International Journal of Data Science and Analytics
https://doi.org/10.1007/s41060-021-00301-0

REGULAR PAPER

Sampling and sparsification for approximating the packedness of trajectories and detecting gatherings

Sepideh Aghamolaei[1] · Vahideh Keikha[2] · Mohammad Ghodsi[3] · Ali Mohades[4]

Received: 1 July 2021 / Accepted: 10 December 2021

Abstract
Packedness is a measure defined for curves as the ratio of maximum curve length inside any disk divided by its radius. Sparsification allows us to reduce the number of candidate disks for maximum packedness to a polynomial amount in terms of the number of vertices of the polygonal curve. This gives an exact algorithm for computing packedness. We prove that using a fat shape, such as a square, instead of a disk gives a constant factor approximation for packedness. Further sparsification using well-separated pair decomposition improves the time complexity at the cost of losing some accuracy. By adjusting the ratio of the separation factor and the size of the query, we improve the approximation factor of the existing algorithm for packedness using square queries. Our experiments show that uniform sampling works well for finding the average packedness of trajectories with almost constant speed. The empirical results confirm that the sparsification method approximates the maximum packedness for arbitrary polygonal curves. In big data models such as massively parallel computations, both sampling and sparsification are efficient and take a constant number of rounds. Most existing algorithms use line-sweeping which is sequential in nature. Also, we design two data-structures for computing the length of the curve inside a query shape: an exact data-structure for disks called hierarchical aggregated queries and an approximate data-structure for a given set of square queries. Using our modified segment tree, we achieve a near-linear time approximation algorithm.

Keywords Length query · Well-separated pair decomposition (WSPD) · Aggregated query diagram · Approximation algorithms · Geographic information systems.

1 Introduction

Packedness of a curve is the maximum ratio c between the length of a curve inside any disk to the radius of that disk. Most real-world curves have this property and it relates to concepts such as the Fréchet distance between curves, the complexity of the free-space diagram, and the performance of algorithms for computing them [16]. We give the first exact algorithm for finding the minimum c, for which a given curve is c-packed, and we give improved approximation factor and time complexity for approximating the minimum c. Our approach is based on clustering the segments via well-separated pair decomposition (WSPD) [27], where the endpoints are clustered into clusters $\tau_1, \ldots, \tau_k$ such that for each pair of points (u, v), there exists a pair τ_i, τ_j, where $u \in \tau_i, v \in \tau_j$, and the distance between a point of τ_i and a point of τ_j is at least sr, and r is the maximum radius of the clusters τ_i and τ_j. Figure 13 represents an example of a WSPD pair.

✉ Sepideh Aghamolaei
aghamolaei@ce.sharif.edu

Vahideh Keikha
keikha@cs.cas.cz

Mohammad Ghodsi
ghodsi@sharif.edu

Ali Mohades
mohades@aut.ac.ir

1 Department of Computer Engineering, Sharif University of Technology, Azadi Ave, Tehran, Iran

2 The Czech Academy of Sciences, Institute of Computer Science, Pod Vodárenskou věží 2, 182 07 Prague, Czech Republic

3 Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

4 Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran

Published online: 29 January 2022

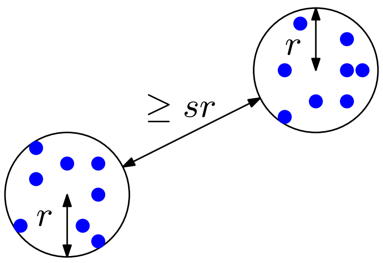

Fig. 1 A WSPD pair with separation factor s

Afterward, we compute the queries for these clusters, and use the results to approximate the results for the whole input.

Trajectories are an important type of data in geographic information systems (GIS). Recently, aggregated query diagram (AQD) [7] has been introduced as another example of data-structures for aggregation queries on trajectories: a selection based on criteria (the Euclidean distance) is applied to the initial data, as well as a summation on the length, then a maximum computation is performed on the partial results to find popular places. When a query arrives, the data-structure is used to answer the query without the need to perform any aggregations.

We generalize AQD [7] to allow multi-resolution queries instead of queries of the same shape and size. The idea is to link the partitions based on the coordinates, so we do not need to store all the details for all the possible sizes. A naive discretization would have resulted in an unbounded error (approximation factor) for the cost, since it would depend on the aspect ratio of the input curve, i.e., the ratio between the maximum and minimum distance between any pair of points.

Assuming we have a batch of windowing length queries, we discuss how to augment a segment tree by putting the queries and data in the same data-structure, which saves the time complexity of aggregation for each query. The idea of building a data-structure on a batch of parallel queries has already been used before for searching in phylogenetic trees [2], nearest neighbor search on selections of data with lower/upper bounds on each dimension under ℓ_1 distance (also known as Manhattan distance) [6], length queries [7], which are queries whose result is a function of both the input and the query: to compute the length of an input segment inside a rectangular query, the query can be shifted such that the set of intersected segments remains the same, but the length inside the shape changes. Sequential batched queries existed long before these results [3]. Our method can be seen as adding some helper queries that are the bounding box of each input segment to help compute the actual queries, which is more similar to the grid points added in [6] than the rest of the methods.

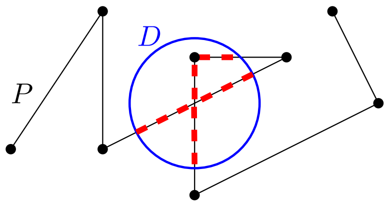

Fig. 2 A polygonal path P and a disk D with packedness about 4

1.1 *c*-packedness

In 2012, Driemel et al. [16] introduced the c-packedness property for a curve as having curve length at most cr inside any disk of radius r (Definition 1), but did not give an algorithm for computing, approximating, or deciding it. Let $P = \{P_1, \ldots, P_n\}$ be a polygonal curve, and let $\overline{P_i P_{i+1}}, \forall i = 1, \ldots, n-1$ denote the i-th edge of P.

Definition 1 (c-Packed Curve [16]) Let P be a polygonal curve. If the length of P within any disk of radius r is upper bounded by cr, P is c-packed.

The packedness of a disk D of radius r with respect to a curve P is denoted by $\gamma(D, P)$ and defined as the length of P inside D, divided by r. Formally,

$$\gamma(D, P) = \frac{\sum_{i=1}^{n-1} |\overline{P_i P_{i+1}} \cap D|}{r}.$$

Figure 2 shows an example with $\gamma(D, P) \approx 4$.

After introducing this concept, fairly good approximation algorithms for c-packed curves have been presented. Driemel et al. [16] have shown that if two curves are c-packed, a $(1+\epsilon)$-approximation of their Fréchet distance can be achieved in $O(\frac{cn}{\epsilon} + cn \log n)$ time. Bringmann and Künnemann further improved this running time to $O(\frac{cn}{\sqrt{\epsilon}} \log^2(\frac{1}{\epsilon}) + cn \log n)$ [9]. Several other studies [7,12,15,17,23,24] are only justified when the input curves are c-packed.

For axis-aligned squares instead of disks, c-packedness was discussed under the name relative-length [21] and they gave an exact $O(n^3)$ algorithm for it. Computing the minimum c for which a curve is c-packed for disks was discussed by Aghamolaei et al. [7] who formalized the problem (Definition 2) and gave an algorithm for approximating the minimum c. Approximation algorithms with factors $6+\epsilon$ and running time $O(n^{4/3} \operatorname{polylog}(n))$, 2-approximation in $O(n^2 \log n)$ time [22], and a $(288+\epsilon)$-approximation randomized algorithm with $O(n \log^2 n)$ time also exist [25].

Definition 2 (Minimum c for c-Packedness of a Curve [7]) For a given polygonal curve P with n vertices, the objective is to compute the minimum value c for which P is c-packed, in other words $\min_{\substack{\forall \text{disks } D, \\ \gamma(D,P) \leq cr}} c$.

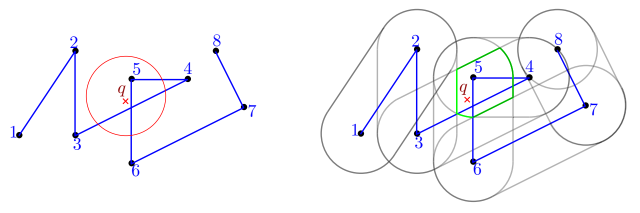

Fig. 3 A polygonal path and a disk query centered at point q on the left, and its AQD on the right. The green region is the cell containing the query point q (in red), so, it intersects with $\{\overline{P_3P_4}, \overline{P_4P_5}, \overline{P_5P_6}\}$

1.2 Aggregated query diagram (AQD)

Aghamolaei et al. [7] introduced a data structure called aggregated query diagram (AQD) for computing the length of the curve inside all translations of a query shape. The authors designed a $(1+\epsilon)$-approximation algorithm for disks by approximating a disk with a regular polygon, and a $(1+\epsilon)$-approximation algorithm for computing the c-packedness within any convex polygon (without rotation), with running times $O(\frac{\log(L/\delta)}{\epsilon}n^3)$ and $O(\frac{\log(L/\delta)}{\epsilon^{5/2}}n^3 \log n)$, respectively.

