Covering a Set of Line Segments with a Few Squares

Joachim Gudmundsson[1], Mees van de Kerkhof[2], André van Renssen[1], Frank Staals[2], Lionov Wiratma[3], and Sampson Wong[1(✉)]

[1] University of Sydney, Sydney, Australia
{joachim.gudmundsson,andre.vanrenssen}@sydney.edu.au, swon7907@uni.sydney.edu.au

[2] Utrecht University, Utrecht, Netherlands
{m.a.vandekerkhof,f.staals}@uu.nl

[3] Parahyangan Catholic University, Bandung, Indonesia
lionov@unpar.ac.id

Abstract. We study three covering problems in the plane. Our original motivation for these problems come from trajectory analysis. The first is to decide whether a given set of line segments can be covered by up to four unit-sized, axis-parallel squares. The second is to build a data structure on a trajectory to efficiently answer whether any query subtrajectory is coverable by up to three unit-sized axis-parallel squares. The third problem is to compute a longest subtrajectory of a given trajectory that can be covered by up to two unit-sized axis-parallel squares.

Keywords: Computational geometry · Geometric coverings · Trajectory analysis

1 Introduction

Geometric covering problems are a classic area of research in computational geometry. The traditional *geometric set cover problem* is to decide whether one can place k axis-parallel unit-sized squares (or disks) to cover n given points in the plane. If k is part of the input, the problem is known to be NP-hard [5,11]. Thus, efficient algorithms are known only for small values of k. For $k = 2$ or 3, there are linear time algorithms [4,17], and for $k = 4$ or 5, there are $O(n \log n)$ time algorithms [12,15]. For general k, the $O(n^{\sqrt{k}})$ time algorithm for unit-sized disks [10] most likely generalises to unit-sized axis-parallel squares [1].

Motivated by trajectory analysis, we study a line segment variant of the geometric set cover problem where the input is a set of n line segments. Given a set of line segments, we say it is *k-coverable* if there exist k unit-sized axis-parallel squares in the plane so that every line segment is in the union of the k squares (we may write coverable to mean k-coverable when k is clear from the context). The first problem we study in this paper is:

The full version of this paper can be found at [7].

Problem 1. Decide if a set of line segments is k-coverable, for $k \in O(1)$.

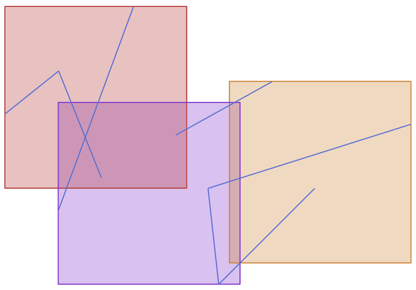
Fig. 1. A set of 3-coverable segments.

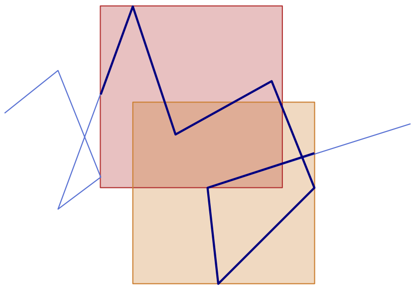
Fig. 2. A 2-coverable subtrajectory.

A key difference in the line segment variant and the point variant is that each segment need not be covered by a single square, as long as each segment is covered by the union of the k squares. See Fig. 1.

Hoffmann [9] provides a linear time algorithm for $k = 2$ and 3, however, a proof was not included in his extended abstract. Sadhu et al. [14] provide a linear time algorithm for $k = 2$ using constant space. In Sect. 2, we provide a proof for a $k = 3$ algorithm and a new $O(n \log n)$ time algorithm for $k = 4$.

Next, we study trajectory coverings. A trajectory $\mathcal{T}$ is a polygonal curve in the plane parametrised by time. A subtrajectory $\mathcal{T}[s,t]$ is the trajectory $\mathcal{T}$ restricted to a contiguous time interval $[s,t] \subseteq [0,1]$, see Fig. 2 for an example. Trajectories are commonly used to model the movement of an object (e.g. a bird, a vehicle, etc.) through time and space. The analysis of trajectories have applications in animal ecology [3], meteorology [18], and sports analytics [6].

To the best of our knowledge, this paper is the first to study k-coverable trajectories for $k \geq 2$. A k-coverable trajectory may, for example, model a commonly travelled route, and the squares could model a method of displaying the route (i.e. over multiple pages, or multiple screens), or alternatively, the location of several facilities. We build a data structure that can efficiently decide whether a subtrajectory is k-coverable.

