported. Let (p, q) be an arbitrary τ-SUM-durable pair. Suppose $q \in u_j$, where u_j is a node of $\mathcal{D}^{\Sigma}$ of radius at most $\varepsilon/4$, with representative point Rep_j. Since $\phi(p, \text{Rep}_j) \leq \phi(p, q) + \phi(q, \text{Rep}_j) \leq 1 + \varepsilon/4$, we have that $u_j \in C_p$. Without loss of generality, assume that $q \in C_{p,j}$. Next, we show that for point $q' \in P$, if $\phi(p, q') \leq 1$ and $\phi(q, q') \leq 1$, we always consider q' in the witness set. Since $\phi(p, q') \leq 1$ we have that $q' \in C_p$. Without loss of generality, assume that $q' \in C_{p,i}$. In this case, $\phi(\text{Rep}_j, \text{Rep}_i) \leq \phi(\text{Rep}_j, q) + \phi(q, q') + \phi(q', \text{Rep}_i) \leq 1 + \varepsilon/2$, so q' is included in $\text{IT}^{\Sigma}_{p,i}$ considered in line 7 of Algorithm 4. It remains to show that if (p, q) is a τ-durable pair, q must be visited during the traversal of points in $C_{p,j}$. We prove it by contradiction. Let $w \in C_{p,j}$ be a point such that $I^-_w \leq I^-_p \leq I^+_q \leq I^+_w$. Suppose after visiting w, the traversal of points in $C_{p,j}$ stops. Implied by the stopping condition, (p, w) is not a τ-SUM-durable pair. Meanwhile, as (p, q) is a τ-SUM-durable pair, (p, w) must also be a τ-SUM-durable pair, implied by $I_q \cap I_p \subseteq I_w \cap I_p$, and the fact that we run the ComputeSumD$(\text{IT}^{\Sigma}_{p,i}, I_p \cap I_q)$ query on the same sets $C_{p,i}$, coming to a contradiction. Thus, every τ-SUM-durable pair must be reported.

Time Complexity. The construction time of IT^{Σ} is $O(n \log^2 n)$, so it takes $O(n \log^3 n)$ time to construct $\mathcal{D}^{\Sigma}$. For each p, it takes $O(\varepsilon^{-O(\rho)} + \log n)$ time to derive the canonical set of nodes C_p and $O(\varepsilon^{-O(\rho)} \log n)$ time to derive the sorted intervals in every node of C_p. For each (sorted) interval

I_q in $C_{p,i}$ we visit $O(\varepsilon^{-O(\rho)})$ other nodes $C_{p,j}$ and we run a $O(\log^2 n)$ time query to find the sum using IT^{Σ}. When we find out that q does not form a τ-SUM-durable pair with p we skip the rest points in $C_{p,i}$ so the running time is output-sensitive. Overall, the running time is bounded by $O(n \log^3 n + (n + \mathrm{OUT}) \cdot \varepsilon^{-O(\rho)} \log^2 n)$, where $K_\tau \leq \mathrm{OUT} \leq K^{\varepsilon}_{\tau}$.

Theorem 5.1. *Given (P, ϕ, I), $\tau > 0$ and $\varepsilon > 0$, the ε-approximate AggDurablePair-SUM problem can be solved in $O(n \log^3 n + (n + OUT) \cdot \varepsilon^{-O(\rho)} \log^2 n)$ time, where $n = |P|$, ρ is the doubling dimension of P, and OUT is the number of pairs reported.*

5.2 UNION

Solving the general AggDurablePair-UNION problem is challenging because of the inherent hardness of computing the union of intervals that intersect a query interval, i.e., we cannot design an efficient primitive for UNION as the ComputeSumD primitive for SUM. In practice, even if the size of the witness set U is large, a smaller subset of U may be all that is required for its union to reach the durability parameter. With this observation, we approach the problem by designing an algorithm whose performance depends on κ, a constraint on the size of the witness set. More precisely, given a durability parameter $\tau > 0$ and a positive integer $\kappa \in \mathbb{Z}^+$, we say a pair $(p_1, p_2) \in P \times P$ is (τ, κ)-*UNION-durable* if $\phi(p_1, p_2) \leq 1$ and there exists $U \subseteq \{u \in P \mid \phi(p_1, u), \phi(p_2, u) \leq 1\}$ such that $|U| \leq \kappa$ and $|\bigcup_{u \in U} I(u, p_1, p_2)| \geq \tau$. An approximate version is defined by replacing the distance constraint ≤ 1 with $\leq 1 + \varepsilon$. We present an $\tilde{O}((n + \mathrm{OUT}) \cdot \kappa \varepsilon^{-O(\rho)})$-time algorithm for $\mathrm{OUT} \in \left[|K_{\tau,k}|, \left|K^{\varepsilon}_{(1-1/e)\tau,\kappa}\right|\right]$, with $K_{\tau,\kappa}$ denoting the set of (τ, κ)-UNION durable pairs and $K^{\varepsilon}_{(1-1/e)\tau,\kappa}$ denoting the set of $((1 - 1/e)\tau, \kappa)$-UNION durable ε-pairs.

(τ, κ)-UNION-durable pair. Next, we focus on finding (τ, κ)-UNION-durable pairs for some known κ, which should work well in practical cases where a handful of witness points are able to provide sufficient coverage for the pair. The overall algorithm has the high-level idea of leveraging the space decomposition as the AggDurablePair-SUM case in Section 5.1, but it requires a primitive different from ComputeSumD and a different way of invoking this primitive across witness subsets.