Definition 3 (Aggregated Query Diagram (AQD) [7]) For a set P of n line segments and a query shape X, a partitioning $c_1, \ldots, c_k$ of the plane with a set $O_i \subset P$ at each c_i, is called an aggregated query diagram if for any point $q \in c_i$, the query shape with representative point q intersects the subset O_i of segments, and k is minimized.

Aghamolaei et al. [7] showed the arrangement created by the Minkowski sum of the shape with the input curve gives the cells of the AQD of that curve and the Minkowski sum places a copy of the shape centered at each point of the curve. Figure 3 is an example of an AQD for the curve and disk of Fig. 2.

The number of the cells in the AQD of a convex shape X of complexity m is $O(n^2m^2)$, and its total complexity is $O(n^3m^2)$. Also, it takes $O(n^3m^2 \log(mn))$ time to construct the corresponding AQD of X.

1.3 Approximating c-packedness

Later in [22], the authors provided a 2-approximation algorithm with running time $O(dn^2 \log n)$ in $\mathbb{R}^d$, and a $(6+\epsilon)$-approximation algorithm based on WSPD [27] with running time $O((\frac{n}{\epsilon^3})^{\frac{4}{3}} \operatorname{polylog} \frac{n}{\epsilon})$ in $\mathbb{R}^2$, for computing the maximum packedness of axis-aligned squares centered at the vertices of the curve.

A generalization of the relative c-packedness is the maximum of the packedness of queries centered at a given point set S. For a set S of points and a curve P, if for any disk centered at a point of S, the length of the path inside the disk divided by its radius is at most c, the curve is an *S-relative c-packed* curve.

c-packedness can be defined for any shape X of fixed orientation and its scalings. Let $P = \{P_1, \ldots, P_n\}$ be a polygonal curve, and X be a shape with the smallest enclosing disk of diameter $2r$. If the length of P within any translation of X is upper bounded by cr, then P is c-packed.

1.4 Semi-group computations and parallelization

For a given set of objects $x_1, \ldots, x_n$, and a commutative and associative binary operator $\oplus$, the basic parallel computations are defined as follows:

– *Semi-group* The parallel semi-group computes $x_1 \oplus \cdots \oplus x_n$.
– *Parallel prefix* The parallel prefix computes all prefixes $x_1 \oplus \cdots \oplus x_i$, for all $i = 1, \ldots, n$. The diminished parallel prefix computes $x_1 \oplus \cdots \oplus x_i$ for $i = 0, \ldots, n-1$.

Examples of semi-group computations are maximum, minimum, summation, union, and intersection. Examples of parallel-prefix computations are computing the rank of elements (in sorted order), computing carries in a bit-wise summation, and some routings where there are no more (source, sink) pairs between a source and its destination(s), like broadcast that sends each data to every other processor. See [26,28] for more information.

In the massively parallel computations (MPC) [8] data is distributed among a sublinear number of machines $O(n^{1-\eta})$ each with sublinear memory $O(n^\eta)$, that process it during $O(\operatorname{poly}(\frac{1}{\eta}))$ parallel rounds and communicate with each other after each round.

The parallel version of batched queries [3] are simultaneous geometric queries in MPC. Some examples in MPC are 2-sided range queries [6] and length queries [7]. We improve the simultaneous length queries algorithm to take $O(\operatorname{poly}(\frac{1}{\eta}))$ rounds instead of $O(\log \frac{L}{\delta})$.

1.5 Contributions

– We show the lower bound on the vertex-relative c-packedness for c-packedness is 2, which matches the upper bound of the existing algorithms.
– We give a massively parallel (MPC) algorithm for S-relative c-packedness for sets S of sublinear size using $O(1)$ rounds, and a MPC approximation algorithm for c-packedness with $O(\log n)$ rounds based on WSPD.
– We give an $O(n^5)$ time exact algorithm for the minimum c-packedness. The number of events in our algorithm is $O(n^3)$, which is close to the lower bound on the time complexity of similar problems, such as maximum subarray in a matrix [29]. Some of the differences are the packedness problem asks for the subarray with the max-

imum average instead of the maximum sum, and that the packedness allows taking fractions of a segment, while the maximum subarray problem does not.

- We give a $O(n(\log^2 n)(\log^2 \frac{1}{\epsilon}) + \frac{n}{\epsilon})$ time $(4+\epsilon)$-approximation algorithm, improving the $(6+\epsilon)$- approximation algorithm, and a $O(n^2)$ time 2- approximation algorithm improving the existing $O(n^2 \log n)$ time 2-approximation algorithm.
- We give a data-structure for length queries using disks of arbitrary size, called Hierarchical Aggregated Query (HAQ) data-structure. It can be constructed in $O(n^6 \log n)$ time, using $O(n^6)$ space, and has query time $O(\log n + k)$.
- We give experimental results for packedness on small and medium-sized datasets and compare the performance of our exact algorithm with the existing relative packedness which is the current state of the art in Sect. 6. For trajectories with more vertices, we compare the approximate solution of WSPD sparsification with random sampling.

A Summary of the Results on Minimum c-Packedness Table 1 summarizes the results on minimum c-packedness and vertex-relative minimum c-packedness.

2 A polynomial-time exact algorithm for *c*-packedness

Here, we show that disks of maximal size and optimal c-packedness can be computed by checking $O(n^4)$ candidate disks.

Lemma 1 *For a polygonal curve P, one event for each continuous set of affine transformations of a query disk Q that intersects with the same set of edges E, given that the same subset F of E is used to define Q, is enough to compute the query with the maximum length.*

Proof A circle can be defined by the two endpoints of its diameter ($|F| = 2$), or three points on its boundary ($|F| = 3$). Let R_1 be the radius of the smallest disk with the same sets E and F, and let R_2 be the radius of the largest such disk. Let $D(c, r)$ be any of the disks with the same intersection set E and defined by points from F and of radius r centered at a point c, and assume $R_1(c) = \min_r D(c, r)$ and $R_2(c) = \max_r D(c, r)$. An example of two disks with the same intersection edge set E and different definition points F is given in Fig. 4.

Since the curve is polygonal, using the chord-length formula, the rate of increase in the length of a chord with distance h from c in terms of r is $\frac{\mathrm{d}}{\mathrm{d}r}\sqrt{r^2 - 4h^2} = \frac{r}{\sqrt{r^2-4h^2}} = \frac{r}{l}$, where l is the length of the chord of the disk of radius r containing a segment $s \in E$, for $r \in [R_1(c), R_2(c)]$. Let

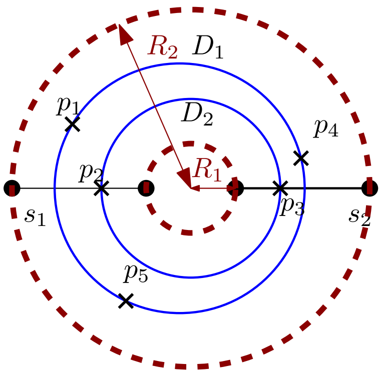

Fig. 4 Two disks with intersection sets $E = \{s_1, s_2\}$. Disk D_1 is defined by $F_1 = \{p_2, p_3\}$ and disk D_2 is defined by $F_2 = \{p_1, p_4, p_5\}$. All the disks defined by s_1 and s_2 lie between the two dashed circles

$m_s = \frac{1}{l}$. So, the packedness of $D(c, r)$ is

$$\frac{L(c) + \sum_{s\in E} m_s(r + \Delta r - R_1(c))}{r + \Delta r} = \sum_{s\in E} m_s + \frac{L(c) - \sum_{s\in E} m_s R_1(c)}{r + \Delta r},$$

where $L(c)$ is the length of the curve inside $D(c, R_1(c))$, and Δr is the changes to r. This function has its maximum in its domain at radius $R_2(c)$, which is $\sum_{s\in E} m_s + \frac{L(c)-\sum_{s\in E} m_s R_1(c)}{R_2(c)}$. Taking the maximum packedness of disks $D(c, r)$ with the same sets E and F over different points c, gives:

$$\max_c \sum_{s\in E} m_s + \frac{L(c) - \sum_{s\in E} m_s R_1(c)}{R_2(c)} = \sum_{s\in E} m_s + \max_c \frac{L(c) - \sum_{s\in E} m_s R_1(c)}{R_2(c)}.$$

Based on the definition of $L(c)$, at least one of the segments has length 0 inside $D(c, R_1)$, so

$$L(c) - \sum_{s\in E} m_s R_1(c) \le 0.$$

So, the maximum happens when $R_2(c)$ is maximized, which happens for the disk of radius R_2 with intersection set E and defined by F. □

Computing the Disk Described in Lemma 1 If smallest enclosing circle of the endpoints of the segments in F contains one point from each segment in E, we are done; otherwise, compute the smallest disk D_s intersecting at least one point of F, and call this set of points F_s. D_s can be computed by enumerating all $\binom{6}{3}$ or $\binom{6}{2}$ possible cases. Then, find the closest point of each segment in E to the center of D_s

Table 1 A summary of results on minimum c-packedness of curves

Shape	Approx.	Time	References	Relative c-Packed
Convex polygon (constant complexity)	$2+\epsilon$	$O(\frac{\log(L/\delta)}{\epsilon} n^3 \log n)$	[7]	-
Circle	$2+\epsilon$	$O(\frac{\log(L/\delta)}{\epsilon^{5/2}} n^3 \log \frac{n}{\epsilon})$	[7]	-
Square	exact	$O(n^3)$	[21]	-
d-cube	2	$O(dn^2 \log n)$	[22]	Yes
Square	$6+\epsilon$	$O((\frac{n}{\epsilon^3})^{\frac{4}{3}} \text{polylog } \frac{n}{\epsilon})$	[22]	Yes
Square	$288+\epsilon$	$O(n \log^2 n)$	[25]	-
Circle	exact	$O(n^5)$	Theorem 1	-
α-fat shapes	$\alpha^2\beta$	$T_c(n)$	Theorem 2	-
Square	2	$O(n^2)$	Section 2.1	Yes
Square, rectangle	$4+\epsilon$	$O(\frac{n}{\epsilon} \log^2 n + \frac{n}{\epsilon^{5/2}} \log \frac{n}{\epsilon})$	Section 3	Yes

L is the length of the curve. δ is the minimum distance between two points from disjoint edges. $T_c(n)$ is the time complexity of a β-approximation for c-packedness using circles

and call this set C. The smallest enclosing disk of $C \cup F_s$ is the solution and it can be computed in $O(|E|)$ time.