Problem 2. Construct a data structure on a trajectory, so that given any query subtrajectory, it can efficiently answer whether the subtrajectory is k-coverable, for $k \in O(1)$.

For $k = 2$ and $k = 3$ we preprocess a trajectory $\mathcal{T}$ with n vertices in $O(n \log n)$ time, and store it in a data structure of size $O(n \log n)$, so that we can test if an arbitrary subtrajectory (not necessarily restricted to vertices) $\mathcal{T}[s,t]$ can be k-covered.

Finally, we consider a natural extension of Problem 2, that is, to calculate the *longest* k-coverable subtrajectory of any given trajectory. This problem is similar in spirit to the problem of covering the maximum number of points by k unit-sized axis-parallel squares [2,13].

Problem 3. Given a trajectory, compute its longest k-coverable subtrajectory, for $k \in O(1)$.

Problem 3 is closely related to computing a trajectory *hotspot*, which is a small region where a moving object spends a large amount of time. For $k = 1$ squares, the existing algorithm by Gudmundsson et al. [8] computes longest 1-coverable subtrajectory of any given trajectory. We notice a missing case in their algorithm, and show how to resolve this issue in the same running time of $O(n \log n)$. Finally, we show how to compute the longest 2-coverable subtrajectory of any given trajectory in $O(n\beta_4(n) \log^2 n)$ time. Here, $\beta_4(n) = \lambda_4(n)/n$, and $\lambda_s(n)$ is the length of a Davenport-Schinzel sequence of order s on n symbols. Omitted proofs can be found in the full version [7].

2 Problem 1: The Decision Problem

2.1 Is a Set of Line Segments 2-Coverable?

This section restates known results that will be useful for the recursive step in Sect. 2.2 and for the data structure in Sect. 3.1. The first result relates the bounding box, which is the smallest axis-aligned rectangle that contains all the segments, to a covering, which is a set of squares that covers all line segments.

Observation 1. *Every covering must touch all four sides of the bounding box.*

The reasoning behind Observation 1 is simple: if the covering does not touch one of the four sides, say the left side, then the covering could not have covered the leftmost vertex of the set of segments. An intuitive way for two squares to satisfy Observation 1 is to place the two squares in opposite corners of the bounding box. This intuition is formalised in Observation 1.

Lemma 1 (Sadhu et al. [14]). *A set of segments is 2-coverable if and only if there is a covering with squares in opposite corners of the bounding box of the set of segments.*

Lemma 1 is useful in that it narrows down our search for a 2-covering. It suffices to check the two configurations where squares are in opposite corners of the bounding box. For each of these two configurations, we simply check if each segment is in the union of the two squares, which takes linear time in total, leading to the following theorem:

Theorem 1. *One can decide if a set of n segments is 2-coverable in $O(n)$ time.*

2.2 Is a Set of Line Segments 3-Coverable?

We notice that for a covering consisting of three squares, Lemma 1 and the pigeon-hole principle imply that there must be one square that touches at least two sides of the bounding box. An intuitive way to achieve this is if one of the squares in the 3-covering is in a corner of the bounding box. We formalise this intuition in Lemma 2.

Lemma 2. *A set of segments is 3-coverable if and only if there is a covering with a square in a corner of the bounding box of the set of segments.*

Again, this lemma allows us to narrow down our search for a 3-covering. We consider four cases, one for each corner of the bounding box. After placing the first square in one of the four corners, we would like to check whether two additional squares can be placed to cover the remaining segments. We start by computing the remaining segments that are not yet covered. We subdivide each segment into at most one subsegment that is covered by the corner square, and up to two subsegments that are not yet covered. Then we can use Theorem 1 to (recursively) check whether two additional squares can be placed to cover all the uncovered subsegments.

The running time for subdividing each segment takes linear time in total. There are at most a linear number of remaining segments. Checking if the remaining segments are 2-coverable takes linear time by Theorem 1. Hence, we have the following theorem:

Theorem 2. *One can decide if a set of n segments is 3-coverable in $O(n)$ time.*

2.3 Is a Set of Line Segments 4-Coverable?

For a 4-covering, it remains true that any covering must touch all four sides of the bounding box. Unlike the three squares case, we cannot use the pigeon-hole principle to deduce that there is a square touching at least two sides of the bounding box. Fortunately, we have only two cases: either there exists a square which touches at least two sides of the bounding box, or each square touches exactly one side of the bounding box. This implies:

Lemma 3. *A set of segments is 4-coverable if and only if: (i) there is a covering with a square in a corner of the bounding box, or (ii) there is a covering with each square touching exactly one side of the bounding box.*

In the first case we can use the same strategy as in the three squares case by placing the first square in a corner and then (recursively) checking if three additional squares can cover the remaining subsegments. This gives a linear time algorithm for the first case.

