

How Packed Is It, Really?

Sariel Har-Peled* Timothy Zhou†

October 11, 2026

Abstract

The congestion of a curve is a measure of how much it zigzags around locally. More precisely, a curve π is c -packed if the length of the curve lying inside any ball is at most c times the radius of the ball, and its congestion is the minimum c for which π is c -packed. This paper presents a randomized 42-approximation algorithm for computing the congestion of a curve (or any set of segments in the plane). It runs in $O(n \log^3 n)$ time and succeeds with high probability. Although the approximation factor is large, the running time improves over the previous fastest constant approximation algorithm of Gudmundsson *et al.* [GSW20], which took $\tilde{O}(n^{4/3})$ time.

Using OpenAI’s ChatGPT Astra, with Anthropic’s Fable assisting with writing and verification, we refine this algorithm to compute a $(1 + \varepsilon)$ -approximation to the congestion (defined with respect to axis-parallel squares) of n segments in the plane in $O_\varepsilon(n \log^6 n)$ time, with high probability, for any fixed $\varepsilon > 0$.

1. Introduction

In 2010, Driemel *et al.* [DHW12] provided a measurement of how “realistic” a curve is (there are several alternative definitions – see [DHW12] and

*Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; sariel@illinois.edu; <http://sarielhp.org/>. Work on this paper was partially supported by a NSF AF award CCF-1907400.

†Department of Computer Science; University of Illinois; 201 N. Goodwin Avenue; Urbana, IL, 61801, USA; tzhou28@illinois.edu.

references therein for details). Formally, a curve π is *c-packed* if the total length of π inside any ball is bounded by c times the radius of the ball. The minimum c for which the curve is c -packed is the *congestion* of the curve. Intuitively, if a curve has high congestion, then it zigzags back and forth around some locality. We expect that many real-world curves do not behave so pathologically – instead, we expect them to exhibit low congestion. For examples of real world curves, see <https://frechet.xyz>. Naturally, if the curve is a tracking of an entity over long enough time, the congestion might be high (for example, a soccer player’s movement during a whole game, an airplane’s path over a month, etc.).

Curves with low congestion lend themselves to efficient algorithms. Notably, they can be efficiently approximated by simpler curves which nearly preserve Fréchet distances, and therefore the Fréchet distances between them can be approximated in near-linear time [DHW12]. In general, if the congestion is $\Omega(n)$, then computing the Fréchet distance is more difficult. Indeed, assuming the Strong Exponential Time Hypothesis (SETH), even approximating Fréchet distance within a constant factor requires quadratic time, as shown by Bringmann [Bri14]. The decision version of the problem is also conjectured to be 3SUM-hard, see Alt [Alt09]. Note that proving a direct connection between 3SUM-hardness and SETH is still an open problem, see Vassilevska Williams [Wil15].

We would like to verify that a given curve is indeed c -packed for a low value of c . Some algorithms for c -packed curves do not require knowing the value of the congestion – rather, their analyses show that if the curve is c -packed for some small c , then the algorithms run in near-linear time. However, verifying that curves are c -packed would increase our confidence that these algorithms are generally applicable. This leads to the question of how quickly one can estimate or compute the congestion. In this paper, we present a constant-factor approximation algorithm for the congestion that runs in near-linear time.

Disks and squares are all the same. As far as the congestion is concerned, whether we use disks/balls or squares/cubes in the definition is the same up to a constant factor (i.e., $\sqrt{2}$ in the plane). Thus, in the following, we usually work with squares, since it is easier.

Previous work. As mentioned earlier, the concept of c -packedness was introduced in Driemel *et al.* [DHW12]. Computing the congestion exactly runs into the issue of minimizing sums of square roots. As such, an exact algorithm in the standard RAM model is unlikely. The work of Vigneron [Vig14] provided a $(1 + \varepsilon)$ -approximation algorithm which runs in $O((n/\varepsilon)^{d+2} \log^{d+2}(n/\varepsilon))$ time. Gudmundsson *et al.* [GvKS13] offered a cubic-time algorithm for this problem. Aghamolaei *et al.* [AKGM20] gave a $(2 + \varepsilon)$ -approximation for the problem of approximating the congestion of planar curves, but unfortunately their running time seems to be at least quadratic.

Some of the previous work of Gudmundsson *et al.* [GvKS13] was also concerned with computing hotspots, which are small regions in the plane where many segments pass through. Computing regions of high congestion naturally leads to finding hotspots. However, one usually fixes the resolution of the desired hotspots (i.e. the radius of the balls being intersected) before searching for them.

More recently, Gudmundsson *et al.* [GSW20] gave a $(6 + \varepsilon)$ -approximation to the congestion (i.e., packedness) of a polygonal curve in $\tilde{O}(n^{4/3}/\varepsilon^4)$ time. As part of their algorithm, they showed that one can quickly find a set of $O(n)$ squares whose congestions yield a constant approximation to the congestion of the curve. They also noted that the problem of finding the congestion of these squares with respect to the segments seems quite similar to the Hopcroft problem, as discussed below.

The Hopcroft problem. Given a set P of n points in the plane, and a set L of n lines in the plane, the question is whether there is a point of P that is incident to one of the lines of L . There is an $\Omega(n^{4/3})$ lower bound for Hopcroft’s problem due to Erickson [Eri95]. As usual, this lower bound holds only in a restricted model of computation (algebraic decision tree model), but it is believed to hold in broader models of computation. The belief that this lower bound is correct (up to maybe some polylog noise) in any reasonable model of computation is quite important as it provides matching lower bound to the main results known in range searching.

In our language, the Hopcroft problem can be stated as having a set of n segments, and a set of n (disjoint) squares, and asking for the maximum congestion of the squares in relation to the segments.

The challenge. If we replace a point by a short segment, then deciding the packedness for sufficiently small squares centered at these points is equivalent to solving the Hopcroft problem. Naturally, this reduction of hardness only works if the approximation is quite small (say, below factor 2). For this reason, Gudmundsson *et al.* [GSW20] deemed it unlikely that their approach, “or a similar approach, can lead to a considerably faster algorithm” for computing congestion. Although this lower bound holds only if the approximation factor is small, it is interesting to ask whether one can break this “lower bound” by settling for a worse approximation.

Our result. Despite this difficulty, we present a randomized $O(n \log^3 n)$ time algorithm that provides a constant approximation to the congestion of a set of segments in $\mathbb{R}^2$. The algorithm bypasses the barrier presented by Hopcroft’s problem by observing that, when computing congestion, the generated instances for the Hopcroft problem have high congestion. In such scenarios, we do not need to compute the congestion of $\Omega(n)$ disjoint squares exactly (as required by the Hopcroft problem).

In [Section 6](#), we show that the approximation factor can be improved to $1 + \varepsilon$, for any fixed $\varepsilon > 0$, while keeping the running time near-linear. Specifically, we get a $(1 + \varepsilon)$ -approximation to the congestion of n segments in the plane, with respect to axis-parallel squares, in $O_\varepsilon(n \log^6 n)$ time, with high probability, see [Theorem 6.7](#). The main new ingredients are a sampling lemma for lengths rather than counts, and the exact computation of the congestion of each (small) list of long segments over all squares, not only over its own quadtree cell.

Sketch of algorithm. The *congestion* of a square with respect to a curve π is the total length of π inside it, divided by its radius. Following Gudmundsson *et al.* [GSW20], we reduce the problem of approximating the congestion of a curve to that of computing the congestion of $O(n)$ squares. Then we build a “few” quadtrees whose cells approximate these squares, so that it suffices to compute the congestion of the quadtrees.

We can compute the congestion of a quadtree cell by finding all the segments of the curve which intersect it, then explicitly computing the total length of the intersections. However, naively searching for all the segments intersecting each of the squares takes $O(n^2)$ time. To speed things up, we store each segment at some quadtree cell with comparable length. For a given

cell, its short segments are those stored at descendants in the quadtree, and its long segments are those stored at ancestors. To find the congestion of a cell, we compute the sum of its long and short congestions – the congestions with respect to all of its long and short segments, respectively. We can quickly compute the short congestion of a cell by summing the lengths of short segments in the quadtree bottom-up.

Exactly computing the long congestion is somewhat tricky. Fortunately, approximating the long congestion only requires counting the maximum number of long segments intersecting any cell. If every cell intersects only a few long segments, then we can quickly enumerate all the intersections by searching the quadtree top-down. If there is a cell intersecting many long segments, then the algorithm performs an exponential search to guess how many intersections it has. The algorithm quickly verifies its guess by taking a random sample of input segments, enumerating its intersections with each quadtree cell, and using the counts to estimate the maximum number of intersections.

Highlight. This work “bends” what was previously believed to be a lower bound on the running time for approximating congestion (i.e., $\Omega(n^{4/3})$). While most of the tools we use are standard, the way we combine them is non-trivial and offers new insights into the problem.

Our first algorithm loses constant factors in several places, and its approximation factor is quite bad compared to previous work (i.e., 42 vs. 6). However, it runs in near-linear time, which is significantly faster. We find this result surprising, as it bypasses the barrier formed by the Hopcroft problem mentioned above. The refined algorithm of [Section 6](#) then brings the approximation factor down to $1 + \varepsilon$, at the price of a few more logarithmic factors in the running time.

AI-assisted work. The algorithm and analysis in [Section 6](#) were developed using OpenAI’s ChatGPT Astra with the High reasoning setting. Anthropic’s Fable was used to refine the writing and help verify and simplify the proofs. Beyond posing the open problem, the human contribution to this part of the paper was minimal.

Paper organization. We start in [Section 2](#) by providing some definitions and background. In [Section 3](#), we reduce the problem of computing the congestion to that of computing the congestions of a small number of quadtrees.

This in turn reduces to the problem of computing the congestion from long and short segments. The short congestion is handled in [Section 3.3](#), while the main challenge of approximating the long congestion is addressed in [Section 4](#), where we approximate the load of the long segments – i.e., the maximum number of long segments intersecting a single cell of the quadtree. In [Section 5](#), we put everything together and obtain the 42-approximation. In [Section 6](#), we refine the algorithm into a $(1 + \varepsilon)$ -approximation. We conclude in [Section 7](#) with a few remarks.

2. Preliminaries

2.1. Standard tools

Definition 2.1. For a real positive number τ , let G_τ be the *grid* partitioning the plane into axis-parallel squares of sidelength τ . Formally, this grid is defined by the mapping $G_\tau(x, y) = (\lfloor x/\tau \rfloor, \lfloor y/\tau \rfloor)$. The number τ is the *width* or *sidelength* of G_τ . For integers i, j , the (i, j) -*grid cell* is the $\tau \times \tau$ square formed by the set $G_\tau^{-1}(i, j)$.

Definition 2.2. A square is a *canonical square* if it is contained inside the unit square, it is a cell in a grid G_w , and w is a power of two. That is, the square corresponds to a node in the infinite quadtree defined over $[0, 1)^2$. The grid generating a canonical square is a *canonical grid*.

2.2. Congestion

Definition 2.3. Let $\square = \square(p, r)$ denote the axis-parallel square in $\mathbb{R}^2$ centered at a point $p \in \mathbb{R}^2$ with *sidelength* $2r$. The square $\square$ can be interpreted as a ball in the L_∞ norm, and as such, its *radius* is r .

Definition 2.4. For a segment s , let $\|s\|$ denote the length of s . Similarly, for a set of segments $\mathcal{S}$, let $\|\mathcal{S}\| = \sum_{s \in \mathcal{S}} \|s\|$ denote the total length of segments in $\mathcal{S}$.