So, we focus on finding the sets described in Lemma 1. Let $C(u, v, w)$ denote the smallest circle containing points u, v, and w.

Algorithm 1 Enumerating The Events of Exact Minimum c-Packedness

```
1: for each $i, j \in 1, \dots, n$ do
2:     $p_i, p_j$ = the closest points of $\overline{P_i P_{i+1}}$ and $\overline{P_j P_{j+1}}$ to each other.
3:     $dp[0] = 1$
4:     for $k = 1, \dots, n$ do
5:         $p_k$ = the point $p$ on segment $s_k$ with the smallest $C(p_i, p_j, p)$.
6:         $dp[k] = \sum_{0 \le z < k,\ p_k \in C(p_i, p_j, p_z)} dp[z]$
7:         if $\forall z < k, p_k \notin C(p_i, p_j, p_z)$ then
8:             $dp[k] = dp[k] + 1$
```

Lemma 2 *The recurrence relation of Algorithm 1 finds $O(n^2)$ events for each pair of segments s_i and s_j.*

Proof We use induction on the number of edges checked so far. At step $k \le n$, the set of edges of the curve that can be used to create the events are $s_1, \dots, s_k$.

The circle determined by p_i and p_j as the endpoints of its diameter is the base case. Based on Lemma 1, since p_i, p_j and p_k are known, there are two cases:

- $C(p_i, p_j, p_k)$ determines a circle that was not given by any subset of $s_1, \dots, s_{k-1}$, i.e., it is a new event.
- $\exists k' : C(p_i, p_j, p'_k)$ determines the circle intersecting s_k, i.e., there is no circle that intersect s_i, s_j and s_k but not any of s_z, $z < k$.

This is all the possibilities of the segments being intersected, which is what the recurrence in the algorithm computes. □

Similarly, the other endpoints of the segments can be considered for removing the segment from the set of intersected segments, and the number of events is still $O(n^2)$, since the number of involved endpoints in computing the events has changed from n to $2n$.

Theorem 1 *The number of events is $O(n^4)$.*

Proof Choosing two segments s_i and s_j has $\binom{n}{2} = O(n^2)$ possibilities, and using Lemma 2, there are $O(n^2)$ events for each of them. This is $O(n^4)$. Each event is counted at most via $\binom{3}{2} = 3$ subroutines, which is $O(1)$. □

For any of the computed events, sets E and F are constructed, and the largest disk described in Lemma 1 is computed, which takes $O(n)$ time per event, and the algorithm runs in $O(n^5)$ time. Assuming the packedness of each event is computed at its construction time, and only the maximum is stored, the space complexity of the algorithm is $O(n)$.

2.1 Vertex-relative *c*-packedness of squares

Corollary 1 *(of Lemma 7 in [22]) Vertex-relative c-packedness is at most twice c-packedness. There is an optimal square H^* of side length r defined by the vertices of the curve, so, there is a square H with side length $2r$ centered at a vertice of the curve, such that H covers H^*.*

Algorithm 2 is a modification of the 2-approximation algorithm of [22] that uses Corollary 1 to find the events and

Algorithm 2 Relative c-Packedness for Squares

```
Require: A curve $P = \{P_1, \dots, P_n\}$
Ensure: The minimum $c$ for which $P$ is vertex-relative $c$-packed
1: $L_x$ = the edges of $P$ sorted on $x$, $L_y$ = the edges of $P$ sorted on $y$.
2: for $i = 1, \dots, n$ do
3:     $L$ = merge $L_x$ and $L_y$ based on their $\ell_\infty$ distance to $p$
4:     Traverse $L$ and update the length $\ell$ and side length $r$ at each point.
5:     $c = \max(c, \frac{\ell}{r})$
```

computes vertex-relative c-packedness, which improves the running time to $O(n^2)$. ℓ_∞ distance of a vector is the maximum of the absolute values of its coordinates.

Merging L_x and L_y

The list L_x is the sorted list of points of P based on their first coordinate. L_y is the list of the points of P based on their second coordinate. The goal is to compute the list L that is sorted based on the distances of the points of P to a point $p = (p_x, p_y)$.

Partition each of the sets L_x and L_y into two sets based on whether their coordinates are at least as much as p or not, call these sets: $L_{x,x \geq x_p}$, $L_{x,x<x_p}$ and $L_{y,y \geq y_p}$, $L_{y,y<y_p}$. We explain how to merge $L_{x,x \geq x_p}$ and $L_{y,y \geq y_p}$, and the rest of the cases are similar. Scan both lists simultaneously, add the element $q = (q_x, q_y)$ with the minimum distance to p from the two list and advance the pointer of that list, so, $q = \arg\min_{q \in L_x \cup L_y} \max(|q_x - p_x|, |q_y - p_y|)$. Since the lists L_x and L_y each contain a copy of the points, only keep the first occurrence of q in the list.

Proof Sketch

The list L is sorted based on the ℓ_∞ distances from p, since increasing the side length is a monotone movement in the directions of the axes, so the intersection of the square with a line through its center and parallel to each axis is also monotone. A similar idea was used in [21], but the algorithm was different. By scaling a square without changing the set of intersected segments or its center, the length of the curve inside it increases by a constant factor (similar to disks as described in Lemma 1) So, updating the lengths when reaching a new point takes $O(1)$ time and the algorithm takes $O(n^2)$ time.

2.2 Packedness using fat shapes

A shape enclosed inside two concentric disks of radii ρ and $\alpha\rho$ is called α-fat [5]. Approximating an α-fat shape with the disk of radius ρ in this definition, gives an α^2-approximation for minimum c-packedness:

Theorem 2 *The packedness of the smaller disk in the definition of an α-fat shape is an α^2-approximation for the packedness of the shape.*

Proof Let P be a c-packed curve, and D be the smaller disk of the optimal translation of query Q. Let l be the length of P inside Q, and r be the radius of Q. Then, l is at least as much as the length of the curve inside D, which is $c\rho$, and the radius of Q is at most $\alpha\rho$. So, the approximation factor is α^2: $c\rho \leq \gamma(Q, P) \leq c\alpha\rho$, $\rho \leq r \leq \alpha\rho \Rightarrow \frac{c}{\alpha} \leq \frac{l}{r} \leq c\alpha$. □

3 An approximation algorithm for *c*-packedness for squares and rectangles based on WSPD

A pair of sets (A, B) of points are s-separated [27], if disks C_A and C_B of radius ρ containing the bounding boxes of A and B fits inside them and the distance between C_A and C_B is at least $s\rho$. A well-separated pair decomposition (WSPD) [27] of a point set S is a set of s-separated pairs $\{(A_i, B_i)\}_{i=1}^m$ from S, such that for any points $p, q \in S$, there is exactly one index i such that $p \in A_i$ and $q \in B_i$, or $p \in B_i$ and $q \in B_i$. The size of a WSPD is $m = O(s^2 n)$.

We say a square is defined by a WSPD pair (A_i, B_i) if it is the smallest square centered at a point of A_i and covers the points of B_i.

Lemma 3 *The relative packedness of the squares defined by WSPD pairs is a $(2+\epsilon)$-approximation for the relative packedness and there are $O(n/\epsilon^2)$ squares.*

Proof For a square D^* with optimal relative packedness centered at p and with q on its boundary, there is a s-separated pair (A_i, B_i) with $p \in A_i, q \in B_i$ or $p \in B_i, q \in A_i$. Based on this property of WSPD, half the side length of D^* is $d \geq s\rho$. Consider the smallest square D centered at p and containing $A_i \cup B_i$; the half of side length of this square is at most $4\rho + d$. Let ℓ be the length of the curve inside D^*. Then, the packedness of D is: $\gamma(D) \geq \frac{\ell}{4\rho+d} \geq \frac{\ell}{4d/s+d} \geq \frac{\ell}{d/2}\frac{1}{8/s+2} \geq \gamma(D^*)\frac{2s+8}{s} = \gamma(D^*)(2+\frac{8}{s})$. For $s = \frac{8}{\epsilon}$, this is a $(2+\epsilon)$-approximation. □

The packedness of a curve with rectangle queries is an upper bound on the packedness of that curve with square queries, since the set of squares is a subset of the set of rectangles. By substituting squares with rectangles in Lemma 3, the result holds for rectangles as well. The reason is that the aspect ratio of rectangles built on points of a WSPD pair is $O(s)$. Note that the results do not hold for arbitrary curves, for example a space-filling curve that almost covers the area of a rectangle has an unbounded approximation factor for squares. More specifically, a $a \times b$ rectangle with $a \geq b$ might contain curve length $f(b)$, while a bounding square has size $a \times a$, resulting in packedness $f(b)/a$ for the bounding square, while any square inside the rectangle has packedness

$f(b)$, resulting in an unbounded approximation factor for a small enough a.