For the remainder of this section, we focus on solving the second case.

Definition 1. *Define L, B, T and R to be the square that touches the left, bottom, top and right sides of the bounding box respectively. See Fig. 3.*

Without loss of generality, suppose that T is to the left of B. This implies that the left to right order of the squares is L, T, B, R. Suppose for now there were a way to compute the initial placement of L. Then we can deduce the position of T in the following way.

Lemma 4. *Given the position of L, if three additional squares can be placed to cover the remaining subsegments, then it can be done with T in the top-left corner of the bounding box of the remaining subsegments.*

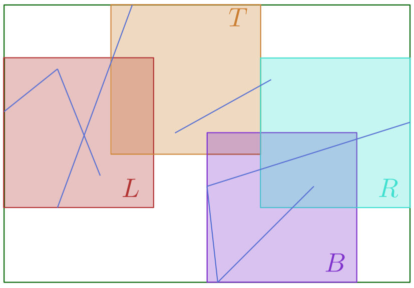

Fig. 3. The squares L, T, B and R.

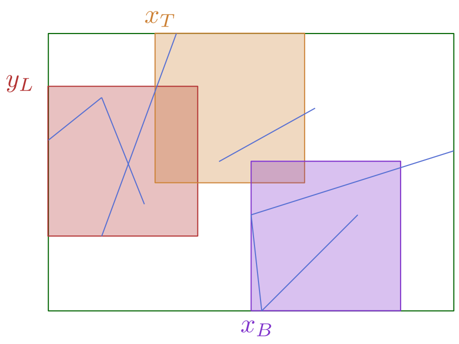

Fig. 4. The variables y_L, x_T and x_B.

The intuition behind this lemma is that after placing the first square, T is the topmost and leftmost of the remaining squares. A formal proof for Lemma 4 is given in the full version [7]. For an analogous reason, after placing the first two squares, we can place B in the bottom-left corner of the bounding box of the remaining segments. Finally, we cover the remaining segments with R, if possible.

We have therefore shown that the position of L along the left boundary uniquely determines the positions of the squares T, B and R along their respective boundaries. Unfortunately, we do not know the position of L in advance, so instead we consider all possible initial positions of L via parametrisation. Let y_L be the y-coordinate of the top side of L, and similarly let x_T, x_B be the x-coordinates of the left side of T and B respectively. See Fig. 4.

Finally, we will try to cover all remaining subsegments with the square R. Define x_{R_1} and x_{R_2} to be the x-coordinates of the leftmost and rightmost uncovered points after the first three squares have been placed. Similarly, define y_{R_1} and y_{R_2} to be the y-coordinates of the topmost and bottommost uncovered points. Then it is possible to cover the remaining segments with R if and only if $x_{R_1} - x_{R_2} \leq 1$ and $y_{R_1} - y_{R_2} \leq 1$.

Since the position of L uniquely determines T, B and R, we can deduce that the variables x_T, x_B, x_{R_1}, x_{R_2}, y_{R_1} and y_{R_2} are all functions of y_L. We will show that each of these functions is piecewise linear and can be computed in $O(n \log n)$ time. We begin by computing x_T as a function of variable y_L.

Lemma 5. *The variable x_T as a function of variable y_L is a piecewise linear function and can be computed in $O(n \log n)$ time.*

Next, we show that x_B is a piecewise linear function of y_L, with complexity $O(n)$, and can be computed in $O(n \log n)$ time.

Lemma 6. *The variable x_B as a function of variable y_L is a piecewise linear function and can be computed in $O(n \log n)$ time.*

Then we compute x_{R_1}, x_{R_2}, y_{R_1} and y_{R_2} in a similar fashion.

Lemma 7. *The variables x_{R_1}, x_{R_2}, y_{R_1}, y_{R_2} as functions of variable y_L are piecewise linear functions and can be computed in $O(n \log n)$ time.*

Finally, we check if there exists a value of y_L so that $x_{R_1} - x_{R_2} \leq 1$ and $y_{R_1} - y_{R_2} \leq 1$. If so, there exist positions for L, B, T and R that covers all the segments, otherwise, there is no such position for L, T, B and R. This yields the following result:

Theorem 3. *One can decide if a set of n segments is 4-coverable in $O(n \log n)$ time.*

3 Problem 2: The Subtrajectory Data Structure Problem

In this section, we briefly describe some of the main ideas for building the data structures that can answer whether a subtrajectory is either 2-coverable or 3-coverable. Details of the data structures can be found in the full version of this paper [7].