For a square $\square = \square(p, r)$, the *conflict list* of $\square$ (for a set $\mathcal{S}$ of segments) is the set of segments intersecting $\square$, that is

$$L(\square) = \{s \in \mathcal{S} \mid s \cap \square \neq \emptyset\}.$$

Let

$$\mathcal{S} \sqcap \square = \{s \cap \square \mid s \in L(\square)\}$$

be the clipping of the segments of $L(\square)$ to $\square$. The *congestion* of the square $\square$, with respect to $\mathcal{S}$, is

$$c(\square) = c_{\mathcal{S}}(\square) = \|\mathcal{S} \sqcap \square\| / r.$$

The *congestion* of the set of segments $\mathcal{S}$ is $c(\mathcal{S}) = \max_{p,r} c_{\mathcal{S}}(\square(p,r))$. Given a set of squares Ξ , its *congestion* is $c_{\mathcal{S}}(\Xi) = \max_{\square \in \Xi} c(\square)$.

Definition 2.5. For a constant $c > 0$, a set $\mathcal{S}$ of segments in $\mathbb{R}^2$ is *c-packed* if, for any point $p \in \mathbb{R}^2$ and any value $r > 0$, the total length of the segments of $\mathcal{S}$ inside a square $\square = \square(p,r)$ is at most cr . That is, the congestion of $\mathcal{S}$ is at most c .

Thus, the congestion of $\mathcal{S}$ is the minimum c for which $\mathcal{S}$ is c -packed. We are interested in approximating $c(\mathcal{S})$. To this end, we follow Gudmundsson *et al.* [GSW20], who reduced the problem to querying the lengths of intersections between the curve and some squares. While Gudmundsson *et al.* state their result for a curve, it holds for any set of segments.

Lemma 2.6 (Lemma 12 in Gudmundsson *et al.* [GSW20]). *Given a set $\mathcal{S}$ of n segments in the plane, and a parameter $\varepsilon \in (0, 1)$, one can compute, in $O(n \log n + n/\varepsilon^2)$ time, a set $\mathcal{G}_{\mathcal{S}}$ of $O(n/\varepsilon^2)$ axis-aligned squares, such that $c(\mathcal{S}) \geq c_{\mathcal{S}}(\mathcal{G}_{\mathcal{S}}) \geq c(\mathcal{S})/(6 + \varepsilon)$.*

Remark 2.7. For completeness, we sketch informally an alternative proof for (a weaker version of) **Lemma 2.6**. Let P be the set of endpoints of the segments of $\mathcal{S}$, and construct a (randomly translated) compressed quadtree $\mathcal{T}$ for the points of P . For every node v in $\mathcal{T}$, we add the square $\square_v$ to the set of candidate squares $\mathcal{G}_{\mathcal{S}}$, and we also add a few scaled copies, say $2\square_v$ and $4\square_v$ to $\mathcal{G}_{\mathcal{S}}$.

Let $\square$ be the square with maximum congestion $c(\mathcal{S})$. We replace $\square$ by a square $\square' \supseteq \square$ that is centered in one of the endpoints of P , with congestion at least $c(\mathcal{S})/2$ (the interesting case is when $\square$ contains no point of P , but then just enlarge it till it does).

We then continue enlarging $\square'$ till it hits another point of P . Let $\square''$ be the resulting square. Informally, the loss in congestion is a constant. Now,

the center of $\square''$ and a point on its boundary both belong to P . With constant probability $\square''$ is fully contained in a cell $\square_v$ of a node v of the quadtree (or one of its enlarged copies added explicitly to $\mathcal{G}_S$), that is only a constant factor bigger. Thus, a square in $\mathcal{G}_S$ provides the desired approximation to the congestion.

3. The algorithm: The long and short of it

3.1. Reduction to quadtrees

In what follows, let $\mathcal{G}_S$ be the set of squares computed by [Lemma 2.6](#) for $\mathcal{S}$ (the value of ε will be specified shortly). To approximate $c(\mathcal{S})$, it suffices to approximate $c_S(\mathcal{G}_S)$.

Since congestion is invariant under translation and scaling, we might as well assume that $\mathcal{S}, \mathcal{G}_S \subseteq [0, 1/8]^2$. We can randomly scale both sets by a random number $\mathbf{s}_r \in [1, 2]$, and shift both sets by a random vector $\mathbf{u}_r \in [0, 1/2]^2$. Let $\Lambda(p) = \mathbf{s}_r p + \mathbf{u}_r$ be the resulting affine mapping. We compute for each square $\square \in \Lambda(\mathcal{G}_S)$ the smallest canonical square $\text{CANON}(\square)$ that contains it. Conceptually, consider the infinite quadtree, and let v be the lowest node in the quadtree whose cell contains $\square$. The square $\square_v$ is $\text{CANON}(\square)$. Algorithmically, $\text{CANON}(\square)$ can be computed in $O(1)$ time using the floor function and some basic bit operations, see [\[Har11\]](#). Next, we build a quadtree that has all these marked canonical squares as nodes. This can be done in $O(n \log n)$ time [\[Har11\]](#). The idea is to repeat this process a sufficient number of times, such that in one of the generated quadtrees, $\text{CANON}(\square)$ and $\square$ are almost identical, and this holds for all the squares of interest; see [Figure 3.1](#).

Lemma 3.1. *For a square $\widehat{\square} \in \mathcal{G}_S$, and a parameter $\varepsilon \in (0, 1/2)$, consider its randomly scaled and shifted copy $\square = \mathbf{s}_r \widehat{\square} + \mathbf{u}_r$, and its canonized version $\square' = \text{CANON}(\square)$.*

The probability that $r = \mathbf{r}(\square) \leq \mathbf{r}(\square') \leq (1 + \varepsilon)r$ is $\geq (\varepsilon/8)^3$.

Proof: By construction $\square \subseteq \square'$. Let $\ell = \mathbf{r}(\widehat{\square})$, and let $L = 2^{\lceil \log_2 \ell \rceil}$ be the rounding up of ℓ to its closest power of 2, so that $\ell \leq L < 2\ell$. Since $\widehat{\square} \subseteq [0, 1/8]^2$, we have $\ell \leq 1/16$ and $L \leq 1/16$.

The radius $r = \mathbf{r}(\square) = \mathbf{s}_r \ell$ of the scaled square is uniformly distributed in $[\ell, 2\ell]$. Let $J = [(1 - \varepsilon/4)L, (1 - \varepsilon/8)L]$ and $2J = [(1 - \varepsilon/4)2L, (1 - \varepsilon/8)2L]$. For an interval J' , let $\|J'\|$ denote its length. We have $\|J\| = (\varepsilon/8)L \geq (\varepsilon/8)\ell$.

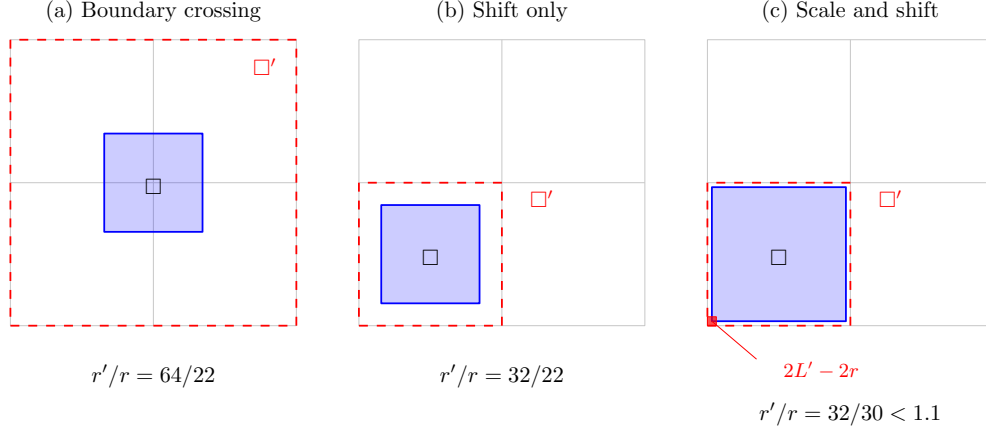


Figure 3.1: Shifting and scaling for tight containment. The shaded red patch in (c) gives admissible lower-left corners within one grid cell.

The two intervals are disjoint, as $(1 - \varepsilon/4)2L > L$, and $J \subseteq [\ell/2, 2\ell]$, as $(1 - \varepsilon/4)L \geq \ell/2$ and $L < 2\ell$. We consider the random scaling $\mathbf{s}_r$ to be *good* if $r \in J \cup 2J$. Since $2J \cap [\ell, 2\ell] = 2(J \cap [\ell/2, \ell])$, we have

$$\|J \cap [\ell, 2\ell]\| + \|2J \cap [\ell, 2\ell]\| = \|J \cap [\ell, 2\ell]\| + 2\|J \cap [\ell/2, \ell]\| \geq \|J \cap [\ell/2, 2\ell]\| = \|J\| \geq (\varepsilon/8)\ell.$$

Thus, the random scaling is good with probability $\geq \varepsilon/8$, and assume this happened. Let $L' = L$ if $r \in J$, and $L' = 2L$ if $r \in 2J$. In either case L' is a power of two, $L' \leq 1/8$, and $2r \in [(1 - \varepsilon/4)2L', (1 - \varepsilon/8)2L']$.

Consider the canonical grid $\mathbf{G}_{2L'}$. The set of all good translations

$$U = \left\{ (x, y) \in \mathbb{R}^2 \mid (x, y) + \mathbf{s}_r \widehat{\square} \text{ contained in a cell of } \mathbf{G}_{2L'} \right\}$$

is a grid like set with sidelength $2L'$, where each grid point is replaced by a square of sidelength $2L' - 2r \geq (\varepsilon/8)2L'$. As $2L' \leq 1/4$ is a power of two, the square $[0, 1/2]^2$ is tiled by cells of $\mathbf{G}_{2L'}$, and the probability that the random shifting $\mathbf{u}_r$ falls in U is at least $(\varepsilon/8)^2$.

If both things happen, then $\square$ is contained in a cell of $\mathbf{G}_{2L'}$. This cell is a canonical square, as $\square \subseteq [0, 3/4]^2$ and $2L' \leq 1/4$, so the cell lies in $[0, 1)^2$. Thus $r \leq r(\square') \leq L' \leq (1 + \varepsilon)(1 - \varepsilon/4)L' \leq (1 + \varepsilon)r$. Namely, $\square \subseteq \square'$, and $r = r(\square) \leq r(\square') \leq (1 + \varepsilon)r$, as desired. $\blacksquare$

Definition 3.2. For a compressed quadtree $\mathcal{T}$, let $\text{cells}(\mathcal{T})$ be the set of all squares formed by nodes of $\mathcal{T}$. The **congestion** of a quadtree $\mathcal{T}$ for a set of segments $\mathcal{S}$ is the quantity $c_{\mathcal{S}}(\mathcal{T}) = c_{\mathcal{S}}(\text{cells}(\mathcal{T}))$.

Lemma 3.3. *Given a set $\mathcal{S}$ of n segments, one can compute, in $O(n \log^2 n)$ time, $m = O(\log n)$ (shifted) compressed quadtrees, $\mathcal{T}_1, \dots, \mathcal{T}_m$, such that $\frac{1}{7}c(\mathcal{S}) \leq \max_i c_{\mathcal{S}}(\mathcal{T}_i) \leq c(\mathcal{S})$. This holds with high probability.*

Proof: Let $\varepsilon \in (0, 1)$ be a sufficiently small constant to be specified shortly. Compute the set $\mathcal{G}_{\mathcal{S}}$ of squares specified by **Lemma 2.6**. Next, compute $m = O(\varepsilon^{-3} \log n)$ randomly shifted copies $\mathcal{G}_{\mathcal{S}}^1, \dots, \mathcal{G}_{\mathcal{S}}^m$ of $\mathcal{G}_{\mathcal{S}}$, as described above. For each one of them we compute a compressed quadtree. Each such compressed quadtree can be interpreted as a shifted compressed quadtree over the original set of squares $\mathcal{G}_{\mathcal{S}}$. Let $\mathcal{T}_1, \dots, \mathcal{T}_m$ be these quadtrees.