High-level Idea. Given a set of intervals $\mathcal{I}$ and a target interval J, our approach is to find a subset of intervals $X \subseteq \mathcal{I}$ of size κ that maximizes the UNION-durability with respect to J, namely $\left|\bigcup_{I \in X}(I \cap J)\right|$ (to compare with τ). There is an apparent connection to the *maximum κ-coverage* problem, where given a family of sets over a set of elements, we want to choose κ sets to cover the maximum number of elements. Here, we can regard each set as $I \cap J$ for each $I \in \mathcal{I}$, and the goal is to choose κ such sets to cover as much of J as possible. The standard greedy algorithm gives an $(1 - 1/e)$-approximation for this problem [32], which inspires us to follow a similar approach. We leave details on data structures, pseudocode, correctness and complexity analysis to Appendix D. Our greedy approach chooses one interval at time to cover J, and the choice is always the one that maximizes the resulting increase in coverage. In more detail, let $X \subseteq \mathcal{I}$ denote the set of intervals already chosen, which leaves $J \setminus \bigcup_{I \in X} I$, the uncovered parts of J, as a set Y of intervals. Consider the pair (I_x, I_y), where $I_x \in \mathcal{I} \setminus X$, and $I_y \in Y$, with the largest overlap, i.e., $I_x \cap I_y$; we greedily choose I_x as the next interval to cover J.

To implement this greedy approach efficiently, we build a data structure $\mathcal{D}^{\cup}$ similarly as $\mathcal{D}^{\Sigma}$ in Section 5.1. $\mathcal{D}^{\cup}$ uses a different variant of the interval tree $\mathsf{IT}^{\cup}$, which, given a query interval J, finds the indexed interval with the largest overlap with J. The overall algorithm reports all (τ, κ)-UNION durable pairs for each anchor point p by querying $\mathcal{D}^{\cup}$, and for each candidate (p, q), performs the greedy choice k times to compute the UNION-durability of (p, q). Each greedy choice involves querying the $\mathsf{IT}^{\cup}$ structures for the $O(\varepsilon^{-O(\rho)} \log^2 n)$ canonical subsets of witness points; some additional elementary data structures help ensure that the greedy algorithm takes $O\left(\kappa \varepsilon^{-O(\rho)} \log^2 n\right)$

time. In Appendix D, we show that the overall time is $O\left(n \log^3 n + (n + \mathsf{OUT}) \cdot \varepsilon^{-O(\rho)} \kappa \log^2 n\right)$, where $|K_{\tau,\kappa}| \leq \mathsf{OUT} \leq \left|K^{\varepsilon}_{(1-1/e)\tau,\kappa}\right|$. Putting everything together, we obtain:

Theorem 5.2. *Given (P, ϕ, I), $\tau > 0, \varepsilon > 0$, and integer $\kappa \in \mathbb{Z}^+$, the ε-approximate AggDurablePair-UNION problem can be solved in $O(n \log^3 n + (n + OUT) \cdot \varepsilon^{-O(\rho)} \kappa \log^2 n)$ time, where $n = |P|$, ρ is the doubling dimension of P, and OUT is the number of pairs reported.*

6 RELATED WORK

In database and data mining, there is a large body of literature on finding patterns in temporal graphs [6, 26, 30, 39, 40, 48, 53]. Hu et al. [33] studied the problem of computing temporal join queries efficiently; the problem of finding durable triangles is a special case of the problem they studied with self-joins. While [33, 53] have provable guarantees, the algorithms are expensive, requiring time super-linear in the number of edges to report all durable triangles. Recently, Deng et al. [19] proposed algorithms to report or count triangles (and other simple patterns) in time super-linear in the graph size. In contrast, we work with an implicit representation of the proximity graph and design algorithms that run in time near-linear in the number of nodes and output size.

There is another line of work in computational geometry on detecting triangles and other simple patterns in intersection graphs. Eppstein and Erickson [22] gave an $O(n \log n)$ algorithm to detect if an intersection graph consisting of unit balls in $\mathbb{R}^d$ has a constant clique. Kaplan et al. [36] can detect a triangle in a unit-disk graph in $\mathbb{R}^2$ in $O(n \log n)$ time where edges can be weighted. The approach in [14] can detect in $\tilde{O}(n^{d/2})$ time if a clique of constant size exists in an intersection graph of general boxes in $\mathbb{R}^d$. Chan [15] recently improved the results on detecting cliques, cycles, and other simple patterns in intersection graphs, where the nodes are boxes, general fat objects in $\mathbb{R}^d$, or segments in $\mathbb{R}^2$, and two nodes are connected if the corresponding objects intersect. For example, if nodes are fat objects, their algorithm can detect a constant cycle or clique in $O(n \log n)$ time. The problems we focus on in this paper have major differences with this line of work: (i) previous methods only worked for detecting whether a pattern exists, while our goal is to report all patterns; (ii) all previous works focused on non-temporal graphs, while we consider the more challenging *temporal graphs*, where nodes have lifespans; (iii) we additionally considered an incremental reporting setting to support queries with different parameters.

The notion of durability has been studied in other queries, such as durable top-k queries [27, 28] and durability prediction [29]. It also has been studied in computational topology, where the goal is to compute "persistent" (durable) topological features; see [20, 21].

7 CONCLUSION

In this paper, we have studied the problem of reporting durable patterns in proximity graphs. We work with an implicit representation of the input graph, and propose efficient algorithms that run in near-linear time in the number of nodes, under any general metric with bounded doubling dimension. For future work, we believe that some of our algorithms and data structures can also be used for counting durable patterns in near-linear time (instead of reporting them). Second, while we have focused on simple patterns such as triangles and paths, it would be interesting to explore near-linear time algorithms for more general and complex patterns. Third, we have considered only the case when nodes have lifespans but otherwise remain stationary; one could further consider the case when their positions change over time (hence inducing also lifespans on edges). A possible direction is to use *kinetic data structures* to maintain the evolving graph topology. Finally, a challenging question is whether we can extend our approach to a general graph already with an explicit representation, but without first computing an embedding.

REFERENCES

[1] Doubling dimension in real-world graphs. https://slideplayer.com/slide/5331329/. Accessed: 2023-04-24.

[2] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, pages 434–443. IEEE, 2014.

[3] P. K. Agarwal, X. Hu, S. Sintos, and J. Yang. Dynamic enumeration of similarity joins. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, 2021.

