Covering a set of line segments with a few squares

Joachim Gudmundsson[1], **Mees van de Kerkhof**[2], **André van Renssen**[3], **Frank Staals**[4], **Lionov Wiratma**[5], **and Sampson Wong**[6]

1 **University of Sydney, Australia**
`joachim.gudmundsson@sydney.edu.au`
2 **Utrecht University, Netherlands**
`m.a.vandekerkhof@uu.nl`
3 **University of Sydney, Australia**
`andre.vanrenssen@sydney.edu.au`
4 **Utrecht University, Netherlands**
`f.staals@uu.nl`
5 **Utrecht University, Netherlands**
`l.wiratma@uu.nl`
6 **University of Sydney, Australia**
`swon7907@uni.sydney.edu.au`

Abstract

We study the problem of covering a set of line segments with a few (up to four) axis-parallel, unit-sized squares in the plane. Covering line segments with two squares has been previously studied, however, little is known for three or more squares. Our original motivation for the line segment covering problem comes from trajectory analysis. We study two trajectory covering problems: a data structure on the trajectory that efficiently answers whether a query subtrajectory is coverable, and an algorithm to compute its longest coverable subtrajectory.

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 to cover n points in the plane. If k is part of the input, the problem is known to be NP-hard [3, 7]. Thus, efficient algorithms are only known for small values of k. For $k = 2$ or 3, there are linear time algorithms [2, 11], and for $k = 4$ or 5, there are $O(n \log n)$ time algorithms [8, 10].

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, see Figure 1.

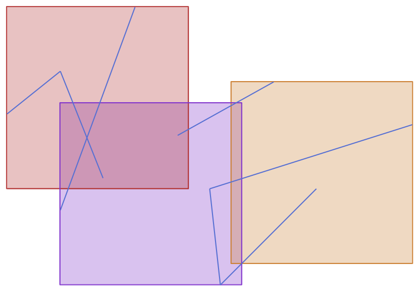

Figure 1 An example a set of n segments which is 3-coverable.

▶ Problem 1. Decide if a set of line segments is k-coverable, for $k = 2, 3, 4$.

36th European Workshop on Computational Geometry, Würzburg, Germany, March 16–18, 2020.
This is an extended abstract of a presentation given at EuroCG'20. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

A key difference in the line segment variant is that each segment need not be covered by a single square, as long as the k squares together cover all n segments. Sadhu et al. [9] recently provided a linear time algorithm for $k = 2$. Hoffman [6] provides a linear time algorithm for $k = 3$, however the proof was not included in the extended abstract. 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, which was the original motivation for studying line segment coverings. A trajectory is a polygonal curve in the plane parametrised by time, and models the movement of an object through time and space. Trajectory analysis has been used to study data sets in animal ecology [1], meteorology [12] and sports analytics [4]. Our trajectory analysis task is to compute a small region where a moving object spends a large amount of time. Such a region is known as a *hotspot* and has applications in segmentation, clustering, or enriching trajectory data with locations of interest [5].

If the hotspot is modelled by k unit-sized axis-parallel squares, then the subtrajectory that it covers is a k-coverable subtrajectory. Formally, a *subtrajectory* is the trajectory restricted to a contiguous time window. Note that the start and end points of the subtrajectory need not be vertices of the original trajectory, see Figure 2. The two problems we study are:

▶ 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 $k = 2, 3$.

▶ Problem 3. Given a trajectory, compute its longest k-coverable subtrajectory, for $k = 2$.

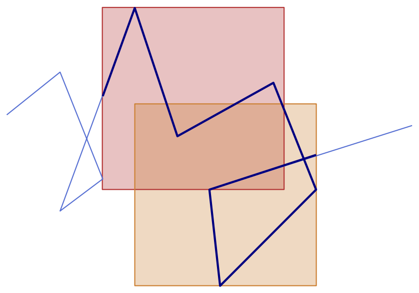

Figure 2 A trajectory (thin) and its longest 2-coverable subtrajectory (thick).