Fix a square $\square \in \mathcal{G}_{\mathcal{S}}$. By **Lemma 3.1**, with probability $\varsigma \geq (\varepsilon/8)^3$, there is a node u , and thus its cell $\square'$, such that $\square \subseteq \square'$ and $r(\square') \leq (1 + \varepsilon)r(\square)$. Thus, we have $c_{\mathcal{S}}(\square') \geq c_{\mathcal{S}}(\square)/(1 + \varepsilon)$.

In particular, the probability that this “good” containment does not happen for $\square$, in all m quadtrees, is $(1 - \varsigma)^m \leq 1/n^{O(1)}$. We conclude that, with high probability, for all squares $\square \in \mathcal{G}_{\mathcal{S}}$, there is at least one node/square, in the computed quadtrees, that tightly contains $\square$.

This readily implies that

$$\max_i c_{\mathcal{S}}(\mathcal{T}_i) \geq \frac{1}{1 + \varepsilon} \max_{\square \in \mathcal{G}_{\mathcal{S}}} c_{\mathcal{S}}(\square) \geq \frac{1}{(1 + \varepsilon)(6 + \varepsilon)} c(\mathcal{S}) \geq \frac{1}{7} c(\mathcal{S}),$$

for $\varepsilon = 1/10$. ■

Remark 3.4. Throughout the algorithm, we use compressed quadtrees, rather than regular quadtrees. The use of compressed quadtrees is necessary to get an efficient runtime, but it does not affect the description of our algorithm significantly. (Intuitively, the compressed quadtree compresses paths and not cells, so one can easily ensure that all cells of interest appear in the compressed quadtree as regular cells). As such, from this point on, we use quadtree as a shorthand for a compressed quadtree, and we ignore the minor low-level technical details that arise because of the compression.

Specifically, the only issue in the algorithm where this would make a difference is in traversing down the tree and assuming that the child is exactly half the size of its parent. This can easily be fixed by working directly with the node’s own record of its dimensions.

3.2. First steps towards approximating the congestion of a quadtree

Given a set $\mathcal{S}$ of n segments and a quadtree $\mathcal{T}$ of size $O(n)$, we wish to compute the congestion of $\mathcal{T}$. To simplify the exposition, we assume that any shift needed is applied to the input segments, and the quadtree is thus the standard one built over $[0, 1]^2$.

3.2.1. A naïve exact algorithm for the congestion of a quadtree

At each node $v \in \mathcal{T}$, the algorithm stores a *conflict list* $L(v)$ – a list of all the segments of $\mathcal{S}$ intersecting the cell $\square_v$ associated with v . It begins by storing $\mathcal{S}$ at the root of $\mathcal{T}$, then recursively traverses down the tree. At each parent node u , the algorithm sends the list $L(u)$ to all its children. Each child v finds all the segments which intersect its cell $\square_v$ and adds them to its own list. At the end of this process, the algorithm has computed the segments intersecting each quadtree node, and it can use them to compute the congestion. Overall, this algorithm takes $O(n^2)$ time.

3.2.2. The long and the short of it

To speed up the naïve algorithm, we implement several strategies. The first is to register the segments directly in the cells at a suitable resolution of the quadtree.

Definition 3.5 (long/short threshold). Let $\alpha > 0$ be a fixed integer constant, to be specified shortly¹. The parameter α is the *transition* constant. A segment s is *α -long* (resp., *α -short*) for a square $\square = \square(p, r)$ if it intersects the interior of $\square$ and $\|s\| \geq \alpha r$ (resp., $\|s\| < \alpha r$). For a square $\square$ with radius r , its set of α -long (resp., α -short) segments is denoted by $\mathcal{S}_{\geq \alpha}(\square)$ (resp., $\mathcal{S}_{< \alpha}(\square)$).

Lemma 3.6. *For each segment $s \in \mathcal{S}$, let $G(s, \alpha)$ be the set of interior-disjoint canonical squares of maximal size for which s is α -long. There are at most $O(1 + \alpha)$ such squares, and they can be computed in $O(1 + \alpha)$ time.*

¹Spoiler alert! The butler did it, $\alpha = 20$, and the hero dies in the end. The hero did try to expose α , but it was too late for them.

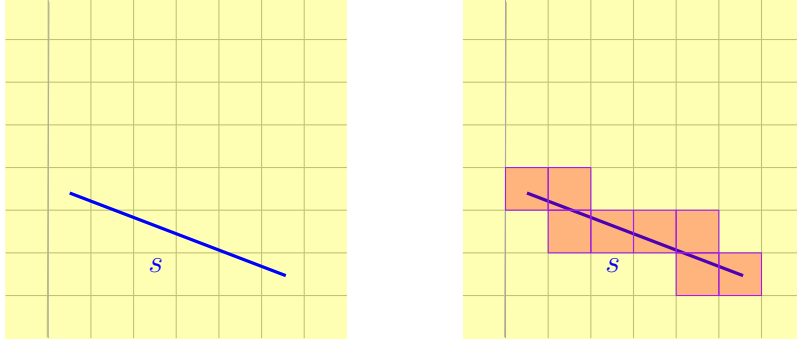


Figure 3.2: Computing a maximal set of α -long segments for a segment s (see [Lemma 3.6](#)). In this case, the segment has length 5, and each grid cell is of radius 0.5, so the segment is 10-long for all the cells of the grid it intersects.

Proof: Let $i = \lfloor \log_2(2\|s\|/\alpha) \rfloor$, and $\tau = 2^i$. Then

$$\frac{\alpha\tau}{2} = \alpha 2^{\lfloor \log_2(2\|s\|/\alpha) \rfloor - 1} \leq \|s\| < \alpha 2^{\lfloor \log_2(2\|s\|/\alpha) \rfloor} = \alpha\tau.$$

A cell of $\mathbf{G}_\tau$ has radius $\tau/2$, and a canonical square of sidelength $\geq 2\tau$ has radius $\geq \tau$. Thus, s is α -long for every cell of $\mathbf{G}_\tau$ whose interior it intersects, but it is not α -long for any larger canonical square. Since every canonical square of sidelength $\leq \tau$ is contained in a cell of $\mathbf{G}_\tau$, the set $\mathbf{G}(s, \alpha)$ is exactly the set of cells of $\mathbf{G}_\tau$ whose interior s intersects. As $\|s\| < \alpha\tau$, the segment s crosses at most α horizontal lines and at most α vertical lines of $\mathbf{G}_\tau$. This implies that s can intersect at most $2\alpha + 1$ grid cells of $\mathbf{G}_\tau$. Computing the grid cells that intersect s is a classical problem in graphics (i.e., [line drawing problem](#)) which can be solved in $O(1 + \alpha)$ time; see [Figure 3.2](#). ■

3.2.3. Registering the segments

The refined quadtree $\mathcal{T}^+$. Given the set $\mathcal{S}$ of n segments and the above quadtree $\mathcal{T}$, the algorithm first computes the set of canonical squares $\Xi = \text{cells}(\mathcal{T}) \cup \bigcup_{s \in \mathcal{S}} \mathbf{G}(s, \alpha)$; see [Lemma 3.6](#). The algorithm then computes the (compressed) quadtree $\mathcal{T}^+$ for Ξ . This can be done in $O(n \log n)$ time [[Har11](#)], as $|\mathcal{S}| = n$, $|\text{cells}(\mathcal{T})| = O(n)$, and $|\Xi| = O((1 + \alpha)n) = O(n)$, as α is a constant. Each square of Ξ is now present as a cell of a node of the computed quadtree.

The algorithm now stores every segment $s \in \mathcal{S}$ in the cells of $\mathbf{G}(s, \alpha)$. Each cell $\square \in \mathbf{G}(s, \alpha)$ corresponds to a node v in the quadtree, and the

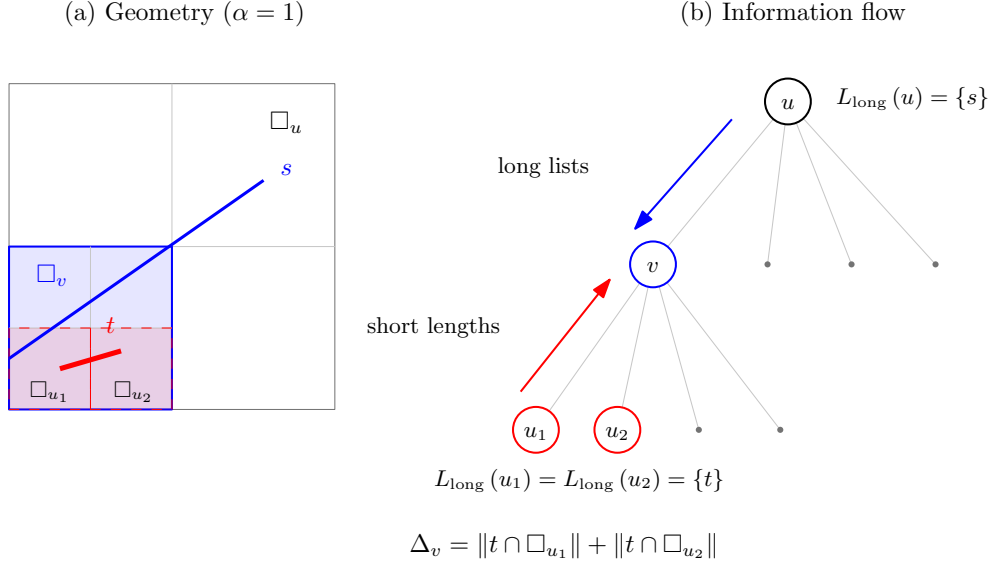


Figure 3.3: Long lists propagate downward; short clipped lengths accumulate upward. The two registered pieces of t contribute once each.

algorithm stores s in $L_{\text{long}}(v)$. This takes $O(n \log n)$ time; see [Har11]. Thus, for every quadtree node v , the algorithm computes a list $L_{\text{long}}(v)$ of segments registered there. Segments registered in this list are long for the cell $\square_v$ but short for cells in higher levels of the quadtree.

A segment s is α -short for the cell of every proper ancestor of a node of $\mathcal{G}(s, \alpha)$. Indeed, such a cell has radius at least the sidelength τ of the cells of $\mathcal{G}(s, \alpha)$, while $\|s\| < \alpha\tau$, see the proof of Lemma 3.6. As such, the lists $L_{\text{long}}(\cdot)$ also determine the short segments of every node, see Definition 3.7, and no separate registration of short segments is needed. The lists $L_{\text{long}}(\cdot)$ have total length $O(n)$.

A segment is registered only once on any path in the quadtree, as the cells of $\mathcal{G}(s, \alpha)$ are interior disjoint; see Figure 3.3.

Definition 3.7. For a node $v \in \mathcal{T}^+$, let $\text{anc}(v)$ (resp., $\text{desc}(v)$) be the list of ancestors (resp., descendants) of v in the tree $\mathcal{T}^+$. By convention, $v \in \text{anc}(v)$ and $v \in \text{desc}(v)$, that is, a node is its own ancestor and descendant. Consider a node $v \in \mathcal{T}^+$, and let $\square_v$ be its associated square. We have that

$$\mathcal{S}_{\geq \alpha}(\square_v) = \bigcup_{u \in \text{anc}(v)} (L_{\text{long}}(u) \cap \square_v) \quad \text{and} \quad \mathcal{S}_{< \alpha}(\square_v) = \bigcup_{u \in \text{desc}(v) \setminus \{v\}} L_{\text{long}}(u).$$

Definition 3.8. The α -long congestion of $\square$ is $c_{\geq\alpha}(\square) = \|\mathcal{S}_{\geq\alpha}(\square) \cap \square\| / r$. Similarly the α -short congestion of $\square$ is $c_{<\alpha}(\square) = \|\mathcal{S}_{<\alpha}(\square) \cap \square\| / r$. Given a quadtree $\mathcal{T}$, its α -long congestion and α -short congestion are

$$c_{\geq\alpha} = c_{\geq\alpha}(\mathcal{T}) = \max_{\square \in \text{cells}(\mathcal{T})} c_{\geq\alpha}(\square) \quad \text{and} \quad c_{<\alpha} = c_{<\alpha}(\mathcal{T}) = \max_{\square \in \text{cells}(\mathcal{T})} c_{<\alpha}(\square).$$

To summarize, we have described how to augment a quadtree $\mathcal{T}$ by adding more cells and registering the segments at the cells, yielding a new quadtree $\mathcal{T}^+$. In what follows, we use this stored information to compute the long and short congestions of $\mathcal{T}^+$ (which are at least the congestions of the sub-quadtree $\mathcal{T}$).