[4] P. K. Agarwal, X. Hu, S. Sintos, and J. Yang. On reporting durable patterns in temporal proximity graphs. https://arxiv.org/abs/2403.16312, 2024.

[5] N. Alon, T. Kaufman, M. Krivelevich, and D. Ron. Testing triangle-freeness in general graphs. *SIAM Journal on Discrete Mathematics*, 22(2):786–819, 2008.

[6] M. Araujo, S. Günnemann, S. Papadimitriou, C. Faloutsos, P. Basu, A. Swami, E. E. Papalexakis, and D. Koutra. Discovery of "comet" communities in temporal and labeled graphs com2. *Knowledge and Information Systems*, 46(3):657–677, 2016.

[7] C. Berkholz, J. Keppeler, and N. Schweikardt. Answering conjunctive queries under updates. In *proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI symposium on Principles of database systems*, pages 303–318, 2017.

[8] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104, 2006.

[9] A. Björklund, R. Pagh, V. V. Williams, and U. Zwick. Listing triangles. In *International Colloquium on Automata, Languages, and Programming*, pages 223–234. Springer, 2014.

[10] M. Borassi, A. Epasto, S. Lattanzi, S. Vassilvitskii, and M. Zadimoghaddam. Better sliding window algorithms to maximize subadditive and diversity objectives. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 254–268, 2019.

[11] N. H. Bshouty, Y. Li, and P. M. Long. Using the doubling dimension to analyze the generalization of learning algorithms. *Journal of Computer and System Sciences*, 75(6):323–335, 2009.

[12] H. Cai, V. W. Zheng, and K. C.-C. Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE transactions on knowledge and data engineering*, 30(9):1616–1637, 2018.

[13] T. M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

[14] T. M. Chan. Klee's measure problem made easy. In *2013 IEEE 54th annual symposium on foundations of computer science*, pages 410–419. IEEE, 2013.

[15] T. M. Chan. Finding triangles and other small subgraphs in geometric intersection graphs. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1777–1805. SIAM, 2023.

[16] J. Chen. Algorithmic graph embeddings. *Theoretical Computer Science*, 181(2):247–266, 1997.

[17] P. Cunningham and S. J. Delany. k-nearest neighbour classifiers-a tutorial. *ACM computing surveys (CSUR)*, 54(6):1–25, 2021.

[18] M. Damian, S. Pandit, and S. Pemmaraju. Distributed spanner construction in doubling metric spaces. In *Principles of Distributed Systems: 10th International Conference, OPODIS 2006, Bordeaux, France, December 12-15, 2006. Proceedings 10*, pages 157–171. Springer, 2006.

[19] S. Deng, S. Lu, and Y. Tao. Space-query tradeoffs in range subgraph counting and listing. In *26th International Conference on Database Theory (ICDT 2023)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2023.

[20] T. K. Dey and Y. Wang. *Computational topology for data analysis*. Cambridge University Press, 2022.

[21] H. Edelsbrunner and J. L. Harer. *Computational topology: an introduction*. American Mathematical Society, 2022.

[22] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11(3):321–350, 1994.

[23] J. Erickson. Static-to-dynamic transformations. http://jeffe.cs.illinois.edu/teaching/datastructures/notes/01-statictodynamic.pdf.

[24] E. Facco, M. d'Errico, A. Rodriguez, and A. Laio. Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific reports*, 7(1):12140, 2017.

[25] A. E. Feldmann and D. Marx. The parameterized hardness of the k-center problem in transportation networks. *Algorithmica*, 82:1989–2005, 2020.

[26] M. Franzke, T. Emrich, A. Züfle, and M. Renz. Pattern search in temporal social networks. In *Proceedings of the 21st International Conference on Extending Database Technology*, 2018.

[27] J. Gao, P. K. Agarwal, and J. Yang. Durable top-k queries on temporal data. *Proceedings of the VLDB Endowment*, 11(13):2223–2235, 2018.

[28] J. Gao, S. Sintos, P. K. Agarwal, and J. Yang. Durable top-k instant-stamped temporal records with user-specified scoring functions. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 720–731. IEEE, 2021.

[29] J. Gao, Y. Xu, P. K. Agarwal, and J. Yang. Efficiently answering durability prediction queries. In *Proceedings of the 2021 International Conference on Management of Data*, pages 591–604, 2021.