A corollary of Lemma 3 is that by preprocessing the length queries computed on rectangles defined by WSPD pairs, a rectangular query defined on two points of this point set can be answered by finding the WSPD pair containing those two points and reporting the result, and the approximation factor is the same as the approximation factor of computing the c-packedness.

3.1 Approximate length queries for squares and rectangles

We first break each segment between each endpoint and the closest query side, resulting in at most $3n$ segments. Segments with both endpoints on queries (set W) and ones with at least one of their original endpoints (set M) are solved using different methods. If the original endpoints of a segment lie on the boundary of queries, it is placed in W. So, $|W| \leq n$ and $|M| \leq 2n$. The rest of this section explains the algorithms for computing M, W and length queries on them.

Line-Sweeping

A line-sweeping algorithm [14] has a set of event points stored in an event queue, and a status data-structure. The event points are traversed in sorted order and at each event point, the status is updated, which can result in adding points to the event queue.

We use two line-sweeping algorithms, one from left-to-right, and the other from bottom-to-top.

The event queue consists of the endpoints of the interval of the projection of the query rectangles on the x-axis (for the left-to-right sweep) and on the y-axis (for the bottom-to-top sweep) and the endpoints of the interval of the projection of the edges of the curve on the axes. Since there are n segments and m queries, the number of events is $m + n$.

The status data-structure stores the set of edges of the curve that are intersected by a rectangle edge at query edge events for the first or last time since their endpoints, so that no other intersecting query edge event has happened since their endpoint event. The query edges are the set of vertical query edges in the left-to-right sweep, and the set of horizontal ones in the bottom-to-top sweep. Computing the last query side visited before an endpoint event is easy, since it only requires the last status. The status has at most m segments at each time (event) and those segments are perpendicular to the sweeping direction. Note that we do not need the order of edges of the curve that intersect the sides of query shapes, and no new events are added at the intersections of an edge with a query shape.

At each edge event, we check if the interval built on the events of a segment and the interval built on the events of a rectangle query in the status intersect, then, we check if they actually intersect and check if this is the first time they intersect. Checking if an endpoint was already separated from the segment can be easily done by marking the endpoints of segments.

At each query event, we update the status to include the last seen side of a query rectangle. Updating the status tree takes $O(\log m)$ time, querying the tree takes $O(\log n)$ time, if a 1D segment tree is used as the status data-structure. Therefore, the overall time complexity of the algorithm is $O((n + m) \log n)$.

Processing M:

For segments in M, we keep the endpoint p that is not on a query shape and remove the rest of the segment, and assign a weight $w(p)$ to p which is equal to the length of the removed segment. Note that if a segment of M is intersected by a rectangle query the segment falls completely inside the query.

Then, we use an axis-parallel windowing query [14] to compute the sum of the weights of the points inside each query.

Processing W: A rectangular windowing query asks for the set of segments that intersect with a given rectangle or fall inside it. Building a 2D segment tree on W is enough to answer rectangular windowing queries, since the problem of finding intersecting segments with a set of rectangle queries on that set reduces to computing the intersections between the set of axis-aligned bounding boxes of the segments of W with the query rectangles: first, search the range x of the query in the first dimension of the tree, then, search the range y of the query on the second dimension of the tree, for the first range. Reporting these intersections would take time linear in the number of intersected segments.

If a segment s falls inside the query rectangle Q, then the bounding box of s also falls inside Q, and it is reported because its x and y ranges are a subset of the x and y ranges of Q. For a segment s that intersects with Q, its bounding box also intersects with Q, and vice versa, because if s is axis-aligned, then its bounding box is a segment and lies on the boundary of Q, otherwise, its not parallel to the boundary edges of Q and s will intersect Q somewhere in the intersection of their x and y ranges.

Windowing query on x or y is an axis-aligned strip, on both x and y it becomes a rectangle, on x, y, $\Delta\theta$ (θ is the inclination angle of the segment) it can detect intersections of the rectangle with segments of slope $\theta \pm \Delta\theta$, and on x, y, $\Delta\theta$, Δr (Δr discretizes the perimeter into segments of length $\epsilon \Delta r$) has the shape of a strip inside the rectangle. To handle rectangle queries, in addition to x, y, $\Delta\theta$, we need another dimension Δr in the segment tree.

Since we only need the first two dimensions to answer rectangular length queries, we group the last two dimensions. Also, after querying the first two dimensions, we can discretize θ based on $x + y$. Similarly, we can discretize the intersection with the boundary based on x, y, θ (the strip

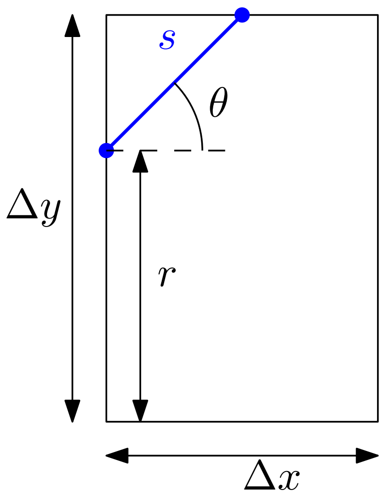

Fig. 5 A segment defined by its bounding box, inclination angle (slope), and its offset

inside the rectangle) for a given ϵ, because the length of the query would be at least $\max(x, y)$ because of another segment in that cell, otherwise the segment tree would have divided the cell. Because of the discretizations $\epsilon \Delta\theta$ and $\epsilon \Delta r$ on the last two dimensions, their trees have $O(\mathrm{poly}(\frac{1}{\epsilon}))$ elements instead of $O(n)$ elements. This is similar to type B queries in [22].

Lemma 4 *Given ranges on $[x_1, x_2]$, $[y_1, y_2]$, θ, r for a segment s, where $[x_1, x_2]$ and $[y_1, y_2]$ are the ranges for the coordinates of the bounding box of s, θ is the inclination angle and r is the offset inside the rectangle x, y, it is possible to compute the length of s. If these values are approximated by factors α_i, $i = 1, \ldots, 4$, the approximation factor for the length of s is $\alpha_1^2\alpha_2^2(1 + \alpha_3 + \alpha_4 + \alpha_3\alpha_4)$.*

Proof The first two ranges $[x_1, x_2]$, $[y_1, y_2]$ define a rectangle $[x_1, x_2] \times [y_1, y_2]$. The third dimension ($\theta$) defines a set of infinitely many parallel line segments. The last dimension (r) defines the intersection of the segment with the rectangle, which uniquely defines the segment. Figure 5 represents these parameters and the segment defined by them.

The length of the segment, assuming it is placed as shown in Fig. 5, is

$$\frac{\Delta y - r}{\Delta y}\sqrt{(\Delta x)^2 + (\Delta x \cot(\theta))^2}$$
$$= \frac{\Delta y - r}{\Delta y}\Delta x\sqrt{1 + \cot(\theta)^2}.$$

Applying the approximation factors of the parameters gives the upper bound on the approximation factor of the length:

$$(\alpha_2^2 + \alpha_4)(\alpha_1^2)(1 + \alpha_3) \leq \alpha_1^2\alpha_2^2(1 + \alpha_3 + \alpha_4 + \alpha_3\alpha_4).$$

□

For $\alpha_1 = \alpha_2 = 1, \alpha_3 = \alpha_4 = \epsilon$, the approximation factor of Lemma 4 is $1 + 4\epsilon$, so we use $\epsilon/4$ in the input algorithm, which only changes the time and space complexities by a constant factor.

A segment in W has the property that its endpoints lie on query edges. Several cases can happen for a segment of W intersecting a set of axis-aligned rectangles:

- Case I: The segment intersects two opposite edges of the same query shape. In this case, we only need to know the slope of the segment to compute the length that falls inside the query. Using a $(1 + \epsilon)$-approximation for the slope results in a $(1+O(\epsilon))$-approximation for the length.
- Case II: The segment intersects two consecutive edges of the same query shape. In addition to the slope, we also need the intersection point of the segment with one of the boundaries to compute the length of the segment. Using a $(1+\epsilon)$-approximation for the slope and the intersection point with the perimeter of the shape gives a $(1 + O(\epsilon))$-approximation for the length.
- Case III: The segment intersects two edges from different query shapes. Based on the elementary intervals of the segment tree, which correspond to two consecutive coordinates of the segments stored in the tree, this case can be partitioned into a set of rectangles where both endpoints lie on the sides of a rectangular cell of the tree. Such a partitioning is automatically computed when elementary intervals (the 2D intervals in the lowest level of the tree) are constructed during the construction of the tree.