3.3. Computing the congestion of the short segments

The α -short congestion is computed using dynamic programming, as described next.

Lemma 3.9. *Given a set $\mathcal{S}$ of n segments in the plane and a quadtree $\mathcal{T}^+$ of size $O(n)$, one can compute, in $O(n)$ time, the α -short congestion for all the nodes of $\mathcal{T}^+$, where α is a constant.*

Proof: We compute the α -short congestion of a quadtree via dynamic programming. The algorithm finds the total length of short segments intersecting each leaf and propagates the values upward.

For every node $v \in \mathcal{T}^+$, the algorithm computes the quantity $\|L_{\text{long}}(v) \cap \square_v\|$. Computing the value for node v requires time proportional to the total size of the list $L_{\text{long}}(v)$, so doing it for all the nodes of the tree takes $O(n)$ time overall. Next, the algorithm traverses the tree bottom-up. For each node along the way, it computes the total lengths of the intersecting short segments, summing over the proper descendants of v :

$$\Delta_v = \sum_{u \in \text{desc}(v) \setminus \{v\}} \|L_{\text{long}}(u) \cap \square_u\| = \sum_{u \text{ child of } v} (\|L_{\text{long}}(u) \cap \square_u\| + \Delta_u).$$

We claim that $\Delta_v = \|\mathcal{S}_{<\alpha}(\square_v) \cap \square_v\|$. Consider a segment $s \in \mathcal{S}_{<\alpha}(\square_v)$, and let τ be the sidelength of the cells of $\mathbf{G}(s, \alpha)$. By the proof of [Lemma 3.6](#), $\alpha\tau/2 \leq \|s\| < \alpha r(\square_v)$, so the sidelength of $\square_v$ is at least 2τ , and $\square_v$ is a union of cells of $\mathbf{G}_\tau$. The cells of $\mathbf{G}(s, \alpha)$ contained in $\square_v$ are proper descendants of

v in $\mathcal{T}^+$, they are interior disjoint, and they cover $s \cap \square_v$, while the other cells of $\mathcal{G}(s, \alpha)$ are interior disjoint from $\square_v$. Thus s contributes exactly $\|s \cap \square_v\|$ to the sum. Conversely, a segment registered at a proper descendant u of v is α -short for $\square_v$, as $r(\square_v) \geq 2r(\square_u) > \|s\|/\alpha$. The lemma follows. ■

4. Approximating the maximum load of the long segments in a quadtree

Handling the long congestion quickly seems challenging. Instead, we quickly approximate a proxy for this quantity – the maximum number of long segments intersecting a single cell.

Definition 4.1. The α -load of a quadtree $\mathcal{T}^+$ is the quantity

$$\rho = \max_{\square \in \text{cells}(\mathcal{T}^+)} |\mathcal{S}_{\geq \alpha}(\square)|.$$

4.1. A naïve algorithm for computing the α -load

To compute the exact load of a given square, we need to find the long segments intersecting it. We can compute the conflict list for each cell by pushing long segments downward from the conflict lists of its ancestors (as done in the naïve algorithm of [Section 3.2.1](#)). Unfortunately, these lists can get quite long.

Lemma 4.2. *One can compute the α -load of $\mathcal{T}^+$ in $O(n \log n + \rho n)$ time. More generally, given a set $R \subseteq \mathcal{S}$ and a threshold t , one can decide whether $\rho(R) = \max_{\square \in \text{cells}(\mathcal{T}^+)} |R \cap \mathcal{S}_{\geq \alpha}(\square)| \leq t$, in $O(n \log n + tn)$ time.*

Proof: The algorithm starts with the precomputed lists $L_{\text{long}}(u)$ and traverses the tree top-down. At each node, it pushes the stored list down to the children. Each child v selects the segments of the incoming list that intersect its cell, and takes the union of this filtered incoming list with $L_{\text{long}}(v)$. This yields the conflict list of v of *all* α -long segments that intersect it, denoted by $L_{\text{LONG}}(v)$. It then pushes this list down to its children, and so on.

For each node v of the quadtree, it is now straightforward to compute the congestion of the segments of $L_{\text{LONG}}(v)$ (or the length of the list $L_{\text{LONG}}(v)$) for the cell of the node v . It follows that this computes the α -long congestion

for each node of the quadtree. Since the maximum length of the lists sent down is ρ , the claim follows.

The second algorithm works in a similar fashion, except that the algorithm propagates downwards only segments of R . If in any point in time the computed conflict list gets bigger than t , the algorithm bails out, returning that $\rho(R) > t$. ■

4.2. From long congestion to load, and back

Instead of computing the α -long congestion of a quadtree exactly, we content ourselves with a constant-factor approximation, by using the load instead.

Lemma 4.3. *For a cell $\square = \square(p, r)$ of $\mathcal{T}^+$, we have $\frac{c_{\geq \alpha}(\square)}{\sqrt{8}} \leq |\mathcal{S}_{\geq \alpha}(\square)| \leq \frac{1 + \alpha}{\alpha} c(\mathcal{S})$.*

Proof: The intersection of each segment of $\mathcal{S}_{\geq \alpha}(\square)$ with $\square$ can have length at most $\sqrt{2} \cdot 2r = \sqrt{8}r$ (as the sidelength of $\square$ is $2r$), so $c_{\geq \alpha}(\square) = \|\mathcal{S}_{\geq \alpha}(\square) \cap \square\| / r$, and $\|\mathcal{S}_{\geq \alpha}(\square) \cap \square\| \leq |\mathcal{S}_{\geq \alpha}(\square)| \cdot r\sqrt{8}$, and the first inequality follows.

As for the second inequality, consider $\square' = \square(p, (1 + \alpha)r)$. By definition, each long segment $s \in \mathcal{S}_{\geq \alpha}(\square)$ has length at least αr and intersects $\square$. The length of $s \cap \square'$ is at least αr , see Figure 4.1. As such, we have

$$|\mathcal{S}_{\geq \alpha}(\square)| \cdot \frac{\alpha r}{(1 + \alpha)r} \leq \frac{\|\mathcal{S}_{\geq \alpha}(\square) \cap \square'\|}{(1 + \alpha)r} \leq c(\square') \leq c(\mathcal{S}). \quad \blacksquare$$

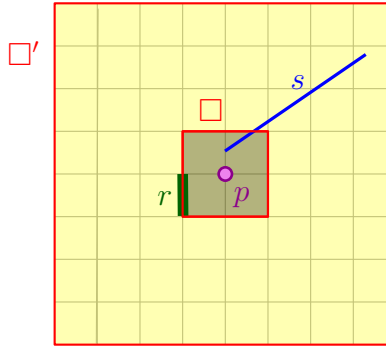


Figure 4.1: An α -long segment for a square $\square = \square(p, r)$ intersects the square $\square' = \square(p, (1 + \alpha)r)$ with a segment of length at least αr (here, $\alpha = 3$).

4.3. Exponential search for maximum α -load

It remains to estimate the load (i.e., maximum number of α -long segments intersecting any cell in the quadtree $\mathcal{T}^+$). The basic idea is to couple random sampling together with exponential search, to approximate the maximum load. If during a round an “overflow” occurs, the algorithm increases the guess for the load (say, by a factor of 2), and moves on to the next round. Using sampling and exponential search to estimate a quantity is an old idea. In the context of geometric settings, it was used before to estimate the maximum depth of nicely behaved regions, see Aronov and Har-Peled [AH08].

4.3.1. The algorithm

Our purpose here is to approximate the α -load of $\mathcal{T}^+$, denoted by ρ , see Definition 4.1.

Estimation via sampling. We perform an exponential search for the size of the largest conflict list. We use the following standard sampling lemma. It follows by a standard application of Chernoff’s inequality – see Beame *et al.* [BHR⁺20, Lemma 2.7] for a proof.

Lemma 4.4. *Consider two (finite) sets $B \subseteq \mathcal{S}$, where $n = |\mathcal{S}|$. Let $\xi \in (0, 1)$ and $\varphi \in (0, 1/2)$ be parameters, and let $r = \lceil c_2 \xi^{-2} \frac{n}{g} \log \varphi^{-1} \rceil$, where c_2 is a sufficiently large constant.*

Let $g > 0$ be a user-provided guess for the size of $|B|$. Consider a random sample R , taken from $\mathcal{S}$ by picking each element with probability $\frac{r}{n}$. Next, consider the estimate $Y = \frac{n}{r} |R \cap B|$ to $|B|$. Then, we have the following:

- (A) *If $Y < g/2$, then $|B| < g$,*
- (B) *If $Y \geq g/2$, then $(1 - \xi)Y \leq |B| \leq (1 + \xi)Y$.*

Both statements above hold with probability $\geq 1 - \varphi$.

The algorithm. The number of cells in $\mathcal{T}^+$ is $O(n)$, and let $\varphi = 1/n^{O(1)}$. Let $\varepsilon \in (0, 1)$ be a prespecified approximation parameter, and let $\xi = \varepsilon/3$. At each round, the algorithm has a threshold number g_i and checks whether (i) $\rho > 8g_i$ (roughly), or (ii) $\rho = \Theta(g_i)$ and it can be approximated reliably and quickly. The i th round starts with parameters

$$g_i = 2^{i-1} \quad \text{and} \quad r_i = \lceil c_2 \xi^{-2} \frac{n}{g_i} \log \varphi^{-1} \rceil, \quad (4.1)$$

where c_2 is a sufficiently large constant. If the i th round failed, the algorithm goes on to the next round (i.e., by increasing i by one).

Let I be the first integer such that $r_i < n$ (i.e., $g_I = \Theta(\xi^{-2} \log n)$ and $I = O(\log \log n + \log \varepsilon^{-1})$), and the algorithm starts with $i = I$. In this first round, the algorithm uses [Lemma 4.2](#) directly. If it finds some conflict list (for $\mathcal{S}$) that contains more than $8g_i = 8g_I$ segments at any point during the execution, it concludes that g_i is too small. Thus this round is a failure, and it goes on to the next round. Otherwise, the algorithm computes ρ exactly and returns its value.

For later rounds, and larger values of g_i , the round begins by taking a random sample $R_i \subseteq \mathcal{S}$ of the segments. Each segment is included in R_i independently with probability $\varsigma_i = r_i/n$ (i.e., a random sample according to [Lemma 4.4](#)). The algorithm then calls the subroutine of [Lemma 4.2](#) with R_i as the list of segments and with $U_i = 8g_i\varsigma_i$ as the threshold. If the output reveals that $\rho(R_i) > U_i$, then the guess g_i is too small, the round failed, and the algorithm continues to the next round.

If a round succeeds (and $i \neq I$), the algorithm outputs $(1 - \xi)Y$ for $Y = \frac{n}{r_i}\rho(R_i)$ as an estimate from below to ρ .

4.3.2. Analysis

Running time. The first round takes $O(\varepsilon^{-2}n \log n)$ time. Since there are only n segments in $\mathcal{S}$, the algorithm always stops by round $O(\log n)$. The threshold used in each round is

$$U_i = 8g_i\varsigma_i = \frac{8g_i r_i}{n} = O\left(\frac{g_i}{n} \cdot \frac{n}{\xi^2 g_i} \log n^{O(1)}\right) = O\left(\frac{\log n}{\varepsilon^2}\right).$$

As such, the running time of each round is $O(\varepsilon^{-2}n \log n)$; see [Lemma 4.2](#), and the overall running time is $O(\varepsilon^{-2}n \log^2 n)$.