Previous work focuses on hotspots regions modelled by a single square [5]. To the best of our knowledge, this paper is the first to study k-coverable subtrajectories for $k \geq 2$.

The results of this paper and their relevant sections are summarised in Table 1.

	$k = 2$	$k = 3$	$k = 4$
Problem 1	Section 2.1	Section 2.2	Section 2.3
Problem 2	Section 3.1	Section 3.2	
Problem 3	Section 4.1		

Table 1 The results of this paper and their relevant sections.

2 Problem 1: The Decision Problem

2.1 Is a set of segments 2-coverable?

This section restates known results which are useful for the recursive step in Section 2.2 and for the data structure in Section 3.1. Recall that a bounding box of a set of segments is the smallest axis-aligned rectangle that contains all segments. Observation 1 uses the bounding box to decide if a set of segments is 2-coverable:

▶ **Observation 1.** *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.*

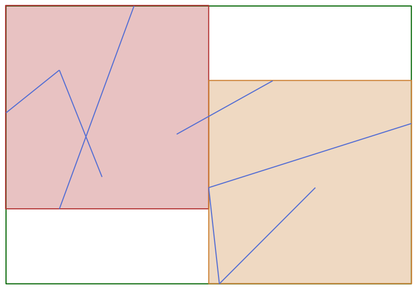

Figure 3 A covering with squares in opposite corners of the bounding box of the set of segments.

Observation 1 is due to Sadhu et al. [9]. See Figure 3 for an illustration. The algorithm is therefore to compute the bounding box, and for each pair of opposite corners, to check if squares placed in these corners cover every segment. This takes linear time in total. Thus:

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

2.2 Is a set of segments 3-coverable?

We start off with a similar observation to Observation 1, but for 3-coverable segments:

▶ **Observation 3.** *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.*

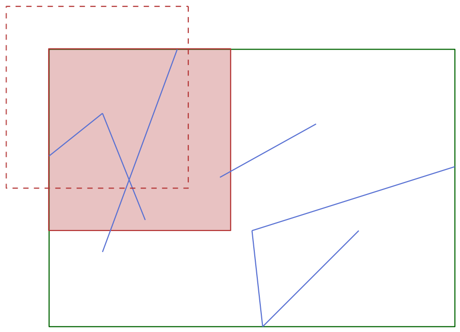

Figure 4 A partial covering with a square in a corner of the bounding box of the set of segments.

Proof. Any 3-covering touches all four sides of the bounding box, in fact, one square in the 3-covering must touch at least two sides. If the two sides are opposite then the bounding box has width less than or equal to one and there is a straightforward one-dimensional algorithm. Otherwise, without loss of generality let the adjacent sides be the top and left sides. If the square is not already in the topleft corner of the bounding box, move the square to the corner position shown in Figure 4. The new square covers more than the previous position, so the modified positions is 3-covering with a square in a corner. ◀

Now, we can use Observation 3 to place the first square into one of four possible positions. We compute up to two uncovered subsegments for each original segment. The set of uncovered subsegments has linear complexity. We then use Theorem 2 to recursively check if the uncovered subsegments are 2-coverable. All steps in this algorithm take linear time.

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

2.3 Is a set of segments 4-coverable?

For any 4-covering, each side of the bounding box is touched by one of the 4-covering squares. We make a similar observation to $k = 2$ and $k = 3$ but we have two cases: either we have a square which touches two sides, or each square touches exactly one side.

▶ **Observation 5.** *A set of segments is 4-coverable if and only if one of the following hold:*

- *There is a covering with a square in a corner of the bounding box.*
- *There is a covering with each square touching exactly one side of the bounding box.*

If the covering has a square in a corner of the bounding box, we can use the same strategy as in the $k = 3$ case and try all four positions of the first square and then recurse. Hence, for the remainder of this section, we assume we are in the second case.

▶ **Definition 6.** 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 Figure 5.