[30] M.-G. Gong, L.-J. Zhang, J.-J. Ma, and L.-C. Jiao. Community detection in dynamic social networks based on multiobjective immune algorithm. *Journal of computer science and technology*, 27(3):455–467, 2012.
[31] S. Har-Peled and M. Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 150–158, 2005.
[32] D. S. Hochbaum. Approximating covering and packing problems: set cover, vertex cover, independent set, and related problems. In *Approximation algorithms for NP-hard problems*, pages 94–143. 1996.
[33] X. Hu, S. Sintos, J. Gao, P. K. Agarwal, and J. Yang. Computing complex temporal join queries efficiently. In *Proceedings of the 2022 International Conference on Management of Data*, pages 2076–2090, 2022.
[34] M. Idris, M. Ugarte, and S. Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1259–1274, 2017.
[35] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 1–10, 1977.
[36] H. Kaplan, K. Klost, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. Triangles and girth in disk graphs and transmission graphs. In *27th Annual European Symposium on Algorithms (ESA 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.
[37] N. Kumar, L. Zhang, and S. Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *Computer Vision–ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part II 10*, pages 364–378. Springer, 2008.
[38] A. Kutuzov, M. Dorgham, O. Oliynyk, C. Biemann, and A. Panchenko. Making fast graph-based algorithms with graph metric embeddings. In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 3349–3355, 2020.
[39] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng. Facetnet: a framework for analyzing communities and their evolutions in dynamic networks. In *Proceedings of the 17th international conference on World Wide Web*, pages 685–694, 2008.
[40] G. Locicero, G. Micale, A. Pulvirenti, and A. Ferro. Temporalri: a subgraph isomorphism algorithm for temporal networks. In *Complex Networks & Their Applications IX: Volume 2, Proceedings of the Ninth International Conference on Complex Networks and Their Applications COMPLEX NETWORKS 2020*, pages 675–687. Springer, 2021.
[41] d. B. Mark, C. Otfried, v. K. Marc, and O. Mark. *Computational geometry algorithms and applications*. Spinger, 2008.
[42] T. E. Ng and H. Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 170–179. IEEE, 2002.
[43] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. *Journal of the ACM (JACM)*, 65(3):1–40, 2018.
[44] M. Overmars and J. van Leeuwen. Worst-case optimal insertion and deletion methods for decomposable searching problems. *Inf. Process. Lett.*, 12(4):168–173, 1981.
[45] M. H. Overmars. *The design of dynamic data structures*, volume 156. Springer Science & Business Media, 1987.
[46] M. Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610, 2010.
[47] P. Pope, C. Zhu, A. Abdelkader, M. Goldblum, and T. Goldstein. The intrinsic dimension of images and its impact on learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.
[48] K. Semertzidis and E. Pitoura. Durable graph pattern queries on historical graphs. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 541–552. IEEE, 2016.
[49] J. B. Tenenbaum, V. d. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.
[50] T. L. Veldhuizen. Leapfrog triejoin: A simple, worst-case optimal join algorithm. In *Proc. International Conference on Database Theory*, 2014.
[51] K. Verbeek and S. Suri. Metric embedding, hyperbolic space, and social networks. In *Proceedings of the thirtieth annual symposium on Computational geometry*, pages 501–510, 2014.
[52] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, pages 1–8, 2012.
[53] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. Lui. Diversified temporal subgraph pattern mining. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1965–1974, 2016.
[54] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. Accurate, efficient and scalable graph embedding. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 462–471. IEEE, 2019.
[55] X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao. Orion: shortest path estimation for large social graphs. *networks*, 1:5, 2010.

[56] X. Zhao, A. Sala, H. Zheng, and B. Y. Zhao. Efficient shortest paths on massive social graphs. In *7th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 77–86. IEEE, 2011.

A COVER TREE FOR BALL REPORTING QUERIES

We consider the case where the doubling dimension is constant or the expansion constant is bounded by a constant. Furthermore, we assume that the spread of the items P is bounded by a polynomial on n. Given a query item q and an error threshold ε the goal is to find a family of sets $C = \{C_1, \ldots, C_m\}$, with $m = O(\varepsilon^{-O(\rho)})$, such that $C_i \subseteq P$, $C_i \cap C_j = \emptyset$, for every item $p \in P$ with $\phi(p, q) \leq 1$, $p \in C_j$, for an index $j \leq m$, and for every point $p \in \bigcup_{i \leq m} C_i$ it holds that $\phi(p, q) \leq 1+\varepsilon$. Finally we require that the distance of any pair of items inside C_i to be at most ε. When the spread is bounded, the cover tree consists of $O(\log n)$ levels. Assume that the root has the highest level and the leaf nodes has the lowest level. Each node v in the cover tree is associated with a representative point $\mathsf{Rep}_v \in P$. For each node v in level i of the cover tree it holds that: (i) If u is another node in level i then $\phi(\mathsf{Rep}_v, \mathsf{Rep}_u) > 2^i$. (ii) If v is not the root node, it always has a parent w in level $i+1$. It holds that $\phi(\mathsf{Rep}_v, \mathsf{Rep}_w) < 2^{i+1}$. (iii) If v is not a leaf node, v has always a child w such that $\mathsf{Rep}_v = \mathsf{Rep}_w$. Assuming that the doubling dimension is ρ we have that each node v of the cover tree has $O(2^{O(\rho)}) = O(1)$ children. The same, constant bound, holds for bounded expansion constant. The standard cover tree has space $O(n)$ and can be constructed in $O(n \log n)$ time [8, 31].

Notice that every node in the lowest level contains one item from P and each item in P appears in one leaf node. Let P_v be the set of points stored in (the leaf nodes of) the subtree rooted at node v. We do not explicitly store P_v in every node v of the cover tree. Instead, for every node v we add a pointer to the leftmost leaf node in the subtree rooted at v. If we also link all the leaf nodes, given a node v, we can report all points in P_v following the pointers, in $O(|P_v|)$ time. Our modified cover tree has space $O(n)$ and can be constructed in $O(n \log n)$ time. For each node v in level i, let $r_v = 2^i$ be its separating radius and $e_v = 2^{i+1}$ be its covering radius.

Lemma A.1. *If $p \in P_v$, $\phi(p, Rep_v) < e_v$.*

Proof. It follows by induction on the level of the tree. In the leaf nodes it holds trivially. We assume that it holds for all nodes in level $i-1$. We show that it holds for every node at level i. Let v be a node at level i. By definition we have that if w is a child of v then $\phi(\mathsf{Rep}_v, \mathsf{Rep}_w) < r_v$. By the induction assumption, if $p \in P_w$ it holds that $\phi(p, \mathsf{Rep}_w) < e_w$, so $\phi(p, \mathsf{Rep}_v) \leq \phi(\mathsf{Rep}_v, \mathsf{Rep}_w) + \phi(p, \mathsf{Rep}_w) < r_v + e_w = 2^{i+1} = e_v$. □

Query procedure. Given a query point q and an error threshold ε, we start the query procedure in the modified cover tree we constructed above. In each level i we visit the nodes v such that $r_v > 1$ and $\phi(q, \mathsf{Rep}_v) \leq 1 + e_v$. Let V_i be the nodes in level i we visit such that $r_v = 1$ for $v \in V_i$. Then we consider each of the node $v \in V_i$ and we get all nodes u in the subtree of v with $r_u = \varepsilon/4$. Let C' be the set of all nodes u we found. We go through each node $w \in C'$ and we check whether $\phi(q, \mathsf{Rep}_w) \leq 1 + \varepsilon/2$. If yes, then we add P_w in C. Otherwise, we skip it.