The result.

Lemma 4.5. *Using the above randomized algorithm, one can compute, in $O(\varepsilon^{-2}n \log^2 n)$ time, a number Δ , such that with high probability, we have $\Delta \leq \rho \leq (1 + \varepsilon)\Delta$, where ρ is the α -load of $\mathcal{T}^+$, see [Definition 4.1](#).*

Proof: If the algorithm terminated after the first round, then it output ρ exactly, and we are done.

Otherwise, if it failed in a round, it must be that it found a node whose conflict list exceeds the “expected” threshold $U_i = 8g_i\zeta_i$. **Lemma 4.4** implies that the random sample in such cases estimates the size of the original conflict list correctly (its estimate Y exceeds $(n/r_i)U_i = 8g_i \geq g_i/2$), and it is at least of size $(1 - \xi)8g_i > 4g_i$. Namely, g_i is (way) too small, and this decision was made correctly with probability $1 - 1/n^{O(1)}$.

If the algorithm succeeded in a round, then **Lemma 4.4** implies that $\rho \leq (1 + \xi)8g_i < 16g_i$, as every estimate is at most $(n/r_i)U_i = 8g_i$. Since the algorithm failed the previous round, we have that $\rho > 4g_{i-1} \geq 2g_i$. **Lemma 4.4** then implies that ρ is $(1 \pm \xi)$ -estimated correctly. Specifically, for the j th “heavy” node (i.e., conflict list size ρ_j is larger than g_i) in the tree, let Y_j be the estimate of its conflict list size. We have that $(1 - \xi)Y_j \leq \rho_j \leq (1 + \xi)Y_j$. This inequality clearly holds also on the max value, which implies that

$$(1 - \xi)Y \leq \rho \leq (1 + \xi)Y \implies \Delta = (1 - \xi)Y \leq \rho \leq \frac{1 + \xi}{1 - \xi} \Delta \leq (1 + 3\xi)\Delta = (1 + \varepsilon)\Delta,$$

as $\xi = \varepsilon/3 < 1/3$.

An important technicality here is that any of the conflict lists in the tree of size smaller than $g_i/2$ are too small after the sampling to compete with the conflict list realizing ρ . Similarly, this also holds for conflict lists of size $\leq g_i$. Thus, all the conflict lists in play for realizing the maximum of the sample are estimated correctly². ■

5. Approximating the maximum congestion

We seem to have lost our way, so let us get back on track. Consider the following quantity

$$\mathcal{C}_{\geq \alpha}(\mathcal{T}^+) = \max_{\square \in \text{cells}(\mathcal{T}^+)} \max[c_{\geq \alpha}(\square), c_{\mathcal{S}_{\geq \alpha}(\square)}((1 + \alpha)\square)],$$

which is the *augmented* long congestion of $\mathcal{T}^+$. We define³

$$\mathcal{C}_{\mathcal{S}}(\mathcal{T}^+) = \max(c_{< \alpha}(\mathcal{T}^+), \mathcal{C}_{\geq \alpha}(\mathcal{T}^+)).$$

²So, confusingly, while the winning estimate might not be from the node with the largest conflict list, it comes from a node with a conflict list of size very close to it – if it was much smaller, it would not have been able to beat it.

³Spoiler alert! The hero is still dead, and it turns out that for $\mathcal{T}_1, \dots, \mathcal{T}_m$ the quadrees of **Lemma 3.3**, $\max_i \mathcal{C}_{\geq \alpha}(\mathcal{T}_i)$ is a good approximation to $c(\mathcal{S})$.

Clearly, $c_S(\mathcal{T}^+)/2 \leq \mathcal{C}_S(\mathcal{T}^+) \leq c(\mathcal{S})$.

Lemma 5.1. *We have $\frac{\alpha}{1+\alpha}\rho \leq \mathcal{C}_{\geq\alpha}(\mathcal{T}^+) \leq \sqrt{8}\rho$.*

Proof: The upper bound is immediate. The lower bound readily follows from the argument used in the proof of [Lemma 4.3](#). ■

Lemma 5.2. *One can compute, in $O(n \log^2 n)$ time, a quantity $\mathcal{A}_S(\mathcal{T}^+)$, such that $\mathcal{A}_S(\mathcal{T}^+) \leq \mathcal{C}_S(\mathcal{T}^+) \leq 3\mathcal{A}_S(\mathcal{T}^+)$. This holds with high probability.*

Proof: Let $\varepsilon = 1/100$, and let Δ be the approximation of ρ computed by the algorithm of [Lemma 4.5](#). This takes $O(n \log^2 n)$ time. We now compute $c_{<\alpha}(\mathcal{T}^+)$ in $O(n)$ time using the algorithm of [Lemma 3.9](#). We compute the quantity

$$\mathcal{A}_S(\mathcal{T}^+) = \max\left(c_{<\alpha}(\mathcal{T}^+), \frac{\alpha}{1+\alpha}\Delta\right).$$

Clearly, $\mathcal{A}_S(\mathcal{T}^+) \leq \mathcal{C}_S(\mathcal{T}^+)$. As for the other direction, observe that

$$\frac{\sqrt{8}(1+\varepsilon)(1+\alpha)}{\alpha}\mathcal{A}_S(\mathcal{T}^+) \geq \frac{\alpha}{1+\alpha}\Delta \cdot \frac{\sqrt{8}(1+\varepsilon)(1+\alpha)}{\alpha} \geq \sqrt{8}\rho \geq \mathcal{C}_{\geq\alpha}(\mathcal{T}^+),$$

by [Lemma 5.1](#). Numerical calculations show that for $\varepsilon = 1/100$ and $\alpha = 20$, $\frac{\sqrt{8}(1+\varepsilon)(1+\alpha)}{\alpha} \leq 3$, which establishes the claim. ■

Finally, we arrive at our main result.

Theorem 5.3. *Let $\mathcal{S}$ be a set of n segments in the plane. One can compute, in $O(n \log^3 n)$ time, a 42-approximation to $c(\mathcal{S})$. The algorithm is randomized and succeeds with high probability.*

Proof: Compute, in $O(n \log^2 n)$ time, the $m = O(\log n)$ quadtrees $\mathcal{T}_1, \dots, \mathcal{T}_m$ of [Lemma 3.3](#). Each quadtree $\mathcal{T}_i$ is refined as described in [Section 3.2.3](#), into a quadtree $\mathcal{T}_i^+$, and we compute for each one of them the quantity $\zeta_i = \mathcal{A}_S(\mathcal{T}_i^+)$, using the algorithm of [Lemma 5.2](#) in $O(n \log^2 n)$ time. Overall, the running time is $O(n \log^3 n)$, and all the steps so far succeeded with high probability. The algorithm returns $\zeta = \max_i \zeta_i \leq c(\mathcal{S})$ as the desired approximation to the congestion. For the quality of approximation, observe that

$$42\zeta = 14 \max_i 3\zeta_i \geq 14 \max_i \mathcal{C}_S(\mathcal{T}_i^+) \geq 7 \max_i c_S(\mathcal{T}_i^+) \geq 7 \cdot \frac{1}{7} c(\mathcal{S}) = c(\mathcal{S}). \quad \blacksquare$$

6. A $(1 + \varepsilon)$ -approximation in near-linear time

In this section we improve the approximation factor to $1 + \varepsilon$, for any fixed $\varepsilon > 0$, while keeping the running time near-linear. The guarantee is for the axis-parallel square congestion of [Definition 2.4](#). Specifically, we show ([Theorem 6.7](#)) that for a set $\mathcal{S}$ of n segments in the plane, and any fixed $q > 0$, one can compute a number A with $A \leq c(\mathcal{S}) \leq (1 + \varepsilon)A$, with probability at least $1 - (n + 2)^{-q}$, in $O_{\varepsilon,q}(n \log^6(n + 2))$ time in the worst case and $O_{\varepsilon,q}(n \log^5(n + 2))$ expected time.

The algorithm of the preceding sections only estimates the congestion of fixed squares, namely quadtree cells and their inflated copies, and it uses the load of a cell (a count of long segments) as a proxy for its long congestion. Here every small long-segment list is evaluated twice: clipped to its own cell, and as a free-standing input whose congestion is maximized over all squares, with center and radius unconstrained. The second evaluation is what allows us to recover an almost optimal square whose center is far from any endpoint. The section is organized as follows. We first show that the congestion of any set of k segments is realized by one of $O(k^3)$ squares, so the congestion of a small list can be computed exactly ([Lemma 6.2](#)). We then prove a sampling lemma for lengths, rather than counts, showing that a random sample of the segments has congestion close to the scaled congestion of the input, once the latter is large enough ([Lemma 6.3](#)). Next, we describe a refined family of candidate squares, built from a well-separated pair decomposition of the endpoints, and an evaluator that computes the two values above for every cell of the quadtrees built for this family. For an accuracy parameter ξ , its output is a $(1 - 3\xi)$ -approximation to the congestion of its input, provided all long lists are small ([Lemma 6.6](#)). Finally, an exponential search over sampling rates, as in [Section 4](#), finds a sample that is dense enough to be accurate, and sparse enough for all long lists to be small ([Section 6.5](#)). See [Algorithm 6.1](#) for the resulting loop.

Conventions. We use the notation of [Section 2](#). We work in the real RAM, as the preceding sections implicitly do, where arithmetic operations, square roots, the floor function, and comparisons take unit time. In particular, [Lemma 6.2](#) is exact in this model, as the sum-of-square-roots issue mentioned in the introduction does not arise when square roots are exact. The input $\mathcal{S}$ is a multiset of segments, that is, segments count with multiplicity. We

discard zero-length segments, and for an empty input the algorithm returns zero. For a single segment s , we write $c_s(\square) = \|s \cap \square\| / r(\square)$. A segment meets a square $\square$ in a chord of length at most the diameter $\sqrt{8}r(\square)$ of $\square$, so for every segment s and square $\square$ we have

$$0 \leq c_s(\square) \leq \sqrt{8} < 3, \quad (6.1)$$

see [Definition 2.4](#).

6.1. Tools from the preceding sections

Tight containment. Let Ξ be a finite family of squares, and let $\delta \in (0, 1/2)$ and $\varphi \in (0, 1)$ be parameters. By [Lemma 3.1](#), for a fixed $\square \in \Xi$, a randomly shifted and scaled quadtree has, with probability at least $(\delta/8)^3$, a cell that contains $\square$ and has radius at most $(1 + \delta)r(\square)$. Repeating the construction $O(\delta^{-3} \log((|\Xi| + 2)/\varphi))$ times and applying the union bound, as in the proof of [Lemma 3.3](#), we get a collection of shifted and scaled compressed quadtrees such that, with probability at least $1 - \varphi$, every $\square \in \Xi$ is contained in a cell of one of the trees of radius at most $(1 + \delta)r(\square)$. We refer to this collection as the *tight containment* of Ξ with parameters δ and φ . We register the n input segments in each tree as in [Section 3.2.3](#). Each tree then has $O(|\Xi| + n)$ cells, which we call its *retained cells*, and its construction takes $O((|\Xi| + n) \log(|\Xi| + n + 2))$ time.

Exact evaluation with bounded lists. Throughout this section we use the long/short threshold $\alpha = 1$. That is, a segment is long for a square $\square$ if it intersects the interior of $\square$ and has length at least $r(\square)$, and $\mathcal{S}_{\geq 1}(\square)$ is the list of long segments of $\square$, see [Section 3.2.3](#). A segment lying along a side of a cell does not meet the interior of that cell, so it is in neither its short nor its long list, although it contributes to the clipped length of the (closed) cell. With the continuous random shift of the quadtree this happens with probability zero, and we ignore it. Consider a tree with N retained cells. The dynamic program of [Lemma 3.9](#) computes the short congestion of every cell exactly. The top-down traversal of [Lemma 4.2](#), with threshold t , computes the long list of every cell in $O((n + N) \log(n + N + 2) + Nt)$ time if all these lists have size at most t , and we stop it as soon as some list exceeds t . If no list exceeds t , the exact congestion of a cell is its short congestion plus the total length of its long segments clipped to the cell, divided by the radius of the cell.