Lemma 5 *A segment tree built on the query squares and the segments of W with dimensions of the coordinate system (x, y) partitions the input into the 3 cases mentioned above.*

Proof A segment tree on a set of segments partitions those segments. The 2D segment tree also partitions the squares. It remains to prove a query contains all the segments that intersect it in consecutive cells of the tree. A segment is continuous on its first coordinate x, so it falls inside adjacent cells in the first dimension. In the second dimension (coordinate y), for each interval x, both the query (square or rectangle) and the segment are continuous. The aggregated trees on y connect the adjacent cells in the y dimension. Then, the aggregated trees on the first dimension (x), concatenate the consecutive x values. Since both the coordinates of queries and the segments are in the tree, these sums include the cost of all queries. □

A set of examples for these cases are shown in Fig. 6.

The intervals on the lengths of these segments are computed by the algorithm. Assume $\epsilon = 0.1$ and the boundary is traversed in counter-clockwise order starting from the leftmost lowermost corner in the last dimension of the tree. The lengths stored in the tree are as follows:

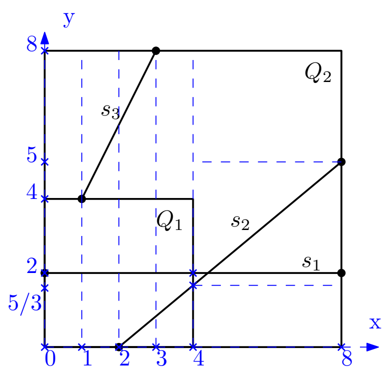

Fig. 6 Examples of different cases of segments of W with respect to square queries

- s_1: This is an example of case I. With inclination angle 0, s_1 intersects with intervals $[0, 4]$, $[4, 8]$ on the x-axis. The y intervals (in the segment tree) for these intervals are $[2, 4]$, $[2, 5]$. Then, the inclination angle (θ) is used, which is $[0, 0.1]$ for all of these intervals. s_1 intersects the boundary of all these cells with offset $[0, 0.1]$. For offset 0, the lengths stored at these cells are $\approx [4, 4.005]$, $[4, 4.011]$ and the algorithm gives the range $[8, 8.061]$ for the length of s_1.
- s_2: It has two points on consecutive edges of the same query (case II) and inclination angle $\tan^{-1}(5/6) \approx 0.695$. The x intervals are $[2, 4]$, $[4, 8]$. The y intervals are $[0, 2]$, $[5/3, 2] \cup [2, 5]$. The range for θ is $[0.6, 0.7]$. The offset ranges are $[0, 0.1]$, $[0.1]$, $[4.4, 4.5]$. The lengths are $\approx 2.603, 0.521, 4.686$, which sum to ≈ 7.81.
- s_3: s_3 has its two endpoints on different queries, and the cells of the tree used to compute the length are $[1, 2] \times [4, 8]$ and $[2, 3] \times [4, 8]$. The inclination angle is $\tan^{-1}(4) \approx 1.326$, which is in the range $[1.32, 1.33]$. The offsets are $[0, 0.1]$. So, the lengths are $\sqrt{5}$, $\sqrt{5}$.

Lemma 6 *Each non-empty elementary 2D interval (cell of a leaf) of size $x \times y$ in our segment tree contains at least* $\max(x, y)$ *length of the curve.*

Proof An elementary interval contains two endpoints in each dimension, so if it is non-empty, there is at least one segment that covers that range. The length of that segment is at least as much as the length of the range. So, the length inside each elementary cell is at least $\max(x, y)$. □

The idea of discretizing θ already existed in spanners like Θ-graphs and Yao-graphs [27], ϵ-kernels [4] and range queries [6].

Lemma 7 *Discretization on $\Delta\theta\epsilon/4$ and $\Delta r\epsilon/4$ gives a $(1 + \epsilon)$-approximation for $\Delta\theta$ and Δr.*

Proof We partition the segments W based on their inclination angle (θ) into a set of groups $[2(i-1)\pi\sqrt{\epsilon}, 2i\pi\sqrt{\epsilon}]$, for $i = 1, \ldots, \lceil \frac{1}{\sqrt{\epsilon}} \rceil$, in which the difference between different angles in a group is at most $2\pi\sqrt{\epsilon}$. Each segment with inclination angle θ with endpoints on a strip of width Δr has length $\frac{\Delta r}{\cos\theta}$. So we can get a $(1 + \epsilon)$-approximation of the total length in any strip (see Lemma 4). □

We preprocess W into a stabbing windowing datastructure on $(x, y, \Delta\theta, \Delta r)$ for counting queries [14]. Assume it is a 4D segment tree, where first the last two dimensions are aggregated, then the nodes of the first two are augmented by the prefix sums of the leaves of their subtrees.

Lemma 8 *A 4D segment tree with aggregated queries can be obtained from a segment tree by storing $O(1)$ data in each node.*

Proof To build such a tree, first build the 4D segment tree, then aggregate the results in the subtrees. Computing the sum of all the segments with less y than y_i, for each y_i from the input gives the aggregation for the y dimension. This can be done by a dynamic program on the tree, where the sums are computed from leaves to root: each internal node takes the values of its children, sums them with its own value, and sends the result to its parent.

For the x dimension, each internal node receives a tree instead of a single value, and has to sum those values. Since the values of y in the subtrees are a subset of values of y in their parent node, this can be done recursively. The aggregation visits every node of the tree at most once.

Each square query is a range on x and a range on y. At query time, the differences of the values of the ranges in the tree give the solution. This is possible because rectangular ranges map to subtrees with consecutive leaves (see Lemma 5). □

It is also possible to use a hash function for $(\Delta\theta, \Delta r)$ after the initial construction (before answering queries) to simplify the algorithm, which also increases the complexity by converting $\log \frac{1}{\epsilon}$ factors in the query time to $\frac{1}{\epsilon}$ factors in the preprocessing time or space.

To answer length queries, aggregate on the last two dimensions, by computing the weighted sum using m_s described in Lemma 1.

Theorem 3 *The sum of the length queries for M and W gives a $(1+\epsilon)$-approximation of the length queries in $O(\frac{n}{\epsilon} \log^2 n + \frac{n}{\epsilon^{5/2}} \log \frac{n}{\epsilon})$ time.*

Proof Lemma 4 shows approximating the parameters x, y, $\Delta\theta$, Δr with factors $1, 1, 1 + \epsilon/4, 1 + \epsilon/4$ gives a $(1 + \epsilon)$-approximation for the length, and Lemma 7 proves the discretization used in the algorithm achieves these approximation factors.

Lemma 5 shows an aggregated segment tree, gives an exact solution for rectangular range queries with coordinates from the input segments. Lemma 8 proved this tree can be constructed from a segment tree by adding sum of subtrees to the internal nodes of the tree. The last two dimensions have $O(1/\epsilon)$ values, because they discretize $\pi\sqrt{\epsilon})$ (for $\Delta\theta$) and $x+y$ (for Δr) by ϵ and $\max(x, y)(\epsilon/4)$ (based on Lemmas 4,6), respectively.

Dimensions x and y each have $O(n/\epsilon)$ leaves, the other two dimensions have $O(1/\sqrt{\epsilon})$ and $O(1/\epsilon)$ leaves, since we have $O(\frac{n}{\epsilon})$ queries and an input of size $n + O(\frac{n}{\epsilon})$. The last two dimensions are aggregated before query time, which takes $O(\frac{1}{\epsilon^{3/2}})$ time. The time complexity of building and aggregating a segment query in this segment tree with aggregations is $O(\frac{n}{\epsilon}\log^2 n + \frac{n}{\epsilon^{5/2}}\log\frac{n}{\epsilon})$. □

4 MPC algorithms for *S*-relative *c*-packedness

In this section, we give parallel equivalences of the algorithm described in Sect. 3.1 and Algorithm 2.

4.1 An approximation algorithm for *c*-packedness of squares

Algorithm 3 is the parallel version of the algorithm described in Sect. 3.1 in the PRAM model. Computing the WSPD takes $O(\log n)$ time in PRAM [11] and building a segment tree and computing a set of $O(n)$ queries takes $O(1)$ time in BSP and $O(n\log^2 n)$ work [18]. The algorithm also works in MPC with slowdown 2, so its round complexity is $O(\log n)$ [20].

Algorithm 3 Approximate Parallel S-Relative c-Packedness for Squares and Rectangles (The algorithm of Sect. 3.1 in PRAM)

```
Require: A curve $P = \{P_1, \dots, P_n\}$
Ensure: The minimum $c$ for which $P$ is vertex-relative $c$-packed
1: $T$ = Build a WSPD in PRAM and for each edge of the WSDP, build
   an axis-aligned rectangle with that edge as its diameter.
2: Build the elementary intervals of the segment tree from Sect. 3.1
   and compute the length query for them.
3: Compute the sum of the values in the leaves of the tree to compute
   the values of rectangles covered by the internal nodes of the segment
   tree.
4: Compute the queries $T$ using this segment tree by simulating the
   sequential algorithm.
5: For all queries $T$, divide the length (value) of the query by the side
   length of the query shape to compute its packedness.
6: return The maximum packedness from the previous step.
```

An algorithm for segment tree and one query with $O(1)$ rounds exists [18], which is the simulation of the BSP algorithm in MapReduce [20]. However, for T queries that request the same block, the queries need to be managed, which adds another factor $O(\log_m |T|)$, where m is the memory of each machine.