Correctness. Let $p \in P$ be an item such that $\phi(q, p) \leq 1$. We need to show that p belongs in a set in C. Let i be the level of the node v such that $p \in P_v$ and $r_v = 1$. Since $\phi(q, p) \leq 1$ it also holds that $\phi(q, \mathsf{Rep}_v) \leq \phi(q, p) + \phi(p, \mathsf{Rep}_v) \leq 1 + e_v$. Hence, we will visit node v in the query procedure and we will add it in set V_i. Since $p \in P_v$ and $v \in V_i$, by definition, item p lies in one of the nodes w in C'. We have $\phi(q, \mathsf{Rep}_w) \leq \phi(q, p) + \phi(p, \mathsf{Rep}_w) \leq 1 + \varepsilon/2$. So we will keep w in C. Finally, notice that for each $p \in P_w$ for a node w in C, we have $\phi(q, p) \leq \phi(q, \mathsf{Rep}_w) + \phi(p, \mathsf{Rep}_w) \leq 1 + \varepsilon/2 + \varepsilon/2 \leq 1 + \varepsilon$. So the query procedure is correct.

Time Complexity. We first bound the number of nodes v we visit in level i with $r_v = 1$. Recall that we only consider v if $\phi(q, \text{Rep}_v) \leq 1 + e_v \leq 3$. Equivalently, we can think of a ball $\mathcal{B}$ of radius 3 and center q. Each node v defines a ball $\mathcal{B}_v$ with center Rep_v and radius 1. Also notice that the centers of any two balls $\mathcal{B}_v, \mathcal{B}_u$ have distance at least 1. The number of nodes we visit in level i is the same as the number of balls $\mathcal{B}_v$ that intersect $\mathcal{B}$. Using the bounded doubling dimension, it is easy to argue that ball $\mathcal{B}$ of radius 3 can be covered by at most $O(2^{O(\rho)}) = O(1)$ balls of radius 1 (a similar argument holds for bounded expansion constant). Hence, we can argue that in each level above i we only visit $O(1)$ number of nodes. So in total we visit $O(\log n)$ nodes until we reach level i with $2^i = 1$. Next, since the doubling dimension is ρ each node can have $O(2^{O(\rho)})$ children. It follows that the number of nodes we visit with $r_u = \varepsilon/4$ in the subtree of any node v (with $r_v = 1$) in C' is $O(\varepsilon^{-O(\rho)})$. Overall, the query time is $O(\log n + \varepsilon^{-O(\rho)})$.

B DYNAMIC SETTING

In this setting, we assume that we do not know the point set P upfront. We start with an empty point set $P' = \emptyset$, and some input parameters τ, ε. The goal is to construct a data structure such that, if all points are inserted and deleted according to their lifespans, it supports the following operations: i) if a point is deleted the data structure is updated efficiently, and ii) if a point p is inserted, the data structure is updated efficiently and τ-durable triangles (if any) of the form (p, q, s) are reported such that $I_p^- \geq \max\{I_q^-, I_s^-\}$ along with some τ-durable ε-triangles that contain p. We call it the DynamicOffDurable problem.

We note that the data structure we need in this dynamic setting is a dynamic version of $\mathcal{D}$. In particular, we slightly modify the standard techniques to convert our static data structure to a dynamic one with amortized update guarantees [23, 44, 45].

We call the new dynamic data structure $\mathcal{D}^{\text{dyn}}$. Let P' be the current instance of $O(n)$ "active" points. $\mathcal{D}^{\text{dyn}}$ consists of $K = O(\log n)$ subsets of points $G_1, \ldots, G_K$ such that for each $i \leq K$, $G_i \subseteq P'$ and $\bigcup_{i \leq K} G_i = P'$. For each $i \leq K$, we have a static data structure $\mathcal{D}$, called $\mathcal{D}^i$, over G_i. It holds that for each $i \leq K$, $|G_i|$ is either 2^i or 0. The total space of the data structure is $O(n \log n)$.

Assume that we have to remove $p \in P$. We identify the group i such that $p \in G_i$. We also identify the leaf node u of the cover tree $\mathcal{T}_i$ that p belongs to. Both of these operations can easily be executed in $O(\log n)$ time with some auxiliary data structures. Then we traverse $\mathcal{T}_i$ from u to the root removing p from the linked interval trees. Notice that the structure of $\mathcal{D}^i$ does not change, instead only the information stored in the nodes of the interval trees containing p are updated.

Next, assume that a new point p is inserted at time I_p^-. We first place p in a temporary min heap H with value $I_p^- + \tau$ (the time instance that p can participate in τ-durable triangles). When we reach time $I_p^- + \tau$, we derive p from H. At this point p is an active point for time more than τ. We insert p in $\mathcal{D}^{\text{dyn}}$ as follows. We find the smallest i such that $G_i = \emptyset$. We move all points $A = \bigcup_{j<i} G_j$ into G_i and construct $\mathcal{D}^i$ over $A \cup \{p\}$. At this point, we also need run a query to find all τ-durable triangles that contain p (having I_p^- as the largest left endpoint). We run the offline query $\text{durableBallQ}^i(p, \tau, \varepsilon/2)$ in each $\mathcal{D}^i$ with $G_i \neq \emptyset$. As we had in the offline case, for each i we get a set of $O(\varepsilon^{-\rho})$ canonical nodes of the cover tree. Each canonical node corresponds a ball of radius at most $\varepsilon/4$. In the dynamic case there are in total $O(\varepsilon^{-\rho} \log n)$ canonical nodes, since there are $O(\log n)$ groups. In order to find the durable triangles, we run Algorithm 1 from the offline case considering $O(\varepsilon^{-\rho} \log n)$ canonical nodes instead of $O(\varepsilon^{-\rho})$. Hence, the running time is $O(\varepsilon^{-\rho} \log^3 n + \text{OUT}_p)$, where OUT_p is the number of τ-durable triangles anchored by p (with I_p^- being the largest left endpoint) along with a number of additional τ-durable ε-triangles anchored by p. Equivalently, we can argue that OUT_p is the τ-durable triangles that p participates in at the

moment $I_p^- + \tau$ along with some additional τ-durable ε-triangles that p participates in. Finally, we re-construct $\mathcal{D}^{\text{dyn}}$ from scratch after $n/2$ updates.