Observation 6.1. *For every square $\square$, $|\mathcal{S}_{\geq 1}(\square)| \leq 2c(\mathcal{S})$.*

This is [Lemma 4.3](#) with $\alpha = 1$: every member of $\mathcal{S}_{\geq 1}(\square)$ contributes length at least $r(\square)$ to the doubled square $2\square$, whose radius is $2r(\square)$.

Well-separated pairs. We also use the well-separated pair decomposition (WSPD) of a planar point set, see Har-Peled [[Har11](#)]. The separation convention we need is spelled out below, where the candidate squares are constructed.

6.2. One finite family captures every subset optimum

Lemma 6.2 (A universal finite family). *Let $k \geq 0$, and consider a multiset of k segments in the plane. There is a family $\mathcal{V}$ of at most*

$$f(k) = (12k + 6)^3 \tag{6.2}$$

squares, depending only on these segments, such that for every subset $\mathcal{U}$ of them,

$$c(\mathcal{U}) = \max_{\square \in \mathcal{V}} c_{\mathcal{U}}(\square).$$

Furthermore, the congestion of the k segments, and a square realizing it, can be computed exactly in $O((k+1)^4)$ time.

Proof: We construct an arrangement in the three-dimensional space of squares, depending only on the full input. Its vertices provide the candidate squares for every subset simultaneously. Zero-length segments may be discarded; if none remain, any square suffices.

Identify the square

$$\square = \square((x, y), r) = [x - r, x + r] \times [y - r, y + r]$$

with the point $(x, y, r) \in \mathbb{R}^3$, where $r > 0$. Its center is (x, y) , its radius is r , and its four corners are $(x + \sigma_1 r, y + \sigma_2 r)$, for $\sigma_1, \sigma_2 \in \{-1, 1\}$.

For each input segment, introduce the planes corresponding to the following two types of events:

- (I) *A segment endpoint lies on a square side.* For an endpoint $u = (u_x, u_y)$, the four corresponding planes are

$$x - r = u_x, \quad x + r = u_x, \quad y - r = u_y, \quad y + r = u_y.$$

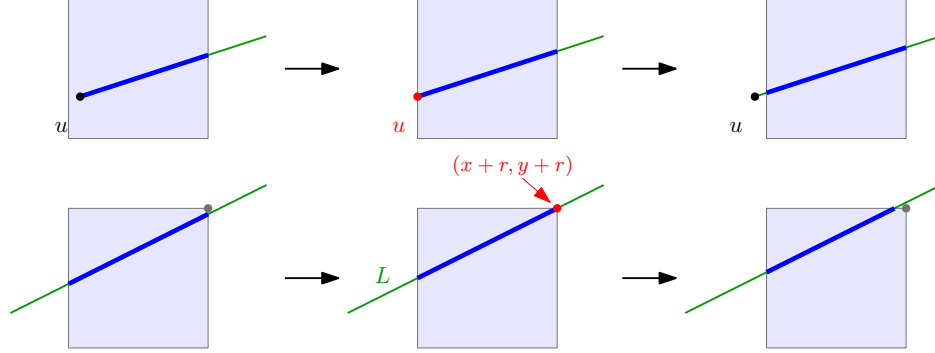


Figure 6.1: The clipping events in [Lemma 6.2](#). Top row: endpoint u crosses a square side ($x - r = u_x$). Bottom row: a square corner crosses the segment's supporting line L (green). Arrows show progression as the square moves right; blue highlights the clipped segment.

- (II) *A square corner lies on the segment's supporting line.* If this line has equation $a_1X + a_2Y = a_0$, the four corresponding planes are

$$a_1x + a_2y + (a_1\sigma_1 + a_2\sigma_2)r = a_0, \quad \sigma_1, \sigma_2 \in \{-1, 1\}.$$

See [Figure 6.1](#). Thus each segment contributes at most twelve planes. Using the entire planes merely refines the partition beyond the actual events.

Let $B = [x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ be the input's bounding rectangle, and let R be half its larger side length. For every subset $\mathcal{U}$, the radius- R square enclosing B has congestion $\|\mathcal{U}\|/R$, at least that of any square of larger radius. Projecting a center into B cannot increase its coordinatewise distance to any input point, so it cannot decrease clipped length. We may therefore clip the arrangement to the box $B \times [0, R]$. Including its six boundary planes gives at most $12k + 6$ planes in total.

Within the relative interior of any face F of this three-dimensional arrangement in $r > 0$, the clipping pattern is fixed. Each clipped endpoint is either an original segment endpoint or the intersection of the segment with a fixed square side. Its position along the segment is therefore affine in (x, y, r) , and so is the clipped length. Consequently, for every subset $\mathcal{U}$, its total clipped length on F is an affine function $W_F(x, y, r)$, and its congestion is

$$c_{\mathcal{U}}\left(\square((x, y), r)\right) = \frac{W_F(x, y, r)}{r}. \quad (6.3)$$

This holds also on lower-dimensional faces, including those where a horizontal or vertical segment lies on a square side.

Extend W_F affinely to the closure of F . Fixing $r = \rho > 0$, maximizing congestion is equivalent to maximizing W_F on the bounded slice $\overline{F} \cap \{r = \rho\}$. This affine function attains its maximum at a vertex of the slice. If this point is not already a vertex of $\overline{F}$, it lies on an edge transverse to the slice. Along this edge, x and y are affine functions of r , so the congestion has the form

$$\frac{W_F(x(r), y(r), r)}{r} = \lambda_0 + \frac{\lambda_1}{r}.$$

This is monotone or constant, so its maximum is at an endpoint. If an endpoint has $r = 0$, then $\lambda_1 \leq 0$, as otherwise W_F/r would be unbounded on F near this endpoint. The function is then nondecreasing in r , so the positive-radius endpoint is at least as good. At a boundary vertex the actual clipped length is at least its limit from F , because the square is closed. Thus some positive-radius vertex has congestion at least that of the original square.

Let $\mathcal{V}$ contain the squares of all positive-radius vertices of the clipped arrangement, together with one arbitrary square to cover the zero-congestion case. The arrangement depends only on the full input, so this family works for every subset simultaneously. It has at most $\binom{12k+6}{3} + 1 \leq f(k)$ members. Enumerating all triples of planes with independent normals produces all vertices, including those of degenerate arrangements. Evaluating the k clipped lengths at every candidate gives the exact maximum in $O((k+1)^4)$ time. Empty input returns zero. ■

We use [Lemma 6.2](#) in two ways. In the analysis of the sampling, in the next subsection, we apply it to the whole input $\mathcal{S}$ (so $k = n$). The resulting family $\mathcal{V}$ is determined by $\mathcal{S}$, and thus is fixed before any sampling takes place. This family is never constructed, as only its existence and its size are used. In the algorithm, we apply the lemma to the list of long segments of a cell, which has at most H_0 segments, where $H_0 = O_{\varepsilon, q}(\log n)$ is the cap set in [Section 6.5](#), and there we do enumerate the vertices of its arrangement and compute the congestion of the list exactly.

6.3. Sampling at geometrically decreasing rates

Assume that $c(\mathcal{S}) > 0$ (recall that the empty input is handled separately). Let $0 < \xi \leq 1/20$ be an accuracy parameter, and let $0 < \varphi < 1$ be the

allowed failure probability. Define

$$J = \lceil \log_2(n+2) \rceil, \quad h = \log\left(2f(n)\frac{J+1}{\varphi}\right), \quad \text{and} \quad T = \frac{24h}{\xi^2}. \quad (6.4)$$

where $f(n) = n^{O(1)}$, see [Eq. \(6.2\)](#).

The threshold T depends only on n , ξ and φ . The algorithm of [Section 6.5](#) uses T as the threshold of its stopping rule. For $j = 0, \dots, J$, let $R_j \subseteq \mathcal{S}$ be a random sample that includes each segment of $\mathcal{S}$ independently with probability 2^{-j} . The samples of different rounds are independent, and the algorithm computes a sample only when its round is reached.

Lemma 6.3. *With probability at least $1 - \varphi$, we have, simultaneously for all j , that*

$$|c(R_j) - 2^{-j}c(\mathcal{S})| \leq \xi \max\{2^{-j}c(\mathcal{S}), T\}. \quad (6.5)$$

Proof: For a fixed round $j \in \{0, \dots, J\}$ and square $\square \in \mathcal{V}$ from [Lemma 6.2](#), set $X_j = \sum_{s \in \mathcal{S}} (I_s - 2^{-j})c_s(\square)$, where I_s is the sampling indicator of s . Let $p = \mathbb{E}[I_s] = 1/2^j$. We have $\mathbb{V}[I_s - p] = p(1 - p) \leq p$. Then, by the independence of these indicator variables, we have

$$\begin{aligned} \mathbb{V}[X_j] &= \sum_{s \in \mathcal{S}} \mathbb{V}[(I_s - 2^{-j})c_s(\square)] \leq \sum_{s \in \mathcal{S}} c_s(\square)^2 \mathbb{V}[I_s] \\ &\leq p \sum_{s \in \mathcal{S}} c_s(\square)^2 \leq 3 \cdot 2^{-j} \sum_{s \in \mathcal{S}} c_s(\square) \leq 3 \cdot 2^{-j}c(\mathcal{S}), \end{aligned}$$

as $c_s(\square) \leq \sqrt{8} \leq 3$. Furthermore, $|(I_s - 2^{-j})c_s(\square)| \leq 3$. Put $V_j = \max\{2^{-j}c(\mathcal{S}), T\}$. [Bernstein's inequality](#), with $T = 24h/\xi^2$, gives

$$\begin{aligned} \mathbb{P}[|X_j| > \xi V_j] &\leq 2 \exp\left(-\frac{\xi^2 V_j^2}{6 \cdot 2^{-j}c(\mathcal{S}) + 2\xi V_j}\right) \\ &\leq 2 \exp\left(-\frac{\xi^2 T}{6 + 2}\right) \leq 2 \exp(-3h). \end{aligned}$$

Union bound over at most $f(n)(J+1)$ choices of square and round gives the asserted probability. Both $c(\mathcal{S})$ and $c(R_j)$ are maxima over the same family $\mathcal{V}$, so the deviation bound at those squares implies [Eq. \(6.5\)](#). $\blacksquare$

We do not need a preliminary estimate of $c(\mathcal{S})$ to choose these rates or the threshold T . The stopping rule in [Section 6.5](#) will certify that the accepted sample is sufficiently dense.

6.4. Evaluation with bounded conflict lists

We now describe the *evaluator*, the procedure that the final algorithm runs on each sample. Since it is applied to samples of the input rather than to the input itself, in this subsection the input is a multiset $\mathcal{S}$ of m segments. Let P be the set of distinct endpoints of the segments of $\mathcal{S}$, and fix an accuracy parameter $0 < \xi \leq 1/10$. The evaluator also receives a failure probability $0 < \varphi < 1$, and a cap $H_0 \geq 1$ on the size of the long lists it is willing to handle. The cap bounds the running time, and it is not an a priori bound on $c(\mathcal{S})$. If some long list has more than H_0 segments, the evaluator reports an overflow, which by [Observation 6.1](#) certifies that $c(\mathcal{S}) > H_0/2$. The evaluator first constructs a family Ξ of $O_\xi(m)$ candidate squares, then builds the tight containment of Ξ , and finally computes two values for every retained cell, as described next.