We begin by building three preliminary data structures. Given a piecewise linear trajectory of complexity n, our preliminary data structures are:

Tool 1. *A bounding box data structure that preprocesses a trajectory in $O(n)$ time, so that given a query subtrajectory, it returns the subtrajectory's bounding box in $O(\log n)$ time.*

Tool 2. *An upper envelope data structure that preprocesses a trajectory in $O(n \log n)$ time, so that given a query subtrajectory and a query vertical line, it returns the highest intersection between the subtrajectory and the vertical line (if one exists) in $O(\log n)$ time. See Fig. 5.*

Tool 3. *A highest vertex data structure that preprocesses a trajectory in $O(n \log n)$ time, so that given a query subtrajectory and a query axis-parallel rectangle, it returns the highest vertex of the subtrajectory inside the rectangle (if one exists) in $O(\log^2 n)$ time. See Fig. 6.*

3.1 Query If a Subtrajectory Is 2-Coverable

Our construction procedure is to build Tool 1 and Tool 2. Our query procedure consists of two steps. The first step is to narrow down the covering to one of two configurations using Lemma 1 and Tool 1. The second step is to check whether one of these configurations indeed covers the subtrajectory. The key idea in the second step is to use Tool 2 along the boundary of the configuration to see if the subtrajectory passes through the boundary. Putting this together yields:

Theorem 4. *Let $\mathcal{T}$ be a trajectory with n vertices. After $O(n \log n)$ preprocessing time, $\mathcal{T}$ can be stored using $O(n \log n)$ space, so that deciding if a query subtrajectory $\mathcal{T}[a, b]$ is 2-coverable takes $O(\log n)$ time.*

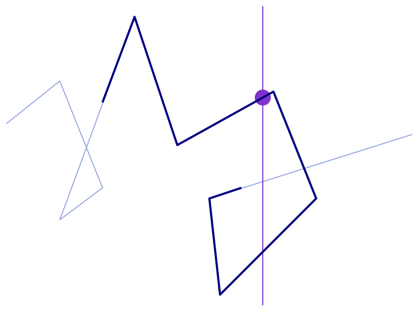

Fig. 5. Tool 2 returns the highest intersection of a subtraj. and a vertical line.

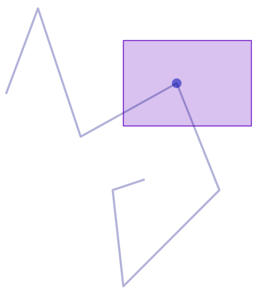

Fig. 6. Tool 3 returns the highest subtrajectory vertex in a query rectangle.

3.2 Query If a Subtrajectory Is 3-Coverable

Our construction procedure is to build Tools 1, 2, and 3. Our query procedure consists of three steps. The first step is to place the first square in a constant number of configurations using Lemma 2 and Tool 1. For each placement of the first square, the second step generates two configurations by placing the remaining two squares. The key idea in the second step is to compute the bounding box of the uncovered subsegments by using a combination of Tools 2 and 3. The third step is to check if a configuration indeed covers the subtrajectory. The key idea in the third step is to use Tool 2 along the boundary of the configuration to see if the subtrajectory passes through the boundary. We require an additional check using Tool 3 in one of the configurations. Putting this together yields:

Theorem 5. *Let $\mathcal{T}$ be a trajectory with n vertices. After $O(n \log n)$ preprocessing time, $\mathcal{T}$ can be stored using $O(n \log n)$ space, so that deciding if a query subtrajectory $\mathcal{T}[a,b]$ is 3-coverable takes $O(\log^2 n)$ time.*

4 Problem 3: The Longest Coverable Subtrajectory

In this section we compute a longest k-coverable subtrajectory $\mathcal{T}[p^*, q^*]$ of a given trajectory $\mathcal{T}$. Note that the start and end points p^* and q^* of such a subtrajectory need not be vertices of the original trajectory. Gudmundsson, van Kreveld, and Staals [8] presented an $O(n \log n)$ time algorithm for the case $k = 1$. However, we note that there is a mistake in one of their proofs, and hence their algorithm misses one of the possible scenarios. We correct this mistake, and using the insight gained, also solve the problem for $k = 2$.