To build a 2D segment tree (Line 3), a parallel semi-group algorithm can be used in each dimension, since summation is a semi-group. At query time, the number of blocks that need to be summed in order to compute a rectangular query is $O(\log^2 n)$. So, the total communication and work required to compute all queries T is $O(|T|\log^2 n) = O(n\log^2 n)$. So, it is near-linear and satisfies the space constraints of MPC.

4.2 An exact MPC algorithm for *S*-relative *c*-packedness

The parallel algorithm can only compute the packedness of $O(n^\eta)$ shapes at each round. The naive algorithm that sends the query shape to all the machines has communication complexity $O(n)$ and round complexity $O(\frac{1}{\eta})$, since routing (broadcasting) also requires using parallel prefix. This is because for broadcasting, n copies need to be created, which do not fit inside the memory of one machine.

Algorithm 4 is the parallel version of Algorithm 2.

Algorithm 4 Parallel S-Relative c-Packedness for Squares (Algorithm 2 in MPC)

```
Require: A curve $P = \{P_1, \dots, P_n\}$
Ensure: The minimum $c$ for which $P$ is vertex-relative $c$-packed
1: Sort the edges on their $x$ to build $L_x$, and on their $y$ to build $L_y$.
2: Run a parallel prefix for each point $P_i \in S$ to compute the packedness
   of squares centered at $P_i$ and through one of $P_j$, $j = 1, \dots, n$ on
   their boundaries.
3: Run a parallel semi-group to find the maximum.
```

It uses two sortings, $|S|$ simultaneous parallel-prefix computations and a semi-group, each of which take $O(\frac{1}{\eta})$ rounds for $|S| = O(n^\eta)$ [19,20], so the algorithm also takes $O(\frac{1}{\eta})$ rounds.

5 A data-structure for exact length queries

The results of [7] allow querying translations of the query shape Q. We also allow querying all scalings with scale factor at least one (enlargements) of Q. Using the events for the candidate values of the radii of the disks with maximum packedness, we build a set of AQDs for each disk of that radius. We call this data-structure *hierarchical aggregated query (HAQ)* data-structure (see Algorithm 5).

During query time, the AQD built for the smallest radius that is greater than or equal to the radius of the query shape is used (see Algorithm 6).

Theorem 4 *The HAQ for disks (Algorithms 5, 6) can be constructed in $O(n^6\log n)$ time using $O(n^6)$ space, and the*

Algorithm 5 Building The Data-Structure HAQ for Disks

```
Require: A polygonal curve $P$
1: $E$ = the set of radii of the events for the $c$-packedness of $P$ for disk queries
2: Build a balanced search tree $T$ on $E$
3: for $r \in E$ do
4:     Build an AQD for radius $r$ and store it at the node with number $r$ in $T$.
5: return The augmented tree $T$.
```

Algorithm 6 Length Query using HAQ for Disks

```
Require: An HAQ data-structure, a query disk $Q$
1: $v$ = Search HAQ to find the smallest radius that is greater than or equal to the radius of $Q$.
2: In the AQD of node $v$, find the center of $Q$ and compute the length.
```

query time is $O(\log n + k)$, where k is the number of intersected segments with the query shape.

Proof The size of an AQD of a polygonal curve with disk queries is $O(n^2)$. Based on Theorem 1, there are $O(n^4)$ different radii. So, the size of the data-structure is $O(n^6)$. The construction time of AQD is $O(n^2 \log n)$, so the construction time of HAQ is $O(n^6 \log n)$.

The query time consists of the time required for finding the value of r, which is the first level of tree and takes $O(\log n)$ time. Then, a query is made to the AQD for radius r, which takes $O(\log n+k)$, since the intersection events do not change between the radii in HAQ and the set of intersected segments remains k.

Algorithm 6 computes an exact solution, since the query shape falls inside the query shape used in the construction of the AQD. □

Given the set of events for a shape other than a disk, by replacing the AQD for disks with AQD for polygons, the HAQ data-structure extends to convex polygons. The complexities change according to the time and space complexity of the data-structure.

6 Empirical results

Our results contain improvements on the time complexity, approximation factor, and applicability to big data. We focus on the approximation factor and perform the experiments on small- and medium-size trajectories. Some of the ideas, such as WSPD sparsification and discretization have been used in different algorithms, where the difference is only in the running time. To isolate the effect of each method, we use the same length computation code and the differences are only in the set of queries for each method.

Figures were drawn using TikZ package on the output of the codes. The implementation language is C++. Experiments were done on a machine with 4 GB RAM and using one processing core. In all our experiments, we only consider the latitude and longitude of the points. Some modifications on the data were done using Microsoft Excel.

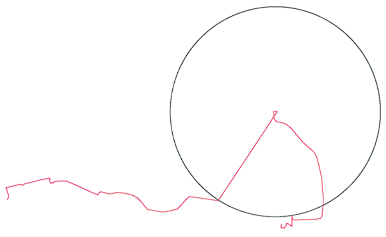
Fig. 7 The disk with maximum relative packedness on trajectory 46 of Go!Track GPS dataset

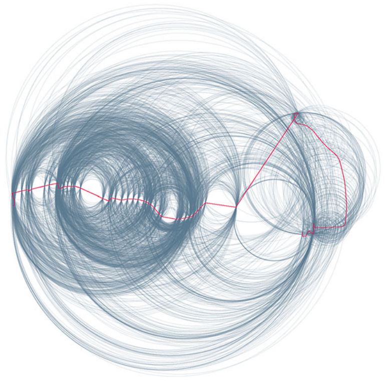
Fig. 8 A random sample of disks checked by our exact algorithm on trajectory 46 of Go!Track GPS dataset

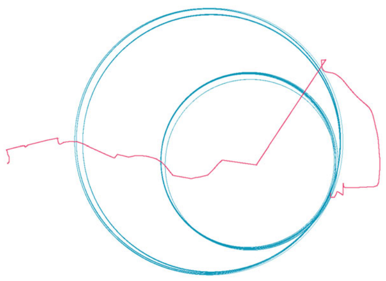
Fig. 9 Top 100 disks based on packedness on trajectory 46 of Go!Track GPS dataset

We define the packedness of a rectangle as the ratio of the curve length inside it to its average side length.

6.1 Exact packedness and relative packedness

In this experiment, we use GPS trajectory data from Go!Track application published as a dataset [13] which is available in UCI Machine Learning Repository [1]. We use trajectory 46 with 110 vertices.

Our exact algorithm reports the packedness of the curve equal to $c \approx 132.230$ and it takes less than 4 minutes to compute this. The relative packedness (which is a 2-approximation of c) is ≈ 119.112. So, here the observed

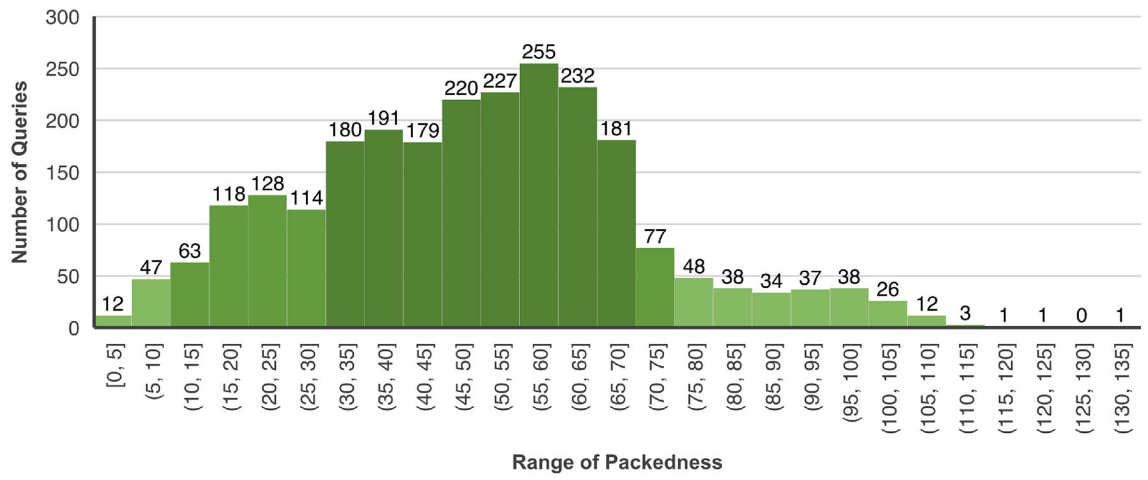

Fig. 10 The distribution of packedness of all queries used by the exact algorithm