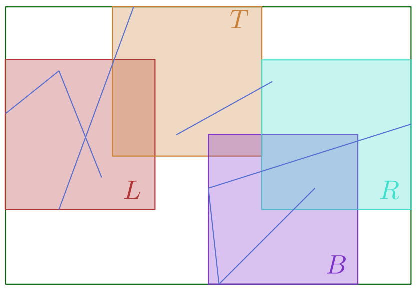

Figure 5 The squares L, T, B, R touch the left, top, bottom and right sides of the bounding box.

Without loss of generality, suppose that T is to the left of B. Then 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. The rest of the algorithm would be as follows. Compute the remaining uncovered subsegments, then place T in its topleft corner of its bounding box. We can do this since T is the leftmost and topmost of the remaining squares. Next, compute the remaining uncovered subsegments, then place B in the bottomright corner of its bounding box. Finally, cover the remaining uncovered subsegments with R, if possible.

Our approach is to simulate the above algorithm for variable positions of L. Let y_L be the y-coordinate of the top side of L, and let x_T be the x-coordinate of the left side of T.

▶ **Lemma 7.** *The variable x_T as a function of variable y_L is piecewise linear and can be computed in $O(n \log n)$ time.*

Proof. We know from our algorithm above that x_T is the leftmost uncovered point of the set of segments after placing L. We divide the uncovered region inside the bounding box into three separate regions: A above L, B below L and C to the right of L. See Figure 6. By definition, all these regions are uncovered. We consider the leftmost points of A, B and C separately, and then compute their minimum to obtain x_T.

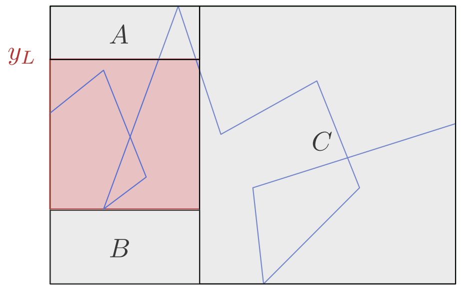

Figure 6 The region above, below, and to the right of L are A, B and C respectively.

As y_L increases, the leftmost point of A moves monotonically to the right. This polygonal curve is shown in Figure 7, and we call this curve the *skyline* of the set of segments. In the full version we show an $O(n \log n)$ divide and conquer algorithm to compute the skyline.

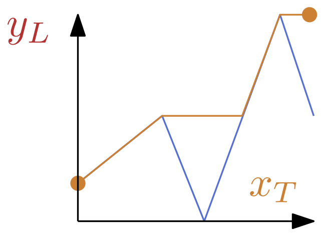

Figure 7 The leftmost point x_T as a function of y_L forming a "skyline".

A similar algorithm for the skyline gives the function for the leftmost point in B as a function of y_L. The leftmost point of C does not depend on y_L and can be computed in linear time. The value of x_T is the minimum value of the leftmost points of A, B and C. Since each is piecewise linear and can be computed in $O(n \log n)$ time, so is their minimum. ◀

We define x_B to be the leftmost uncovered point after placing L at y_L and T at x_T. In the full version of this paper, we show that x_B is a piecewise linear function of y_L and can be computed in $O(n \log n)$ time. Additional skylines need to be computed but a similar approach works. The same is also true for y_{R_1} and y_{R_2}, which are defined to be the topmost and bottommost points after placing L, T and B at y_L, x_T and x_B respectively. Finally, we check if there exists a y_L so that $y_{R_1} - y_{R_2}$ attains a value less than or equal to 1. If so, there is a position for L, T, B, R that covers the set of segments. This yields Theorem 8.

▶ **Theorem 8.** *One can decide whether a set of segments is 4-coverable in $O(n \log n)$ time.*

An open problem is to provide a polynomial time algorithm that decides if a set of segments is k-coverable for $k \geq 5$. Significantly new ideas are required as Observation 5 does not seem to extend to larger values of k.