4.1 A Longest 1-Coverable Subtrajectory

Gudmundsson, van Kreveld, and Staals state that there exists an optimal placement of a unit square, i.e. one such that the square covers a longest 1-coverable subtrajectory of $\mathcal{T}$, and has a vertex of $\mathcal{T}$ on its boundary [8, Lemma 7]. However, that is incorrect, as illustrated in Fig. 7. Let $p(t)$ be a parametrisation of the trajectory. Fix a corner c of the square and shift the square so that c follows $p(t)$. Let $q(t)$ be the point so that $\mathcal{T}[p(t), q(t)]$ is the maximal subtrajectory contained in the square, and let $\phi(t)$ be the length of this subtrajectory. This function ϕ is piecewise linear, with inflection points not only when a vertex of $\mathcal{T}$ lies on the boundary of the square, but also when $p(t)$ or $q(t)$ hits a corner of the square. The argument in [8] misses this last case. Instead, the correct characterization is:

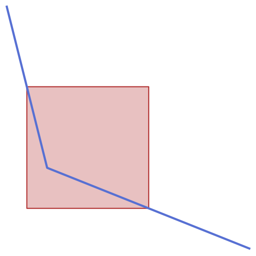
Fig. 7. An optimal placement that has no vertex on the boundary of the square.

Lemma 8. *Given a trajectory $\mathcal{T}$ with vertices $v_1, .., v_n$, there exists a square H covering a longest 1-coverable subtrajectory so that either:*

- *there is a vertex v_i of $\mathcal{T}$ on the boundary of H, or*
- *there are two trajectory edges passing through opposite corners of H.*

We give the full proof of this lemma in the full version of this paper [7]. To compute a longest 1-coverable subtrajectory we also have to consider this scenario. We use the existing algorithm to test all placements of the first type from Lemma 8 in $O(n \log n)$ time. Next, we briefly describe how we can also test all placements of the second type in $O(n \log n)$ time.

Lemma 9. *Given a pair of non-parallel edges e_i and e_j of $\mathcal{T}$, there is at most one unit square H such that the top left corner of H lies on e_i, and the bottom right corner of H lies on e_j.*

It follows that any pair of edges e_i, e_j of $\mathcal{T}$ generates at most a constant number of additional candidate placements that we have to consider. Let $\mathcal{H}_{ij}$ denote this set. Next, we argue that there are only $O(n)$ relevant pairs of edges that we have to consider.

We define the *reach* of a vertex v_i, denoted $r(v_i)$, as the vertex v_j such that $\mathcal{T}[v_i, v_j]$ can be 1-covered, but $\mathcal{T}[v_i, v_{j+1}]$ cannot. Let $\mathcal{H}_i = \mathcal{H}_{(i-1)j}$ denote the set of candidate placements corresponding to v_i and $v_j = r(v_i)$. Analogously, we define the *reverse reach* $rr(v_j)$ of v_j as the vertex v_i such that $\mathcal{T}[v_i, v_j]$ can be 1-covered, but $\mathcal{T}[v_{i-1}, v_j]$ cannot, and the set $\mathcal{H}'_j = \mathcal{H}_{(i-1)j}$. Finally, let $\mathcal{H} = \bigcup_{i=1}^{n} \mathcal{H}_i \cup \mathcal{H}'_i$ be the set of placements contributed by all reach and reverse reach pairs. Observe that this set consists of $O(n)$ placements, as all individual sets $\mathcal{H}_i$ and $\mathcal{H}'_i$ have at most one element.

Lemma 10. *Let $p^* \in e_i$ and $q^* \in e_j$ lie on edges of $\mathcal{T}$, and let H be a unit square with p^* in one corner, and q^* in the opposite corner. We have that $H \in \mathcal{H}$.*

Once we have the reach $r(v_i)$ and the reverse reach $rr(v_i)$ for every vertex v_i we can easily construct $\mathcal{H}$ in linear time (given a pair of edges e_i, e_j we can construct the unit squares for which one corner lies on e_i and the opposite corner lies on e_j in constant time). We can use Tool 1 to test each candidate in $O(\log n)$ time. So all that remains is to compute the reach of every vertex of $\mathcal{T}$; computing the reverse reach is analogous.

Lemma 11. *We can compute $r(v_i)$, for each vertex $v_i \in \mathcal{T}$, in $O(n \log n)$ time in total.*

Theorem 6. *Given a trajectory $\mathcal{T}$ with n vertices, there is an $O(n \log n)$ time algorithm to compute a longest 1-coverable subtrajectory of $\mathcal{T}$.*

4.2 A Longest 2-Coverable Subtrajectory

In this section we reuse some of the observations from Sect. 4.1 to develop an $O(n\beta_4(n)\log^2 n)$ time algorithm for the $k = 2$ case. Here, $\beta_4(n) = \lambda_4(n)/n$, and $\lambda_s(n)$ is the length of a Davenport-Schinzel sequence of order s on n symbols.