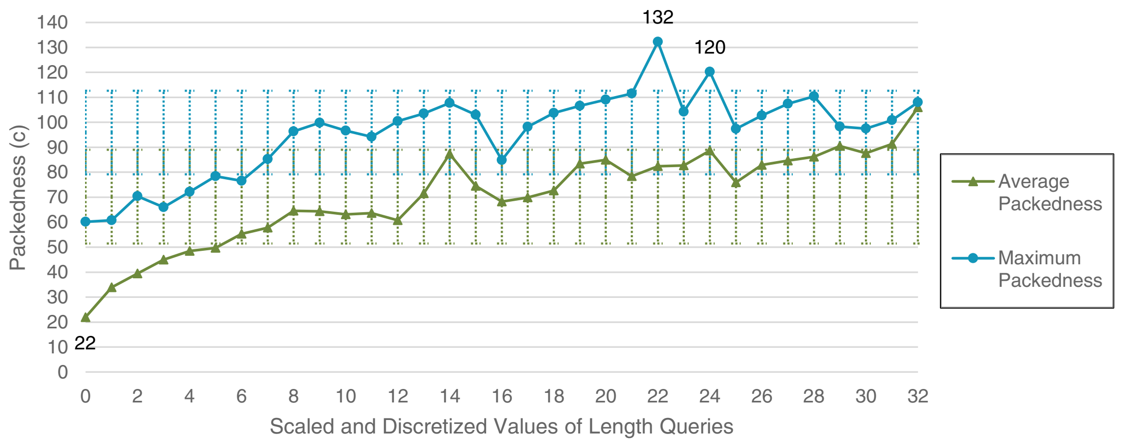

Fig. 11 Packedness in terms of the curve length inside each query shape (value of length query)

approximation factor is ≈ 1.11. Figure 7 shows the disk with maximum relative packedness. Figure 8 shows a random sample of the possible disks with sampling ratio 1%. Figure 9 shows the top 100 disks based on packedness. Both Figs. 7 and 9 seem to show disks enclosing curve length close to their diameters, but the curve length inside each of the disks in Fig. 9 seems to be more. The steps seem to be shorter at the beginning of the trajectory where more circles (queries) were chosen (Fig. 8).

The diagram of Fig. 10 shows the distribution of packedness of queries. Figure 11 shows the average packedness and maximum packedness in terms of the value of length queries, after binning the values of candidate disk queries from our exact packedness algorithm. Most queries have packedness around 80 ± 30. Noises in GPS data can be measurement errors that happen for small movements and some jumps where a point far from the actual trajectory is added to it. For example, a car in traffic makes tiny movements, each of which adds a measurement error which aggregates to a large amount compared to the same car traveling the same distance with constant velocity. Jumps increase the maximum packedness, but the average packedness is less affected by them.

6.2 Approximate packedness of GPS trajectories

On large data sets, algorithms with running times more than $n^{1+\rho}$ for a constant $\rho \in (0, 1)$, become inefficient. Using WSPD pairs as the sparsification method, we decrease the number of candidate queries for the one with the maximum packedness.

First, we use curve 20090520231518 from GeoLife GPS Trajectories dataset [30–32] which has 16480 vertices. Computing the exact packedness of this curve requires checking $\binom{32960}{3} = 745, 832, 508, 960 \approx 10^{12}$ circles using the exact algorithm (Theorem 1).

Since small changes in the trajectory are not visible in a low-resolution drawing, we also give the packedness of

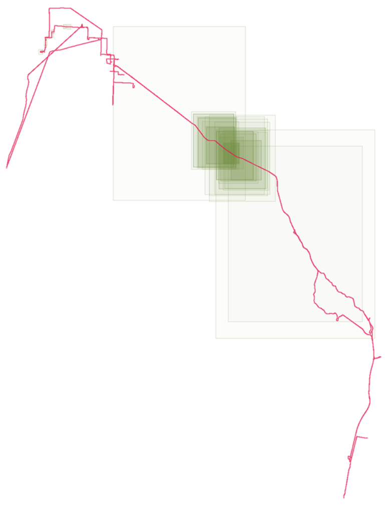
Fig. 12 Packedness of the curve using random queries

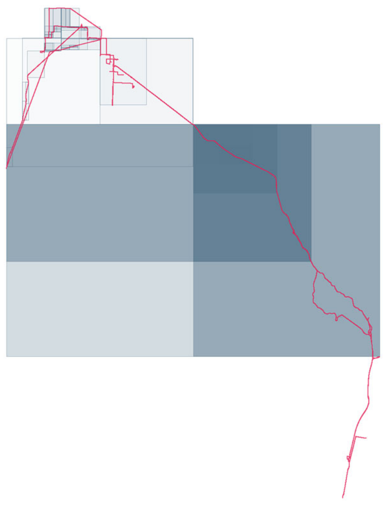
Fig. 14 A trajectory (in red) and a selection of non-empty rectangles. Darker regions contain more path length

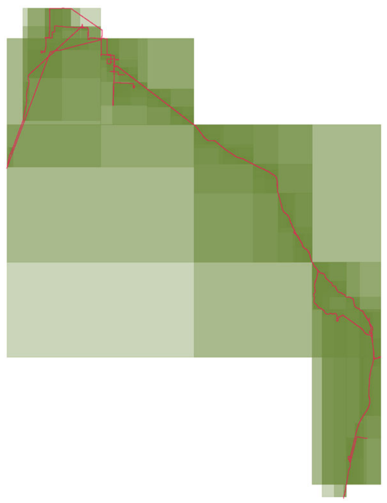
Fig. 13 A trajectory (in red) and candidate rectangles (using WSPD) for computing its packedness (in green). Darker regions are tested more often for finding the rectangle with the maximum packedness

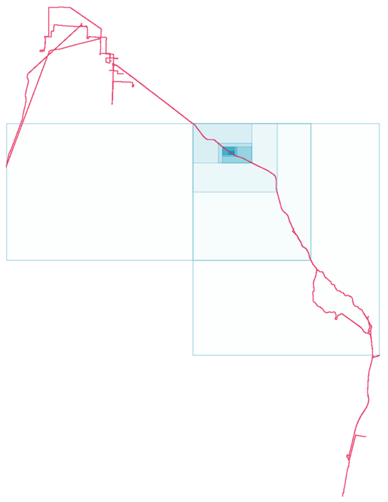
Fig. 15 A trajectory (in red) and a selection of rectangles with high packedness. Darker regions have a higher approximate packedness (colour figure online)

a random sample of the path in Fig. 12. The vertices were sampled based on their x coordinates with 50 points between them. The radii were sampled by factors of 50 based on the distance to their nearest vertical or horizontal neighbor (also known as the L_∞ norm).

The high packedness of the center part of the curve might be due to high traffic in that part of the city.

Using WSPD with separation factor $s = 0.0001$, the number of queries on this trajectory reduces to 1173 (Note that since WSPD pairs approximate all pairwise distances in a point set (see [27]), so this summarization technique works for all rectangles, not just squares).

Figure 13 shows this trajectory, along with the set of rectangles chosen based on the WSPD pairs built on its vertices.

Figure 14 shows the curve lengths inside the candidate rectangles.

Figure 15 shows the rectangle with the highest approximate packedness.

Both codes take less than 2 minutes, use about a thousand queries, and give roughly the same values for the packedness: The WSPD sparsification reports 751.75 and the random sampling reports 2109.4, which is much less than the factor $2 + 8/s = 8002$.

The second trajectory that we use in this experiment is associated with bird 104 of GPS Data of Seabirds [10] dataset, which is the trajectory with the maximum number of data points in that dataset. We use longitude and latitude of the birds location. The number of vertices of this trajectory is 6048. Using WSPD summarization, the number of queries

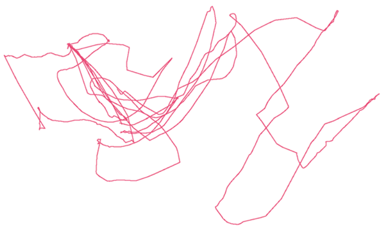

Fig. 16 Track 104 of GPS Data of Seabirds

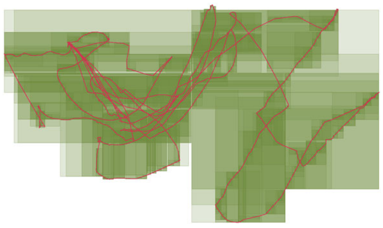

Fig. 17 Track 104 of GPS Data of Seabirds and the rectangular queries computed using WSPD

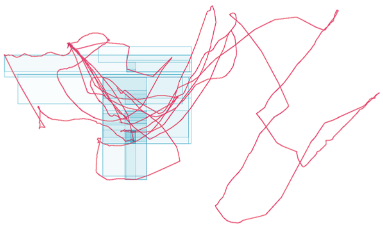

Fig. 18 Track 104 of GPS Data of Seabirds and rectangles with high packedness

reduces to 9117, and the number of queries with high packedness is 398.

Figure 16 shows the input track.

Figure 17 shows the WSPD candidate rectangles for computing packed regions.

Figure 18 represents the selected rectangles. The values of the parameters were set the same as the previous track. Our algorithm estimates 7230.21 for the packedness of this curve.

So, the sparsification method based on WSPD efficiently determines regions with high packedness.

6.3 Implementation details and improvements

A 2D segment tree for points can be used to implement the segment tree for segments and rectangles: simply query both endpoints of the segment, and break the segment on the boundaries of the node containing it. This is done at both preprocessing and query steps.

Our segment tree for segments also supports similar queries, such as weighted length query, and it works for dynamic weights as well: subtract the previous weight and add the new weight.