3 Problem 2: The Subtrajectory Data Structure Problem

In the full version of this paper, we provide the details of the following data structures, i.e., their construction and query procedures and corresponding running times. Given a piecewise linear trajectory of complexity n, we build:

▶ **Tool 9.** *A bounding box data structure that preprocesses a trajectory in $O(n)$ time, so that given a query subtrajectory, returns its bounding box in $O(\log n)$ time.*

▶ **Tool 10.** *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, returns the highest intersection between the subtrajectory and the vertical line (if one exists) in $O(\log n)$ time.*

▶ **Tool 11.** *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, returns the highest vertex of the subtrajectory inside the rectangle (if one exists) in $O(\log n)$ time.*

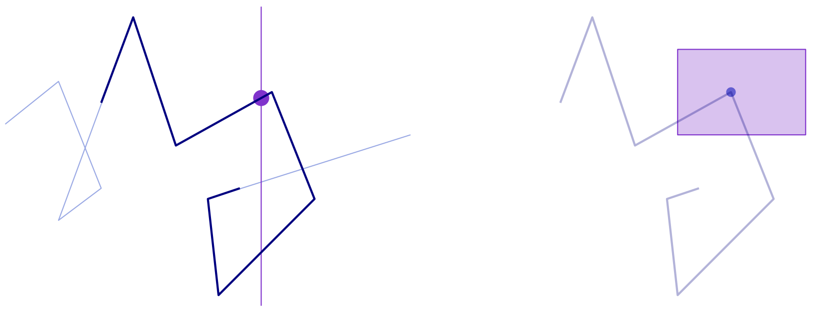

Figure 8 Tool 10 (left) returns the highest intersection between a vertical line and a subtrajectory. Tool 11 (right) returns the highest subtrajectory vertex in a query rectangle.

3.1 Is a query subtrajectory 2-coverable?

Our data structure is built like so. At preprocessing, we construct Tools 9 and 10 in $O(n \log n)$ time. At query time, we use Tool 9 to compute the bounding box of the subtrajectory in $O(\log n)$ time. By Observation 1, the subtrajectory is coverable only if there is a covering with squares in opposite corners of the bounding box. Consider the union of the two squares in opposite corners of the bounding box, as shown in the left diagram of Figure 9.

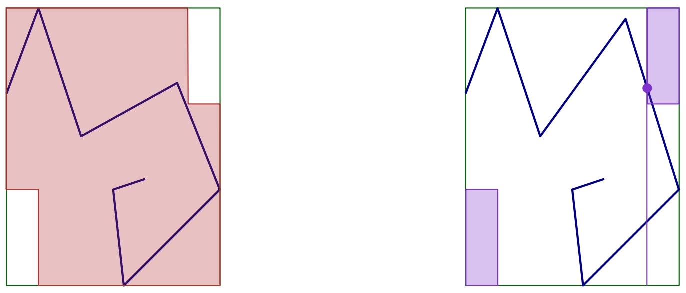

Figure 9 The union of the pair of squares (left) and its complement (right).

To check if this pair of squares covers the subtrajectory, we only need to check if its complement is empty. The complement is shaded in purple in Figure 9. We use Tool 10 to return the highest intersection of the subtrajectory with one of the sides of the purple rectangle, as shown in the same figure. This detects whether there is a subtrajectory edge which pierces any side of any of the purple rectangles. We check all sides of the rectangles in $O(\log n)$ time, and do this for both choices of pairs of opposite corners. Thus:

▶ **Theorem 12.** *There exists a data structure that preprocesses a trajectory in $O(n \log n)$ time, so that given a query subtrajectory, it answers whether the subtrajectory is 2-coverable in $O(\log n)$ time.*

3.2 Is a query subtrajectory 3-coverable?

We provide a sketch of the algorithm for $k = 3$, see the full version for details. In preprocessing time, we construct Tools 9, 10 and 11. At query time, we use Tool 9 to compute the bounding box of the subtrajectory. By Observation 3 there is a square in one of the corners of the bounding box. We then use a combination of Tool 10 and 11 to compute the bounding box of the uncovered subsegments. See Figure 10.

