Fifteenth International Workshop on Algebraic and Combinatorial Coding Theory
June 18-24, 2016, Albena, Bulgaria *pp. 244–249*

A Dynamic Data Structure for Segment Intersection Queries: Extended Abstract [1]

KALINA PETROVA kalina.h.petrova@gmail.com
Princeton University

ROBERT TARJAN prof.tarjan@gmail.com
Department of Computer Science, Princeton University, and Intertrust Technologies

Abstract. This paper deals with the problem of supporting a set of axis-aligned segments in the plane so that new segments can be inserted and for any given axis-aligned query rectangle Q, all intersection points of pairs of segments inside Q can be reported. This problem is motivated by scenarios of motion planning in which a number of moving entities on predetermined trajectories are given and collisions are to be prevented. If N is the number of segments and K is the number of points reported in a query, there is an algorithm that supports queries in $O(\sqrt{N}\log N + K\log^3 N)$ time and insertions in $O(\sqrt{N}\log^3 N)$ time using $O(N\log^2 N)$ space and $O(N\sqrt{N}\log^3 N)$ preprocessing time. It is hoped that this solution will provide insight in the general problem of dynamization of range query algorithms and pairwise intersection query algorithms.

1 Introduction

Range searching is one of the most studied areas of computational geometry. In this paper we are going to consider a range searching problem in which we are interested in the intersection points between pairs of segments in the plane. The formal definition of the problem is as follows.

The Segment Intersection Problem. Given a set O of N axis-aligned segments in the plane with $O = V \cup H$ where V is the set of vertical segments and H is the set of horizontal segments, support the following operations:

- *Query:* given an axis-aligned rectangle Q, report all K points $p = H_i \cap V_j \cap Q$, where $H_i \in H, V_j \in V$.
- *Insertion:* add a new axis-aligned segment o to O.

The paper Finding Pairwise Intersections Inside a Query Range [1] solves the static version of the problem (it supports queries but not insertions). The reason why the dynamic version is significantly harder than the static one is that there can be as many as N new intersection points as a result of an insertion. This makes it hard to achieve a sublinear time complexity *both* for the insertion and for the query operation.

[1]This research is supported by Princeton University.

The study of pairwise intersection points of geometrical objects is motivated by motion planning. If we imagine that the geometrical objects are the trajectories of moving entities, like robots in a factory or airplanes in the sky, we might want to know all points where two entities may potentially collide so we can take measures to avoid collisions. Furthermore, it is likely that we want to ask this question for particular areas only. The trajectories of objects can generally be arbitrary curves but we restrict ourselves to the case where they are axis-aligned segments for now.

We provide an algorithm and a set of data structures that allows us to perform queries in $O(\sqrt{N}\log N + K\log^3 N)$ amortized time and insertions in $O(\sqrt{N}\log^3 N)$ amortized time using $O(N\log^2 N)$ space and $O(N\sqrt{N}\log^3 N)$ preprocessing time.

2 Approach

We will follow the skeleton of the solution of the static version of the problem described in [1]. We make use of the concept of *witness points*. Each segment O_i has up to two points on it that are considered the witness points *associated with it* (we will also say that O_i is associated with each witness point that is associated with it). The following points comprise the set of witness points W.

- For every vertical segment $V_i \in O$, the topmost and bottommost intersection points of V_i with a horizontal segment are witness points associated with V_i.
- For every horizontal segment $H_j \in O$, the leftmost and rightmost intersection points of H_j with a vertical segment are witness points associated with H_j.

Next, the solution of the static version of the problem in [1] defines the set $O^*(Q)$ to consist of the following members of O.

- For each witness point W_l inside Q, if $O_i \in O$ is the segment associated with W_l, then $O_i \in O^*(Q)$.
- Let $V(Q)$ be all vertical segments in O that cross Q completely from top to bottom, and let $H(Q)$ be all horizontal segments in O that cross Q completely from left to right. Then if $V(Q) \neq \varnothing$ and $H(Q) \neq \varnothing$, we have $V(Q) \subseteq O^*(Q)$.

Lemma 1 in [1] states that if $V_i \in V$ and $H_j \in H$ intersect inside Q, then either $V_i \in O^*(Q)$ or $H_j \in O^*(Q)$. Note that this lemma applies to the dynamic case as well as it only depends on the current contents of O.

Our solution follows the skeleton of that in [1], but we use different data structures. We support a data structure W_{points} that helps us, when given a query rectangle Q, to find all witness points in Q. We also support data structures V_{cross} and H_{cross}, which can report all the segments crossing a query rectangle Q completely from top to bottom and from left to right respectively. Finally, we store all segments in O in a data structure S that can report all segments intersecting an axis-aligned query segment.