Following the analysis in [23, 44, 45] and observing that each point can change at most $O(\log n)$ groups and the construction time of the offline $\mathcal{D}$ is $O(n \log^2 n)$, we have that the insertion of a point takes $O(\log^3 n)$ amortized time. The deletion takes $O(\log^2 n)$ time.

Theorem B.1. *Given (P, ϕ, I), $\tau > 0$, and $\varepsilon > 0$, ε-approximate DynamicOffDurable can be solved using a data structure of $O(n \log n)$ space, $O(\log^3 n)$ amortized update time, and $O(\varepsilon^{O(-\rho)} \log^3 n + OUT_p)$ time to report all new τ-durable triangles (along with some ε-triangles) anchored by p, where OUT_p is the output size after inserting point p, where $n = |P|$, and ρ is the doubling dimension of P.*

C EXTENSIONS

We show how we can extend our results in any ℓ_α norm and we describe how to report other patterns (except of triangles) of constant size. Furthermore, we show how to report all durable star patterns. While we only describe the results in the offline setting, all of them can be extended to the online setting using the approach as shown in Section 4.

C.1 ℓ_α metric

In order to find all τ-durable triangles in any ℓ_α metric we use a quadtree $\mathcal{T}$ instead of a cover tree over the input points P. Each node u of the cover tree is associated with a square $\square_u$. Let $P_u - P \cap \square_u$. Given a point $p \in P$ we find a set of $O(\log n + \varepsilon^{-d})$ canonical nodes C in $\mathcal{T}$ such that for every $u \in C$, the diameter of $\square_u$ is at most $\varepsilon/2$ and $||p - \square_u||_\alpha \leq 1 + \varepsilon/4$. Using the same procedure we followed for constant doubling dimensions over the canonical subsets C_p, we obtain:

Theorem C.1. *Given (P, ϕ, I), $\varepsilon > 0$, and $\tau > 0$, where P is a set of n points in $\mathbb{R}^d$ and ϕ is any ℓ_α norm, the ε-approximate DurableTriangle can be solved in $O\left(n(\varepsilon^{-d} \log n + \varepsilon^{-2 \cdot d} + \log^2 n) + OUT\right)$ time, where OUT is the number of triangles reported, satisfying $|T_\tau| \leq OUT \leq |T_\tau^\varepsilon|$.*

C.2 Other patterns

In this subsection we show how we can extend the offline algorithm to report i) durable cliques of constant size, ii) durable paths of constant size, and iii) durable k-star patters. The algorithms can also be extended to handle incremental queries, similarly to tau-durable triangles.

Cliques. Let $S \subseteq P$ be a subset of points. S is a τ-durable m-clique if i) $|S| = m$, ii) for every pair $p, q \in S \times S$, $\phi(p, q) \leq 1$, and iii) $|\cap_{p \in S} I_p| \geq \tau$. Similarly, S is called a τ-durable ε-m-clique if i, iii remain the same and for every pair $p, q \in S \times S$, $\phi(p, q) \leq 1 + \varepsilon$.

We consider that $m = O(1)$. The algorithm to report all τ-durable m-cliques is similar to Algorithm 1. The only difference is that instead of considering all pairs $C_{p,i}$, $C_{p,j}$ of the nodes in the cover tree, we consider all possible subsets of size m. Let $C_{p,j_1}, \ldots, C_{p,j_m}$ be the family of m subsets. If all pairwise distances among the representative points are at most $1 + \varepsilon/2$ then we report all m-cliques $C_{p,j_1} \times \ldots \times C_{p,j_m}$. The correctness follows straightforwardly from Section 3. In particular we report all τ-durable m-cliques and we might also report a few τ-durable ε-m-cliques. The running time is also asymptotically the same with Algorithm 1.

Paths. Let $S \subseteq P$ be a subset of points. S is a τ-durable m-path if i) $|S| = m$, ii) there is an ordering of the points such that the distance of two consecutive points is at most 1, and iii) $|\cap_{p \in S} I_p| \geq \tau$. Similarly, S is called a τ-durable ε-m-path if i), iii) remain the same, and the distance between consecutive points is at most $1 + \varepsilon$.

We consider that $m = O(1)$. The algorithm to report all τ-durable m-cliques is similar to Algorithm 1. The only difference is that instead of considering all pairs $C_{p,i}$, $C_{p,j}$ of the nodes in the

cover tree, we consider all possible subsets of size m. Let $C_{p,j_1}, \ldots, C_{p,j_m}$ be the family of m subsets. We try all possible $O(m!) = O(1)$ orderings and we check if we find an ordering $C_{p,j_1}, \ldots, C_{p,j_m}$ such that $\phi(\text{Rep}_{j_1}, \text{Rep}_{j_2}) \leq 1+\varepsilon/2, \phi(\text{Rep}_{j_2}, \text{Rep}_{j_3}) \leq 1+\varepsilon/2, \ldots \phi(\text{Rep}_{j_{m-1}}, \text{Rep}_{j_m}) \leq 1+\varepsilon/2$. If this is true then we report all m-paths $C_{p,j_1} \times \ldots \times C_{p,j_m}$. The correctness follows straightforwardly from Section 3. In particular we report all τ-durable m-paths and we might also report a few τ-durable ε-m-paths. The running time is also asymptotically the same with Algorithm 1.

k-star patterns. Let $S \subseteq P$ be a subset of points. S is a τ-durable m-star if i) $|S| = m$, ii) there is a central point $p \in S$ such that $\phi(p, q) \leq 1$ for every other $q \in S$, and iii) $|\cap_{p \in S} I_p| \geq \tau$. Similarly, S is called a τ-durable ε-m-star if i), iii) remain the same, and the distance between p and any other point in S is at most $1 + \varepsilon$.