6.4.1. The candidate squares

In the construction of the candidate squares, and in its analysis, distances and diameters are measured in the L_∞ norm. Set $\eta = \xi^2/100$, and build a WSPD of P with separation parameter η . That is, a set of $O(m\eta^{-2})$ pairs (P_1, P_2) of subsets of P , such that

$$\max\{\text{diam}(P_1), \text{diam}(P_2)\} \leq \eta \text{dist}_\infty(P_1, P_2)$$

for each pair (P_1, P_2) , and such that every two distinct points of P are covered by some pair. Each pair (P_1, P_2) comes with representatives $a \in P_1$ and $a' \in P_2$, and we denote $d = \|a - a'\|_\infty > 0$. The pair contributes squares to Ξ at the scales

$$t_\ell = \frac{(1 + \xi)^\ell}{64}, \quad 0 \leq \ell \leq J_{\text{geo}}, \quad J_{\text{geo}} = \lceil \log_{1+\xi}(128/\xi) \rceil.$$

Specifically, for each scale t_ℓ , let $R = t_\ell d$, and add to Ξ the grid of squares around a

$$\square(a + \xi R(i, j), (1 + \xi)R), \quad i, j \in \mathbb{Z}, \quad |i|, |j| \leq \lceil 2/\xi \rceil, \quad (6.6)$$

and the corresponding grid of squares around a' . The family Ξ is the set of all the squares added in this way, over all pairs and all scales. It has $O(m\xi^{-7} \log(1/\xi))$ squares.

Lemma 6.4 (Ordinary scales). *Suppose $\square = \square(p, r)$ contains an endpoint v and there is an endpoint w with*

$$\xi r \leq \|v - w\|_\infty \leq 32r. \quad (6.7)$$

Then some square of Ξ contains $\square$ and has radius at most $(1 + \xi)^2 r$.

Proof: Choose the WSPD pair covering v, w , and orient it so that v belongs to the cluster represented by a . Put $d_0 = \|v - w\|_\infty$. Separation gives

$$|d - d_0| \leq 2\eta d_0, \quad \|a - v\|_\infty \leq \eta d_0 \leq 32\eta r \leq \xi r.$$

It follows that $r/d \in [1/64, 2/\xi]$. Some chosen scale R therefore satisfies $r \leq R \leq (1 + \xi)r$. Furthermore, $\|p - a\|_\infty \leq (1 + \xi)r < 2R$. A grid center in [Eq. \(6.6\)](#) lies within ξR of p , so its square of radius $(1 + \xi)R$ contains $\square$. Its radius is at most $(1 + \xi)^2 r$. $\blacksquare$

6.4.2. The same squares handle isolated endpoint clusters

The following lemma handles squares for which [Lemma 6.4](#) does not apply. In this case the endpoints near $\square$ form a tiny cluster, and the lemma shows that a square $\square' \in \Xi$ around this cluster has the following property. For any cell C that tightly contains $\square'$ (as some retained cell of the tight containment does), the congestion of the long list of C , optimized over all squares, almost recovers the congestion of $\square$.

Lemma 6.5 (Isolated scales). *Let $\square = \square(p, r)$ contain an endpoint v , and suppose that [Eq. \(6.7\)](#) fails for every endpoint w . Then one of the following holds:*

- (I) $c_{\mathcal{S}}(\square) \leq 2\xi c(\mathcal{S})$, or
- (II) *there is a square $\square' \in \Xi$ such that every square C with $\square' \subseteq C$ and $r(C) \leq (1 + \xi)r(\square')$ satisfies*

$$c(\mathcal{S}_{\geq 1}(C)) \geq c_{\mathcal{S}}(\square) - 2\xi c(\mathcal{S}). \quad (6.8)$$

Proof: The endpoints near v form a tiny cluster U surrounded by an empty annulus. A segment meeting $\square$ either has both endpoints in U , and then it is so short that it contributes little to the congestion of $\square$, or it has an endpoint outside the annulus, and then it is long for a suitable candidate square around the cluster. Formally, let $U = P \cap \square(v, \xi r)$. Since [Eq. \(6.7\)](#)

fails for every endpoint w , every endpoint either lies in U , at distance less than ξr from v , or has distance greater than $32r$ from v . A segment with both endpoints in U lies in $\square(v, \xi r)$, so the total length of all such segments is at most $\xi r c(\mathcal{S})$, and their contribution to $c_{\mathcal{S}}(\square)$ is at most $\xi c(\mathcal{S})$. If $U = P$, every segment is of this kind, which gives the first alternative.

Suppose that $U \neq P$. Choose the nearest endpoint $w \notin U$ to v in L_{∞} , and put $d_0 = \|v - w\|_{\infty} > 32r$. Take the WSPD pair covering v, w , with representative a on v 's side and representative distance d . Separation gives

$$|d - d_0| \leq 2\eta d_0, \quad \|a - v\|_{\infty} \leq \eta d_0 < d_0.$$

By the choice of w , the last inequality forces $a \in U$, so $\|a - v\|_{\infty} < \xi r$.

The scales include some R with $d/8 \leq R \leq (1 + \xi)d/8$. Choose the candidate in [Eq. \(6.6\)](#) centered exactly at a , with radius $r' = (1 + \xi)R$. Since $\xi \leq 1/10$,

$$d/8 \leq r' \leq d/6, \quad r' \geq (1 - 2\eta)d_0/8 > (2 + \xi)r. \quad (6.9)$$

As $v \in \square$, we have $\|p - a\|_{\infty} \leq (1 + \xi)r$, so the square $\square$ lies in the interior of $\square(a, r') = \square'$. Every outside endpoint $z \notin U$ satisfies

$$\|z - a\|_{\infty} \geq d_0 - \xi r > (1 - \xi/32)d_0 \geq d/2 \geq 3r'.$$

Thus a segment meeting $\square$ and having an endpoint outside U has length at least $2r'$, and it belongs to $\mathcal{S}_{\geq 1}(C)$ for any C as in the statement, since it meets the interior of $C \supseteq \square'$ and $r(C) \leq (1 + \xi)r' < 2r'$.

Hence every segment meeting $\square$, except those with both endpoints in U , belongs to $\mathcal{S}_{\geq 1}(C)$. By the first paragraph,

$$c(\mathcal{S}_{\geq 1}(C)) \geq c_{\mathcal{S}_{\geq 1}(C)}(\square) \geq c_{\mathcal{S}}(\square) - \xi c(\mathcal{S}),$$

which is stronger than [Eq. \(6.8\)](#). ■

Note that the cluster U and the endpoint w are used only in the analysis. The algorithm does not need to identify isolated clusters, as the family Ξ already contains the square $\square'$.

6.4.3. Evaluation and approximation guarantee

The evaluator builds the tight containment of Ξ with parameters $\delta = \xi$ and φ , and registers the segments of $\mathcal{S}$ in each tree with the long threshold

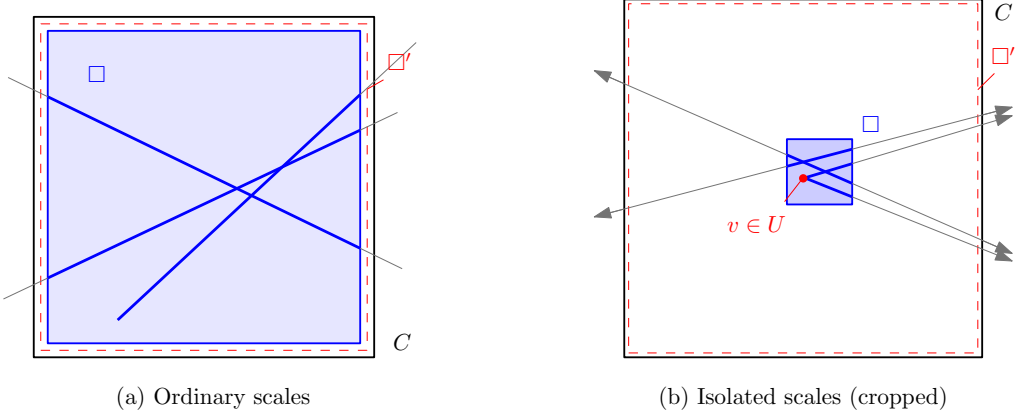


Figure 6.2: (a) $\square \subseteq \square' \subseteq C$; evaluate $c_{\mathcal{S}}(C)$. (b) Evaluate $c(\mathcal{S}_{\geq 1}(C))$ over all squares. Here $\mathcal{S}$ is the depicted segment set, and the red dot represents the cluster U containing v . Arrows continue to endpoints outside U , more than $32r(\square)$ from v .

$\alpha = 1$. It then computes the long list $\mathcal{S}_{\geq 1}(C)$ of every retained cell C by the traversal of Lemma 4.2 with threshold H_0 , and reports an overflow if some list exceeds H_0 . Otherwise, for every retained cell C , it computes the following two values.

- (I) The exact congestion $c_{\mathcal{S}}(C)$ of the cell. This is its exact short congestion plus the lengths of the segments of $\mathcal{S}_{\geq 1}(C)$ clipped to C , divided by the radius of C .
- (II) The congestion $c(\mathcal{S}_{\geq 1}(C))$ of the segments of its long list, computed exactly using Lemma 6.2. Here the segments are not clipped to C , and the maximum is over all squares. In particular, the square realizing this maximum may be far smaller than C , or have a different center. Its location is not constrained by the tree.

The two cases are illustrated in Figure 6.2. The evaluator returns the largest of all these values, denoted by B . Every cell is a square and every long list is a subset of $\mathcal{S}$, so both kinds of values are at most $c(\mathcal{S})$, and $B \leq c(\mathcal{S})$.

Lemma 6.6 (Bounded-list evaluator). *Given a multiset $\mathcal{S}$ of m segments and a cap $H_0 \geq 1$, the evaluator either reports an overflow, which certifies that $c(\mathcal{S}) > H_0/2$, or returns a value $B \leq c(\mathcal{S})$. With probability at*

least $1 - \varphi$, a returned value also satisfies

$$B \geq (1 - 3\xi)c(\mathcal{S}).$$

The running time is

$$O_\xi(m \log^2((m+2)/\varphi) + m(H_0 + 1)^4 \log((m+2)/\varphi)).$$

For an empty input the evaluator returns zero immediately.

Proof: The lower bound on B is the heart of the proof. We take a square $\square$ with congestion close to $c(\mathcal{S})$, argue that we can assume that it contains an endpoint, and then apply [Lemma 6.4](#) or [Lemma 6.5](#), depending on whether or not $\square$ has an endpoint w satisfying [Eq. \(6.7\)](#), to show that one of the values computed by the evaluator is almost $c_{\mathcal{S}}(\square)$.

Condition on the event that every square of Ξ is contained in a retained cell of radius at most $(1+\xi)$ times its own, and suppose that no list overflows. Consider a square $\square = \square(p, \rho)$ whose congestion is arbitrarily close to $c(\mathcal{S})$. If $\square$ contains no endpoint, enlarge it concentrically until its boundary meets an endpoint. We claim that the congestion does not decrease. Indeed, while no endpoint is inside, every contributing segment meets the square in the full chord of its supporting line L , and scaling by $1/\rho$ about p shows that $\|L \cap \square(p, \rho)\|/\rho$ is the chord length of the fixed square $\square(p, 1)$ along a line parallel to L at Euclidean distance $\text{dist}(p, L)/\rho$ from p . This distance does not increase as ρ grows, and the chord lengths of a centrally symmetric convex region along parallel lines form an even concave function of the offset, so they do not decrease as the offset moves toward zero. Segments that start meeting the square during the enlargement only add length. Thus we may assume that $\square = \square(p, r)$ contains an endpoint v .