Our algorithm to compute a longest 2-coverable subtrajectory $\mathcal{T}[p^*, q^*]$ of $\mathcal{T}$ consists of two steps. In the first step we compute a set S of candidate starting points on $\mathcal{T}$, so that $p^* \in S$. In the second step, we compute the longest 2-coverable subtrajectory $\mathcal{T}[p, q]$ for each starting point $p \in S$, and report a longest such subtrajectory. With slight abuse/reuse of notation, for any point $p \in S$, we denote the endpoint q of this longest 2-coverable subtrajectory $\mathcal{T}[p, q]$ by $r(p)$. This generalizes our notion of *reach* from Sect. 4.1 to arbitrary points on $\mathcal{T}$.

Computing the Reach of a Point. We modify the data structure in Theorem 4, i.e. the data structure for answering whether a given subtrajectory is 2-coverable, to answer the reach queries. We do so by applying parametric search to the query procedure. Note that applying a simple binary search will give us only the edge containing $r(p)$. Furthermore, even given this edge it is unclear how to find $r(p)$ itself, as the squares may still shift, depending on the exact position of $r(p)$.

Lemma 12. *Let $\mathcal{T}$ be a trajectory with n vertices. After $O(n \log n)$ preprocessing time, $\mathcal{T}$ can be stored using $O(n \log n)$ space, so that given a query point p on $\mathcal{T}$ it can compute the reach $r(p)$ of p in $O(\log^2 n)$ time.*

Corollary 1. *Given a trajectory $\mathcal{T}$, and a set of m candidate starting points on $\mathcal{T}$, we can compute the longest 2-coverable subtrajectory that starts at one of those points in $O(n \log n + m \log^2 n)$ time.*

Computing the Set of Starting Points. It remains only to construct a set S of candidate starting points with the property that the starting point of a longest 2-coverable subtrajectory is guaranteed to be in the set. Our construction consists of six types of starting points, which when grouped up into their respective types, we will call *events*. The six types of events are vertex events, reach events, bounding box events, bridge events, upper envelope events, and special configuration events. Figures 8, 9, and 10 illustrate these events, and show how a longest 2-coverable subtrajectory may start at such an event. We then prove that it suffices to consider *only* these six types of candidate starting points. Finally, we bound the number of events, and thus candidate starting points, and describe how to compute them. Combining this with our result from Corollary 1 gives us an efficient algorithm to compute a longest 2-coverable subtrajectory. Note that in Definitions 2–7, for simplicity we define the events only in one of the four cardinal directions. However, in our construction in Definition 8 we require all six events for all four cardinal directions.

Definition 2. *Given a trajectory $\mathcal{T}$, p is a vertex event if p is a vertex of $\mathcal{T}$.*

Definition 3. *Given a trajectory $\mathcal{T}$, p is a reach event if $r(p)$ is a vertex of $\mathcal{T}$, and no point $q < p$ satisfies $r(q) = r(p)$.*

Definition 4. *Given a trajectory $\mathcal{T}$, p is a bounding box event if the topmost vertex of $\mathcal{T}$ within the subtrajectory $\mathcal{T}[p, r(p)]$ has the same y-coordinate as p.*

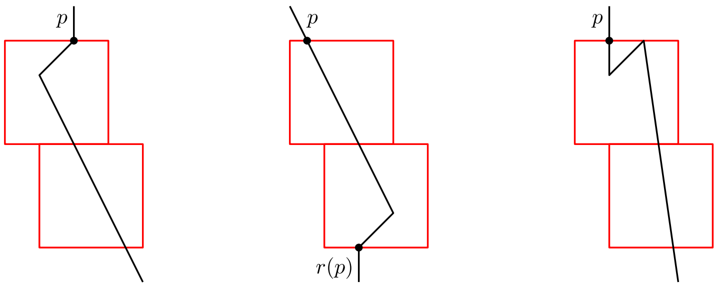

Fig. 8. A vertex event (left), a reach event (middle), and a bounding box event (right).

Definition 5. *Given a trajectory $\mathcal{T}$, p is a bridge event if:*

- *the point p is the leftmost point on $\mathcal{T}[p, r(p)]$, and*
- *the point p is one unit to the left of a point $u \in \mathcal{T}[p, r(p)]$, and*
- *the point u is one unit above the lowest vertex of $\mathcal{T}[p, r(p)]$.*

Definition 6. *Given a trajectory $\mathcal{T}$, p is an upper envelope event if:*

- *the point p is the leftmost point on $\mathcal{T}[p, r(p)]$, and*
- *the point p is one unit to the left of a point $u \in \mathcal{T}[p, r(p)]$, and*
- *the point u is an intersection or vertex on the upper envelope of $\mathcal{T}[p, r(p)]$.*

Definition 7. *Given a trajectory $\mathcal{T}$, p is a special configuration event if there is a covering of squares H_1 and H_2 so that H_1 contains the top-left corner of H_2, and either:*