We consider that $m = O(1)$. The algorithm to report all τ-durable m-cliques is similar to Algorithm 1. For each point $p \in P$, we run a query $\text{durableBallQ}(p, \tau, \varepsilon/2)$, but instead of querying points within distance 1 from p, i.e., points in ball $\mathcal{B}(p, 1)$, we query pints within distance 2 from p, i.e., points in ball $\mathcal{B}(p, 2)$. We need that change because p might belong to an m-star pattern S having the largest left endpoint $I_p^- \geq \max_{q \in S} I_q^-$, while not being the central point. In this case, it is always true that $S \subseteq P \cap \mathcal{B}(p, 2)$. Hence, we get $C_p = \{C_{p,1}, \ldots, C_{p,k}\}$, for $k = O(\varepsilon^{-O(\rho)})$ canonical nodes of the cover tree that approximately cover $\mathcal{B}(p, 2)$. Then we visit each node $C_{p,j}$. We initialize a counter $c = 0$. For every other node $C_{p,h} \in C_p$ we check whether $\phi(\text{Rep}_j, \text{Rep}_h) \leq 1 + \varepsilon/2$. If yes then we update $c = c + |C_{p,h}|$. In the end, if $c > m$ there exist $|C_{p,j}|$, m-star patterns to report. Hence, for each point $q \in C_{p,j}$ we report q as the central point and then we visit all nodes $C_{p,h}$ with $\phi(\text{Rep}_j, \text{Rep}_h) \leq 1 + \varepsilon/2$ to report all points in $C_{p,h}$. The correctness follows straightforwardly from Section 3 and the fact that for each p we report all m-star patterns that p belongs to (not necessarily as the central point) having the maximum left endpoint on its corresponding temporal interval. In particular we report all τ-durable m-star patterns and we might also report a few τ-durable ε-m-star patterns. The running time is also asymptotically the same with Algorithm 1.

D MISSING MATERIAL IN SECTION 5

D.1 UNION

Algorithm. In Algorithm 5 we show to how to find all (τ, κ)-UNION durable pairs. The high level idea follows from Algorithm 4. Instead of IT^{Σ}, we have the primitive data structure $\text{IT}^{\cup}$. Given a query interval I_{in} the goal is to find the interval $\hat{I} \in \mathcal{I}$ such that $|\hat{I} \cap I_{in}|$ is maximized, where $\mathcal{I} = \{I_p \mid p \in P\}$. This can be found using a variant of the interval tree $\text{IT}^{\cup}$ as follows: First, among all intervals that intersect I_{in}^- it finds the interval $\hat{I}_a \in \mathcal{I}$ with the largest right endpoint. Second, among all intervals that intersect I_{in}^+ it finds the interval $\hat{I}_b \in \mathcal{I}$ with the smallest left endpoint. Third, among all intervals that lie completely inside I_{in} it finds the longest interval $\hat{I}_c \in \mathcal{I}$. In the end, we return $\hat{I} \in \{\hat{I}_a, \hat{I}_b, \hat{I}_c\}$ with the longest intersection $|\hat{I} \cap I_{in}|$. Similarly to IT^{Σ}, the data structure $\text{IT}^{\cup}$ has space $O(n \log n)$, it can be constructed in $O(n \log^2 n)$ time, and given a query interval I_{in}, it returns $\hat{I}$ in $O(\log^2 n)$ time. Using $\text{IT}^{\cup}$, we construct $\mathcal{D}^{\cup}$ similarly to $\mathcal{D}^{\Sigma}$. The only difference is that for each node in the cover tree there exist an $\text{IT}^{\cup}$ data structure (instead of IT^{Σ}). The procedure $\text{ComputeMaxUnionD}(\text{IT}_{p,i}^{\cup}, I_{in})$ returns the interval $\hat{I}_i$ that has the largest intersection with I_{in} and its corresponding point in P lies in $C_{p,i}$.

Next, we describe the subroutine $\text{MaxIntersection}(I_{in})$. It simply considers all canonical nodes in C_p running $\text{ComputeMaxUnionD}(\text{IT}_{p,i}^{\cup}, I_{in})$ for each $C_{p,i} \in C_p$ such that $\phi(\text{Rep}_i, \text{Rep}_j) \leq 1+\varepsilon/2$. Hence $\text{MaxIntersection}(I_{in})$ visits all canonical nodes that are close to both p and q, and among these nodes $C_{p,i}$, it returns the interval $\hat{I} = \arg\max_{\hat{I}_i} |\hat{I}_i \cap I_{in}|$, i.e., the interval with the largest intersection with I_{in}. We also notice that ComputeMaxUnionD can be easily modified so that we

Algorithm 5: ReportUNIONPair($\mathcal{D}^{\cup}, p, \tau, \varepsilon, \kappa$)