If there is an endpoint w satisfying [Eq. \(6.7\)](#), then by [Lemma 6.4](#) some square of Ξ contains $\square$ and has radius at most $(1+\xi)^2 r$, and by tight containment some retained cell C contains this square and has radius at most $(1+\xi)^3 r$. Since $\square \subseteq C$, the exact congestion of C , which is the first value computed by the evaluator, is at least

$$\frac{c_{\mathcal{S}}(\square)}{(1+\xi)^3} \geq (1-3\xi)c_{\mathcal{S}}(\square).$$

Otherwise, [Eq. \(6.7\)](#) fails for every endpoint w , and [Lemma 6.5](#) applies. Its first alternative, $c_{\mathcal{S}}(\square) \leq 2\xi c(\mathcal{S})$, cannot hold once $c_{\mathcal{S}}(\square)$ is close enough to

$c(\mathcal{S})$, which is positive for a nonempty input. So there is a square $\square' \in \Xi$ as in the second alternative, and by tight containment some retained cell C contains $\square'$ and has radius at most $(1 + \xi)r(\square')$. By [Eq. \(6.8\)](#), the second value computed by the evaluator for C , namely $c(\mathcal{S}_{\geq 1}(C))$, is at least $c_{\mathcal{S}}(\square) - 2\xi c(\mathcal{S})$. In both cases, letting $c_{\mathcal{S}}(\square)$ tend to $c(\mathcal{S})$ proves $B \geq (1 - 3\xi)c(\mathcal{S})$. The upper bound $B \leq c(\mathcal{S})$ holds unconditionally, as every cell is a square and every list is a subset of the input.

As for the running time, the traversal stops on the first list exceeding H_0 , and such a list certifies $c(\mathcal{S}) > H_0/2$ by [Observation 6.1](#). Otherwise, there are $O_{\xi}(\log((m + 2)/\varphi))$ trees with $O_{\xi}(m)$ retained cells each, and the exact optimization of [Lemma 6.2](#) costs $O((H_0 + 1)^4)$ per cell, which gives the second term of the bound. The construction of the trees, the registration of the segments, the computation of the short congestions, and the enumeration of the long lists all fit in the stated bound. ■

6.5. A simple stopping rule

We now put everything together, reusing the sampling and exponential-search framework of [Section 4](#), where we now test the congestion of the sample instead of its maximum long list size. The algorithm runs the evaluator on samples of decreasing density. A round is rejected if its sample is too dense, that is, if a long list overflows or the returned value exceeds the threshold T .

Algorithm 6.1 (Sampling for square congestion). Given n segments $\mathcal{S}$, an approximation parameter $\varepsilon > 0$, and a confidence parameter $q > 0$:

- (A) Set $\xi = \min\{\varepsilon, 1\}/20$ and $\varphi = \frac{1}{2}(n+2)^{-q}$. Compute J and T from [Eq. \(6.4\)](#), and set $H_0 = \lceil 2T \rceil$.
- (B) For $j = 0, 1, \dots, J$, in this order:
 - (i) Form R_j by including each segment of $\mathcal{S}$ independently with probability 2^{-j} .
 - (ii) Run the evaluator of [Lemma 6.6](#) on R_j with cap H_0 and failure probability $\varphi/(J+1)$.
 - (iii) If it reports an overflow or returns $B_j > T$, continue with the next round. Otherwise, stop and return

$$A = \frac{1 - \xi}{1 + \xi} 2^j B_j. \quad (6.10)$$

- (C) Return zero if all rounds were rejected.

The final return of zero only guarantees that the algorithm terminates within the stated running time. We show below that, when the algorithm succeeds, some round is accepted, so the fallback is never used.

Theorem 6.7. *Let $\mathcal{S}$ be a set of n segments in the plane, and let $\varepsilon > 0$ and $q > 0$ be fixed. The above algorithm returns a number A such that*

$$A \leq c(\mathcal{S}) \leq (1 + \varepsilon)A$$

with probability at least $1 - (n+2)^{-q}$. Its running time is $O_{\varepsilon,q}(n \log^6(n+2))$ in the worst case and $O_{\varepsilon,q}(n \log^5(n+2))$ in expectation, in the same model of computation as in the preceding sections. In particular, the number of candidate scales, and the running time, do not depend on the spread of the coordinates of the input.

Proof: The proof has three parts. We first show that some round is accepted. We then show that the sample of the accepted round is dense enough for [Eq. \(6.5\)](#) to give a relative error, which yields the approximation guarantee. Finally, we bound the running time.

Condition on the event of [Lemma 6.3](#), for all rounds simultaneously, and on the success of the tight containment in every evaluator that is executed. By the union bound, the total failure probability is at most $2\varphi = (n+2)^{-q}$.

Since $c(\mathcal{S}) \leq 3n$, by [Eq. \(6.1\)](#), and $2^{-J} \leq 1/(n+2)$, we have $2^{-J}c(\mathcal{S}) \leq 3$. The definition of T gives $3 \leq (1 - \xi)T$, and therefore, by [Eq. \(6.5\)](#),

$$c(R_J) \leq 2^{-J}c(\mathcal{S}) + \xi T \leq T.$$

By [Observation 6.1](#), every long list in round J has size at most $2T \leq H_0$, so no list overflows, and the returned value satisfies $B_J \leq c(R_J) \leq T$. Consequently, round J is accepted if it is reached, and in any case some round is accepted.

Let j be the index of the accepted round, and suppose first that $j > 0$. Round $j-1$ was rejected, and this certifies that $c(R_{j-1}) > T$. Indeed, either a list overflowed, and then $c(R_{j-1}) > H_0/2 \geq T$ by [Lemma 6.6](#), or the returned value satisfied $B_{j-1} > T$, and $c(R_{j-1}) \geq B_{j-1}$. We claim that

$$2^{1-j}c(\mathcal{S}) > (1 - \xi)T.$$

Indeed, if $2^{1-j}c(\mathcal{S}) \leq (1 - \xi)T$, then [Eq. \(6.5\)](#) would give $c(R_{j-1}) \leq 2^{1-j}c(\mathcal{S}) + \xi T \leq T$, a contradiction. Thus, the accepted round satisfies $T < 2^{1-j}c(\mathcal{S})/(1 - \xi)$, so the deviation permitted by [Eq. \(6.5\)](#) is $\xi \max\{2^{-j}c(\mathcal{S}), T\} \leq \frac{2\xi}{1-\xi}2^{-j}c(\mathcal{S})$, and

$$\frac{1-3\xi}{1-\xi}2^{-j}c(\mathcal{S}) \leq c(R_j) \leq \frac{1+\xi}{1-\xi}2^{-j}c(\mathcal{S}). \quad (6.11)$$

If $j = 0$, then $R_0 = \mathcal{S}$, so [Eq. \(6.11\)](#) holds trivially. By [Lemma 6.6](#), the evaluator gives $(1 - 3\xi)c(R_j) \leq B_j \leq c(R_j)$. Multiplying through by $2^j(1 - \xi)/(1 + \xi)$, the returned value obeys

$$A \leq c(\mathcal{S}) \quad \text{and} \quad \frac{c(\mathcal{S})}{A} \leq \frac{1+\xi}{(1-3\xi)^2}.$$

For $0 < \xi \leq 1/20$,

$$\frac{1+\xi}{(1-3\xi)^2} - 1 = \frac{7\xi - 9\xi^2}{(1-3\xi)^2} \leq 10\xi \leq \varepsilon.$$

This proves the approximation guarantee.

It remains to bound the running time. By [Eq. \(6.4\)](#), $h = O_q(\log(n+2))$, so the cap is $H_0 = \lceil 2T \rceil = O_{\varepsilon,q}(\log(n+2))$. By [Lemma 6.6](#), the evaluator of round j thus takes $O_{\varepsilon,q}(m_j \log^5(n+2))$ time, where m_j is the size of its sample,

plus the $O(n)$ time needed to draw the sample. There are $O(\log(n + 2))$ rounds, which gives the claimed worst-case bound. For the expectation, conceptually draw all rounds, including those after stopping. Their expected total size is

$$\sum_{j=0}^J \mathbb{E}[m_j] = n \sum_{j=0}^J 2^{-j} < 2n.$$

Stopping only reduces the work, so the expected time is $O_{\varepsilon,q}(n \log^5(n + 2))$. ■

7. Conclusions

We provided a near-linear time algorithm that computes a constant factor approximation to the congestion of a polygonal curve (i.e., the minimum c such that the curve is c -packed). We consider the result to be quite surprising, even though the constant is undesirably large (i.e., 42). We then showed, in [Section 6](#), using AI to develop this part, that the approximation factor can be improved to $1 + \varepsilon$, for any fixed $\varepsilon > 0$, while keeping the running time near-linear.

The constant factor algorithm of [Section 5](#) works verbatim in any constant dimension – it has not used planarity in any way. The quality of approximation deteriorates with the dimension d , but it is still a constant when d is a constant. The running time remains $O(n \log^3 n)$. Extending the $(1 + \varepsilon)$ -approximation of [Section 6](#) to higher dimensions is left for future work. Another important property of our algorithms is that they do not require the input segments to form a curve.

The dependence on ε , and the number of logarithmic factors, in the running time of the algorithm of [Section 6](#) are likely far from optimal. Improving them is an interesting problem for future research.

Nothing in our algorithm is a no-starter from a practical point of view. Performing an experimental study using this algorithm is an interesting direction for future research.

Acknowledgements. Work on this paper started more than a decade ago in a collaboration between the first author and Anne Driemel. The authors thank the anonymous referees for their detailed and insightful comments that significantly improved this work.

References

- [AH08] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
- [AKGM20] Sepideh Aghamolaei, Vahideh Keikha, Mohammad Ghodsi, and Ali Mohades. Windowing queries using Minkowski sum and their extension to mapreduce. *J. Supercomput.*, April 2020.
- [Alt09] Helmut Alt. *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, chapter The Computational Geometry of Comparing Shapes, page 235–248. Springer-Verlag, 2009.
- [BHR⁺20] Paul Beame, Sariel Har-Peled, Sivaramakrishnan Natarajan Ramamoorthy, Cyrus Rashtchian, and Makrand Sinha. Edge estimation with independent set oracles. *ACM Trans. Algo.*, 16(4), September 2020.
- [Bri14] K. Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 661–670, 2014.
- [DHW12] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Disc. Comput. Geom.*, 48:94–127, 2012.
- [Eri95] Jeff Erickson. On the relative complexities of some geometric problems. *Proc. 7th Canadian Conference on Computational Geometry*, page 85–90, 1995.
- [GSW20] Joachim Gudmundsson, Yuan Sha, and Sampson Wong. Approximating the packedness of polygonal curves. *CoRR*, abs/2009.07789, 2020. to appear in ISAAC 2020.
- [GvKS13] Joachim Gudmundsson, Marc van Kreveld, and Frank Staals. Algorithms for hotspot computation on trajectory data. In *Proc. 21st ACM SIG. Int. Conf. Adv. Geo. Info. Sys. (SIGSPATIAL)*, SIGSPATIAL’13, page 134–143, New York, NY, USA, 2013. Association for Computing Machinery.

- [Har11] S. Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Math. Surveys & Monographs*. Amer. Math. Soc., Boston, MA, USA, 2011.
- [Vig14] A. Vigneron. Geometric optimization and sums of algebraic functions. *ACM Trans. Algo.*, 10(1):4:1–4:20, 2014.
- [Wil15] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In Thore Husfeldt and Iyad A. Kanj, editors, *10th Int. Symp. Param. Exact Comput, IPEC*, volume 43 of *LIPICs*, pages 17–29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

A. Bernstein’s inequality

Theorem A.1 (Bernstein’s inequality). *Let $X_1, \dots, X_m$ be independent real random variables with $\mathbb{E}[X_i] = 0$ and $|X_i| \leq L$ almost surely, where $L > 0$. Set $X = \sum_{i=1}^m X_i$ and $\sigma^2 = \sum_{i=1}^m \mathbb{V}[X_i]$. Then, for every $t > 0$,*

$$\mathbb{P}[|X| \geq t] \leq 2 \exp \left(-\frac{t^2}{2\sigma^2 + (2/3)Lt} \right).$$