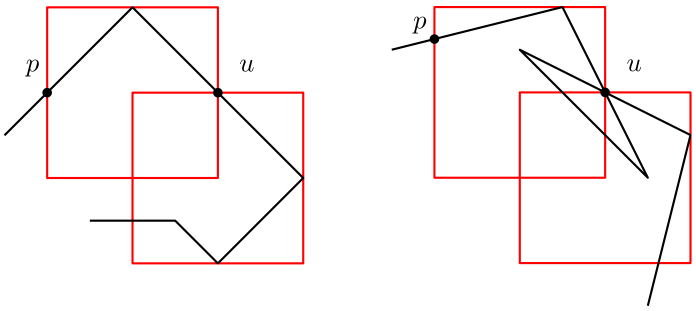

Fig. 9. Examples of a bridge event (left), and an upper envelope event box (right).

1. *point p is in the top-right corner of H_1 and $r(p)$ is in the bottom-left corner of H_1, or*
2. *point p is in the top-left corner of H_1 and the trajectory $\mathcal{T}$ passes through the bottom-right corner of H_1, or*
3. *point p is in the top-left corner of H_1, $r(p)$ is in the bottom-right corner of H_2, and the trajectory $\mathcal{T}$ passes through the two intersections of H_1 and H_2.*

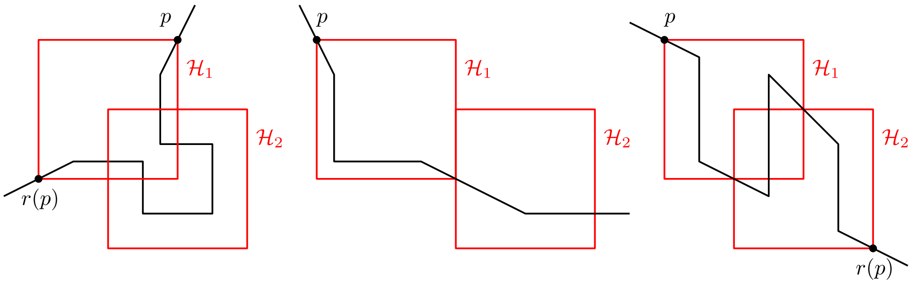

Fig. 10. Examples of the three types of special configuration events.

Using these definitions and their analogous versions in all four cardinal directions, we subdivide the trajectory $\mathcal{T}$ to obtain a trajectory $\mathcal{T}_3$ that forms our set of candidate starting points.

Definition 8. *Given a trajectory $\mathcal{T}$, let $\mathcal{T}_1$ be a copy of $\mathcal{T}$ with these additional points added to the set of vertices of $\mathcal{T}_1$:*

– *all the vertex, reach, bounding box, and bridge events of $\mathcal{T}$ for all four cardinal directions.*

Next, let $\mathcal{T}_2$ be a copy of $\mathcal{T}_1$ with these additional points added to the set of vertices of $\mathcal{T}_2$:

– *all the upper envelope events of $\mathcal{T}_1$ for all four cardinal directions.*

Finally, let $\mathcal{T}_3$ be a copy of $\mathcal{T}_2$ with these additional points added to the set of vertices of $\mathcal{T}_3$:

– *all the special configuration events of $\mathcal{T}_2$ for all configurations of $\mathcal{H}_1$ and $\mathcal{H}_2$.*

Next, we argue that the set of vertices of this trajectory $\mathcal{T}_3$ is a suitable set of candidate starting points.

Lemma 13. *The set $\mathcal{T}_3$ is guaranteed to contain the starting point of a longest coverable subtrajectory of $\mathcal{T}$.*

We now bound the number of candidate starting points.

Lemma 14. *Trajectory $\mathcal{T}_3$ has $O(n\beta_4(n))$ vertices, and can be constructed in time $O(n\beta_4(n)\log^2 n)$. More specifically, for each type of event, the number of such events and the time in which we can compute them is*

	#events	computation time
Vertex events	$O(n)$	$O(n)$
Reach events	$O(n)$	$O(n\log^2 n)$
Bounding box events	$O(n)$	$O(n\log^2 n)$
Bridge events	$O(n)$	$O(n\log^2 n)$
Upper envelope events	$O(n\beta_4(n))$	$O(n\beta_3(n)\log^2 n)$
Special configuration events	$O(n\beta_4(n))$	$O(n\beta_4(n)\log^2 n)$

Proof. We briefly sketch the idea for only the upper envelope events. Refer to full version for the details, the proofs for the other events, and the description of the algorithms that compute these events [7].