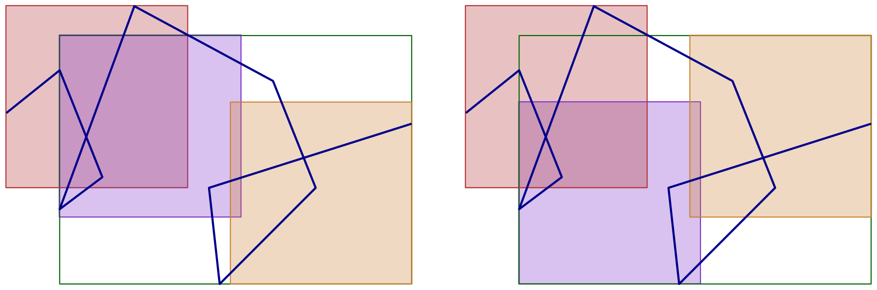

Figure 10 The two choices for the last two squares, as given by their bounding box (thin green).

The bounding box of the uncovered subsegments gives two possible positions for the last two squares. It remains only to check if the three squares cover the subtrajectory. For this final step we use Tool 10 to check for any edges piercing the boundary of the union of the three squares. We require one final check using Tool 11 to detect if the trajectory exits the top of the purple square in the right diagram of Figure 10. Putting this together yields:

▶ **Theorem 13.** *There exists a data structure that preprocesses a trajectory in $O(n \log n)$ time, so that given a query subtrajectory, it answers whether the subtrajectory is 3-coverable in $O(\log n)$ time.*

4 Problem 3: The Longest Coverable Subtrajectory Problem

4.1 The longest 2-coverable subtrajectory

Suppose we were given the starting point of the longest coverable subtrajectory. Then by parametric search in conjunction with the data structure in Theorem 12, we can compute the latest ending point so that the subtrajectory is still coverable.

Hence, the problem reduces to finding a set of points so that the starting point of the longest coverable subtrajectory is guaranteed to be inside the set. In the full version of the paper, we show that there is a set of size $O(n2^{\alpha(n)})$ that can be computed in $O(n2^{\alpha(n)} \log^2 n)$ time that is guaranteed to contain the optimal starting point. Hence, we have:

▶ **Theorem 14.** *There is an $O(n2^{\alpha(n)} \log^2 n)$ time algorithm to compute the longest 2-coverable subtrajectory of the trajectory.*

An open problem is whether there is a polynomial time algorithm to compute the longest 3-coverable subtrajectory of a given trajectory. New ideas are required to compute the set of additional starting points for 3-coverable subtrajectories.

References

1 Maria Luisa Damiani, Hamza Issa, and Francesca Cagnacci. 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*, pages 253–262, 2014.

2 Zvi Drezner. On the rectangular p-center problem. *Naval Research Logistics (NRL)*, 34(2):229–234, 1987.

3 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Information Processing Letters*, 12(3):133–137, 1981.

4 Joachim Gudmundsson and Michael Horton. Spatio-temporal analysis of team sports. *ACM Computing Surveys (CSUR)*, 50(2):22, 2017.

5 Joachim Gudmundsson, Marc van Kreveld, and Frank Staals. Algorithms for hotspot computation on trajectory data. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 134–143, 2013.

6 Michael Hoffmann. Covering polygons with few rectangles. In *Abstracts 17th European Workshop Computational Geometry*, pages 39–42, 2001.

7 Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM Journal of Computing*, 13(1):182–196, 1984.

8 Doron Nussbaum. Rectilinear p-piercing problems. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation, ISSAC*, pages 316–323, 1997.

9 Sanjib Sadhu, Sasanka Roy, Subhas C. Nandy, and Suchismita Roy. Linear time algorithm to cover and hit a set of line segments optimally by two axis-parallel squares. *Theoretical Computer Science*, 769:63–74, 2019.

10 Michael Segal. On piercing sets of axis-parallel rectangles and rings. *International Journal of Computional Geometry and Applications*, 9(3):219–234, 1999.

11 Micha Sharir and Emo Welzl. Rectilinear and polygonal p-piercing and p-center problems. In *Proceedings of the 12th Annual Symposium on Computational Geometry*, pages 122–132, 1996.

12 Andreas Stohl. Computation, accuracy and applications of trajectories—A review and bibliography. *Developments in Environmental Science*, 1:615–654, 2002.