$C_p : \{C_{p,1}, C_{p,2}, \cdots, C_{p,k}\} \leftarrow$ durableBallQ$(p, \tau, \varepsilon/2)$, with Rep_i denoting the representative point of the cover tree node for $C_{p,i}$, and $\mathsf{IT}^{\cup}_{p,i}$ denoting the annotated interval tree for this cover tree node;
foreach $j \in [k]$ **do**
  foreach $q \in C_{p,j}$ *in descending order of* I^+_q **do**
    $I' \leftarrow I_q \cap I_q$;
    $\bar{I} \leftarrow$ MaxIntersection(I');
    $H \leftarrow$ newHeap$(\{(\bar{I}, I', |I \cap I'|)\})$;
    $t \leftarrow 0$;
    for $h = 1 \ldots \kappa$ **do**
      $(I_x, I_y, |I_x \cap I_y|) \leftarrow$ the top element of H;
      $t \leftarrow t + |I_x \cap I_y|$;
      foreach $I_z \in I_y \setminus I_x$ **do**
        $\bar{I} \leftarrow$ MaxIntersection(I_z);
        H.insert$(\bar{I}, I_z, |\bar{I} \cap I_z|)$;
    if $t \geq (1 - 1/e)\tau$ **then report** (p, q);
    else break;
Subroutine MaxIntersection(I_{in}) **begin**
  $\mu \leftarrow -1$;
  $\hat{I} \leftarrow \emptyset$;
  foreach $i \in [k]$ **do**
    if $\phi(Rep_i, Rep_j) \leq 1 + \frac{\varepsilon}{2}$ **then** $\hat{I}_i \leftarrow$ ComputeMaxUnionD$(\mathsf{IT}^{\cup}_{p,i}, I_{in})$;
    if $|\hat{I}_i \cap I_{in}| > \mu$ **then**
      $\mu \leftarrow |\hat{I}_i \cap I_{in}|$ and $\hat{I} \leftarrow \hat{I}_i$;
  return $(\hat{I})$;

always skip I_p and I_q from the procedure of finding the interval in $\mathcal{I}$ with the largest intersection with I_{in}. From the proofs in the previous sections we have that $\phi(p, q) \leq 1 + \varepsilon/2$ while the witness set has distance at most $1 + \varepsilon$ from both p and q. Overall, MaxIntersection(I_{in}) finds an interval $\hat{I} = I_s$ such that

$$|I_s \cap I_{in}| \geq \max_{s' \in P : \phi(p,s'), \phi(q,s') \leq 1} |I_{s'} \cap I_{in}|,$$

and

$$\phi(p, s), \phi(q, s) \leq 1 + \varepsilon.$$

Finally, we describe ReportUNIONPair$(\mathcal{D}^{\cup}, p, \tau, \varepsilon, \kappa)$. After finding C_p and for each $C_{p,j}$ we visit $q \in C_{p,j}$ in descending order of I^+_q as we did in Algorithm 4. For each pair (p, q) we check, we run the greedy algorithm for the max k-coverage problem. First, we find the interval $\bar{I} \in \mathcal{I}$ that has the largest intersection with $I' = I_p \cap_q$. We create a max heap H and we insert the pair $(\bar{I}, I')$ with value $|\bar{I} \cap I'|$. Then the algorithm proceeds in κ iterations. In each iteration, it finds the pair (I_x, I_y) in the max heap H with the maximum $|I_x \cap I_y|$. I_x is an interval from $\mathcal{I}$, while I_y is an uncovered segment of I'. Hence, in each iteration, it finds the interval I_x that covers the largest uncovered

area of I'. Then we add $|I_x \cap I_y|$ in variable t that maintains the overall union the algorithm has computed. An interval I_x might split I_y into two smaller uncovered segments or into one smaller uncovered segment of I'. In each case, $I_z \subseteq I_y$ represents one uncovered segment created after adding I_x. We run MaxIntersection(I_z) and we find the interval $\bar{I} \in \mathcal{I}$ that covers the largest portion of I_z and we insert the pair $(\bar{I}, I_z)$ in max heap H with value $|\bar{I} \cap I_z|$. We repeat the same procedure for κ iteration. In the end we check whether $t \geq (1 - 1/e)\tau$. If yes, we report the pair (p, q), otherwise we skip $C_{p,j}$ and continue with the next canonical node. Overall, this algorithm gives an implementation of the greedy algorithm for the max κ-coverage problem in our setting, using efficient data structure to accelerate the running time.

Correctness. The correctness of this algorithm follows by the correctness of the greedy algorithm for the max κ-coverage problem, the correctness of MaxIntersection(I_{in}), and the correctness of Algorithm 4. For a pair (p, q) if we find that $t \geq (1-1/e)\tau$, then (p, q) is definitely an $((1-1/e)\tau, \kappa)$-UNION ε-pair. As we argued in Algorithm 4, assume that for a $q \in C_{p,j}$ we find that $t < (1 - 1/e)\tau$. Then it is safe to skip $C_{p,j}$ because there is no other (τ, κ)-UNION durable pair to report. Notice that the approximation factor for the greedy algorithm is $1 - 1/e$, so if the greedy implementation returns $t < (1 - 1/e)\tau$, we are sure that the pair (p, q) is not (τ, κ)-UNION durable. Hence, any other $w \in C_{p,j}$ with $I_w^+ < I_q^+$ will also not be (τ, κ)-UNION durable. In any case, our algorithm returns all (τ, κ)-UNION durable pairs and might return some additional $((1 - 1/e)\tau, \kappa)$-UNION durable ε pairs. Hence, it holds that $|K_{\tau,\kappa}| \leq \mathsf{OUT} \leq |K^{\varepsilon}_{(1-1/e)\tau,\kappa}|$.

Time Complexity. Next, we analyze the running time of our algorithm. As pointed out, $\mathsf{IT}^{\cup}$ is constructed in $O(n \log^2 n)$ time and it finds the interval that covers the largest uncovered area of a query interval in $O(\log^2 n)$ time. Hence, $\mathcal{D}^{\cup}$ is constructed in $O(n \log^3 n)$ time. The subroutine MaxIntersection(I_{in}) calls ComputeMaxUnionD($\mathsf{IT}^{\cup}_{p,i}, I_{in}$), $O(\varepsilon^{-O(\rho)})$ times. For each pair (p, q) we check, the subroutine MaxIntersection(I_{in}) is called $O(\kappa)$ times, while all update operations in the max heap H takes $O(\kappa \log n)$ time. For each point p we might check at most $O(\varepsilon^{-O(\rho)})$ pairs that are not reported, one for each canonical node in C_p. Overall, Algorithm 5 runs in $O(n \log^3 n + (n + \mathsf{OUT})\varepsilon^{-O(\rho)}\kappa \log^2 n)$ time.

Received December 2023; revised February 2024; accepted March 2024