Consider the set V of vertices of $\mathcal{T}$ and intersections of $\mathcal{T}$ with itself, and observe that every point $u \in V$ generates at most $O(1)$ candidate starting points p that are upper-envelope events. However, there may be $\Theta(n^2)$ such intersection points. We argue that not all of these intersection points generate valid starting points.

Let $p_1, .., p_k$ be the upper envelope events of the trajectory, and let $U = u_1, .., u_k$ be the edge of $\mathcal{T}$ that contains the point u on $\mathcal{T}[p, r(p)]$ one unit to the right of p. We argue that U is a Davenport-Schinzel sequence of order 4 on $n-1$ symbols [16], and thus has complexity $k = O(n\beta_4(n))$. It follows that there are $O(n\beta_4(n))$ upper envelope events.

Since U is a Davenport-Schinzel sequence of order $s = 4$, it has no alternating (not necessarily contiguous) subsequences of length $s+2 = 6$. Our first step is to show that if the sequence of edges a, b, a occurs (not necessarily consecutively) then the starting points corresponding to the first two segments of the sequence must be x-monotone. Hence, an alternating sequence of edges of length six yields alternating, x-monotone sequence of starting points of length five. We then argue that this leads to a contradiction. $\square$

By Lemma 14 we can compute $m = O(n\beta_4(n))$ candidate starting times for a longest 2-coverable subtrajectory of $\mathcal{T}$ in $O(n\beta_4(n)\log^2 n)$ time. Using Corollary 1 we can thus compute this subtrajectory in $O(n \log n + m \log^2 n) = O(n\beta_4(n)\log^2 n)$ time.

Theorem 7. *Given a trajectory $\mathcal{T}$ with n vertices, there is an $O(n\beta_4(n)\log^2 n)$ time algorithm to compute a longest 2-coverable subtrajectory of $\mathcal{T}$.*

References

1. Agarwal, P.K., Procopiuc, C.M.: Exact and approximation algorithms for clustering. Algorithmica **33**(2), 201–226 (2002)
2. Bereg, S., et al.: Optimizing squares covering a set of points. Theor. Comput. Sci. **729**, 68–83 (2018)
3. Damiani, M.L., Issa, H., Cagnacci, F.: Extracting stay regions with uncertain boundaries from GPS trajectories: a case study in animal ecology. In: Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 253–262 (2014)
4. Drezner, Z.: On the rectangular p-center problem. Naval Res. Logistics (NRL) **34**(2), 229–234 (1987)
5. Fowler, R.J., Paterson, M., Tanimoto, S.L.: Optimal packing and covering in the plane are NP-complete. Inf. Process. Lett. **12**(3), 133–137 (1981)
6. Gudmundsson, J., Horton, M.: Spatio-temporal analysis of team sports. ACM Comput. Surv. (CSUR) **50**(2), 22 (2017)
7. Gudmundsson, J., van de Kerkhof, M., Renssen, A., Staals, F., Wiratma, L., Wong, S.: Covering a set of line segments with a few squares. CoRR, abs/2101.09913 (2021)
8. 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)
9. Hoffmann, M.: Covering polygons with few rectangles. In: Abstracts 17th European Workshop Computational Geometry, pp. 39–42 (2001)
10. Hwang, R.Z., Lee, R.C.T., Chang, R.C.: The slab dividing approach to solve the Euclidean p-center problem. Algorithmica **9**(1), 1–22 (1993)
11. Megiddo, N., Supowit, K.J.: On the complexity of some common geometric location problems. SIAM J. Comput. **13**(1), 182–196 (1984)
12. Nussbaum, D.: Rectilinear p-piercing problems. In: Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, ISSAC, pp. 316–323 (1997)
13. Mahapatra, P.R.S., Goswami, P.P., Das, S.: Maximal covering by two isothetic unit squares. In: Canadian Conference on Computational Geometry, pp. 103–106 (2008)
14. Sadhu, S., Roy, S., Nandy, S.C., Roy, S.: Linear time algorithm to cover and hit a set of line segments optimally by two axis-parallel squares. Theor. Comput. Sci. **769**, 63–74 (2019)
15. Segal, M.: On piercing sets of axis-parallel rectangles and rings. Int. J. Comput. Geometry Appl. **9**(3), 219–234 (1999)
16. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and their Geometric Applications. Cambridge University Press, Cambridge (1995)

17. Sharir, M., Welzl, E.: Rectilinear and polygonal p-piercing and p-center problems. In: Proceedings of the 12th Annual Symposium on Computational Geometry, pp. 122–132 (1996)
18. Stohl, A.: Computation, accuracy and applications of trajectories–a review and bibliography. Dev. Environ. Sci. **1**, 615–654 (2002)