If the endpoints are chosen from a $\tau \times \tau$ grid, by building the segment tree on the grid, which takes $O(\tau^2 \log \tau)$ time, it is possible to have insertions, deletions, and modifications. Both length queries and modifications take $O(\log \tau)$ time, and traverse the tree from leaf to root on each of their endpoints.

If you are interested in the packedness of the shape of the curve in large scale, first simplify the curve, then compute the packedness. Otherwise, the packedness mostly shows the regions with high traffic. The set of possible radii is the one described in HAQ, i.e., the same set of query sizes used when computing the packedness.

To get a better approximation factor on big data, use parallel or MapReduce algorithms for WSPD. The simplest method is the recursive construction based on a quadtree, where the subproblems are pairs of nodes of the tree. By building and storing these pairs after the first round, you can recurse on their children until the WSPD condition is satisfied or the space limit is reached, where you store the current pairs for the next round.

Sampling works well on curves where the packedness is high at least for a dense or long part of the curve, for sampling the vertices or fixed intervals of the curve length.

When normalizing the data, be careful to scale the coordinates by the same factor, or scale the query shape to match it. For example if you are using TikZ package to draw the diagrams.

7 Conclusions and open problems

Detecting mass gatherings based on mobile GPS data, analyzing the movement of a robot to detect unwanted circular sequences of movements, tracing genetic mutations in animals based on their migration paths, and estimating the traffic congestion of roads based on GPS trajectories from vehicles are examples where computing the traveled distance in an area can give valuable insights about the data.

We described the problem of computing the packedness of a curve as a batch of length queries. We then reduced the number of queries by grouping the spatially similar queries of roughly the same size together. For other problems, domain-specific pruning rules and grouping correlated queries can be used to sparsify queries. On massive data sets, a sparsified batch of queries can be parallelized by sending the sparsified queries to all machines. Even an inherently sequential algorithm such as line-sweeping can be parallelized using this method, since it has spatial dependency defined by the sweep line.

Our hierarchical aggregated queries (HAQ) data-structure is a proof of concept for the existence of polynomial time algorithms for a batch of length queries. Generalizing our method to finding non-intersecting top-k queries, computing packedness assuming some vertices of the curve can be outliers and therefore ignored, and designing a simplification algorithm whose simplification error for each point of the curve is a function of the packedness of that point are some immediate future works.

Defining personalized movement profiles based on packedness instead of just using thresholds on the speed for defining movement patterns such as walking, jogging, etc. for humans, incorporates more detailed information, including the effects of weather conditions and day of the year on traffic conditions in different areas. To build such profiles, the packedness of different parts of a GPS trajectory can be used, assuming the sampling frequency is fixed. Packedness and features extracted from it, can be stored in a non-spatial database, which makes them easier to integrate with such systems, compared to communicating queries and their solutions at query time. The extract-transform-load (ETL) process required for computing packedness in the streaming setting requires three passes over the data. In the first pass, the sparsified or sampled queries are computed. In the second pass, the values of these queries are computed. Finally in the third pass, the values of sparsified queries are used to solve the actual batch of queries. Reducing the number of sparsified queries, and therefore the amount of memory used by the algorithm, from near-linear to sublinear makes the algorithm more efficient and scalable in the streaming setting.

Author Contributions Sepideh Aghamolaei, Vahideh Keikha, Mohammad Ghodsi, and Ali Mohades contributed to writing, review, and editing. Sepideh Aghamolaei contributed to implementation. Mohammad Ghodsi and Ali Mohades contributed to supervision.

Funding V. Keikha was Supported by the Czech Science Foundation, Grant No. GJ19-06792Y, and with institutional support RVO:67985807.

Availability of data and material Publicly available.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Code availability Publicly available at https://github.com/saghamolaei/Packedness.

References

1. GPS Trajectories. UCI Machine Learning Repository (2016)
2. Afshar, R., Goodrich, M.T., Matias, P., Osegueda, M.C.: Reconstructing biological and digital phylogenetic trees in parallel. In: 28th Annual European Symposium on Algorithms (ESA 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
3. Agarwal, P.K.: Partitioning arrangements of lines ii: applications. Discrete Comput. Geom. **5**(6), 533–573 (1990)
4. Agarwal, P.K., Har-Peled, S., Varadarajan, K.R.: Approximating extent measures of points. J. ACM (JACM) **51**(4), 606–635 (2004)
5. Agarwal, P.K., Katz, M.J., Sharir, M.: Computing depth orders for fat objects and related problems. Comput. Geom. **5**(4), 187–206 (1995)
6. Aghamolaei, S., Baharifard, F., Ghodsi, M.: Geometric spanners in the MapReduce model. In International Computing and Combinatorics Conference, pp. 675–687. Springer, Berlin (2018)
7. Aghamolaei, S., Keikha, V., Ghodsi, M., Mohades, A.: Windowing queries using Minkowski sum and their extension to MapReduce. J. Supercomput. (2020)
8. Beame, P., Koutris, P., Suciu, D.: Communication steps for parallel query processing. J. ACM (JACM) **64**(6), 1–58 (2017)
9. Bringmann, K., Künnemann, M.: Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. In International Symposium on Algorithms and Computation, pp. 517–528. Springer, Berlin (2015)
10. Browning, E., Bolton, M., Owen, E., Shoji, A., Guilford, T., Freeman, R.: Predicting animal behaviour using deep learning: Gps data alone accurately predict diving in seabirds. Methods Ecol. Evol. (2017). https://doi.org/10.5061/dryad.t7ck5
11. Callahan, P.B.: Dealing with higher dimensions: the well-separated pair decomposition and its applications. Ph.D. thesis, Johns Hopkins University (1995)
12. Chen, D., Driemel, A., Guibas, L.J., Nguyen, A., Wenk, C.: Approximate map matching with respect to the Fréchet distance. In 2011 Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 75–83. SIAM (2011)
13. Cruz, M.O., Macedo, H., Guimaraes, A.: Grouping similar trajectories for carpooling purposes. In 2015 Brazilian Conference on Intelligent Systems (BRACIS), pp. 234–239. IEEE (2015)
14. De Berg, M., Van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational geometry. In Computational Geometry, pp. 1–17. Springer, Berlin (1997)
15. Driemel, A., Har-Peled, S.: Jaywalking your dog: computing the Fréchet distance with shortcuts. SIAM J. Comput. **42**(5), 1830–1866 (2013)
16. Driemel, A., Har-Peled, S., Wenk, C.: Approximating the Fréchet distance for realistic curves in near linear time. Discrete Comput. Geom. **48**(1), 94–127 (2012)
17. Driemel, A., Krivošija, A.: Probabilistic embeddings of the Fréchet distance. In International Workshop on Approximation and Online Algorithms, pp. 218–237. Springer, Berlin (2018)
18. Gerbessiotis, A.V.: An architecture independent study of parallel segment trees. J. Discrete Algorithms **4**(1), 1–24 (2006)
19. Goodrich, M.T., Sitchinava, N., Zhang, Q.: Sorting, searching, and simulation in the MapReduce framework. In International Symposium on Algorithms and Computation, pp. 374–383. Springer, Berlin (2011)
20. Goodrich, M.T., Sitchinava, N., Zhang, Q., IT-Parken, A.: Sorting, searching, and simulation in the MapReduce framework. arXiv preprint arXiv:1101.1902 (2011)
21. Gudmundsson, J., van Kreveld, M., Staals, F.: Algorithms for hotspot computation on trajectory data. In Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 134–143 (2013)
22. Gudmundsson, J., Sha, Y., Wong, S.: Approximating the packedness of polygonal curves. In Cao, Y., Cheng, S.W., Li, M. (eds.) 31st International Symposium on Algorithms and Computation (ISAAC 2020), *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 181, pp. 9:1–9:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020). 10.4230/LIPIcs.ISAAC.2020.9. https://drops.dagstuhl.de/opus/volltexte/2020/13353

23. Gudmundsson, J., Smid, M.: Fast algorithms for approximate Fréchet matching queries in geometric trees. Comput. Geom. **48**(6), 479–494 (2015)
24. Har-Peled, S., Raichel, B.: The Fréchet distance revisited and extended. ACM Trans. Algorithms (TALG) **10**(1), 1–22 (2014)
25. Har-Peled, S., Zhou, T.: How packed is it, really? CoRR (2021). arXiv:2105.10776
26. Leighton, F.T.: Introduction to parallel algorithms and architectures: Arrays · trees · hypercubes. Elsevier, Amsterdam (2014)
27. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, Cambridge (2007)
28. Parhami, B.: Introduction to Parallel Processing: Algorithms and Architectures. Springer Science and Business Media, Berlin (2006)
29. Williams, V.V., Williams, R.R.: Subcubic equivalences between path, matrix, and triangle problems. J. ACM (JACM) **65**(5), 1–38 (2018)
30. Zheng, Y., Li, Q., Chen, Y., Xie, X., Ma, W.Y.: Understanding mobility based on gps data. In Proceedings of the 10th International Conference on Ubiquitous Computing, pp. 312–321 (2008)
31. Zheng, Y., Xie, X., Ma, W.Y., et al.: Geolife: a collaborative social networking service among user, location and trajectory. IEEE Data Eng. Bull. **33**(2), 32–39 (2010)
32. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from gps trajectories. In Proceedings of the 18th International Conference on World Wide Web, pp. 791–800 (2009)