Given a query rectangle Q, we perform the following operations, as described in [1].

1. Perform a query in W_{points} to find all witness points in Q. For each reported witness point, insert the corresponding segment in $O^*(Q)$.

2. Perform queries in V_{cross} and H_{cross} to decide if the number of segments crossing Q completely from top to bottom, and the number of segments crossing Q completely from left to right, are both not zero. If so, report all segments crossing Q completely from top to bottom, and insert them in $O^*(Q)$.

3. For each $O_i \in O^*(Q)$, perform an intersection query with the range $O_i \cap Q$ in S, to find all objects $O_j \neq O_i$ intersecting O_i inside Q. Note that if O_i is vertical, we only consider O_j's which are horizontal and vice versa.

Provided that W_{points}, V_{cross}, H_{cross}, and S are up to date, the proof of correctness of this procedure coincides with the proof of **Lemma 1** in [1].

Now let us consider what data structures we need for W_{points}, V_{cross}, H_{cross}, and S.

We are going to make use of the data structure described in the paper An Implementation of a Multidimensional Dynamic Range Tree Based on an AVL Tree [2], which we will call an *AVL range tree*. An AVL range tree keeps a set of n points in k-dimensional space and supports the following operations.

- *Report:* Given a k-dimensional axis-aligned box B, report all points inside B in time $O(\log^k n + t)$, where t is the number of points found.

- *Add:* Add a new point with coordinates $(x_1, \dots, x_k)$ in time $O(\log^k n)$.

- *Delete:* Delete the point with coordinates $(x_1, \dots, x_k)$ in time $O(\log^k n)$.

From now on we will represent a vertical segment as (x, y_1, y_2), where x is the x coordinate of the segment, and y_1 and y_2, $y_1 < y_2$, are the y coordinates of the lower and upper end of the segment respectively. Likewise, we represent horizontal segments using three numbers (x_1, x_2, y) and we represent rectangles using four numbers (x_1, x_2, y_1, y_2).

First, consider the data structure S. We need to keep all segments $O_i \in O$ in a data structure such that when given a segment $O_i \cap Q$, $O_i \in O^*(Q)$, we can report the segments in O that intersect $O_i \cap Q$. We are going to keep all vertical segments in one data structure and all horizontal segments in another.

We will describe the data structure S_h that keeps all horizontal segments, the data structure S_v for vertical segments is analogous. We represent a horizontal segment (x_1, x_2, y) as a point in 3D space, and we keep all such points corresponding to horizontal segments in a three-dimensional AVL range tree S_h. When we have to report all horizontal segments that intersect a given vertical segment $O_i \cap Q = (y_1, y_2, x)$, this is equivalent to finding all points (x_1, x_2, y) such that $x_1 \in (-\infty, x], x_2 \in [x, \infty), y \in [y_1, y_2]$. Thus we perform a report operation with the box $(-\infty, x] \times [x, \infty) \times [y_1, y_2]$, and the reported points correspond to the segments we are looking for. Whenever a new horizontal segment is added to O, we add it to S_h.

Now consider the data structures V_{cross} and H_{cross}. We only describe V_{cross}, the case of H_{cross} being analogous. A vertical segment (x', y'_1, y'_2) crosses a query rectangle $Q = (x_1, x_2, y_1, y_2)$ completely from top to bottom if and only if $x' \in [x_1, x_2]$, $y'_1 \in (-\infty, y_1]$ and $y'_2 \in [y_2, \infty)$. Thus we store all vertical segments in a three-dimensional AVL range tree V_{cross} as points in 3D space. To find all vertical segments that cross a query rectangle $Q = (x_1, x_2, y_1, y_2)$ completely from top to bottom, we just query V_{cross} with the box $[x_1, x_2] \times (-\infty, y_1] \times [y_2, \infty)$. When a new vertical segment (x', y'_1, y'_2) is inserted, we add the 3D point corresponding to it to V_{cross}.

The data structure W_{points} is more complicated. In the static case of the problem a 2-dimensional AVL range tree suffices for it and yields a good performance. However, in the dynamic case the insertion operation may change as many as N of the witness points, so we have to consider alternatives. We reduce the problem of maintaining W_{points} to another problem which we call the Point Projection Problem.

3 Reduction to the Point Projection Problem

In this section we describe how to represent the witness points together with the segments they are associated with as points in 3D space and how to translate our current problem into a problem about these points which we call the *Point Projection Problem.* Let us divide the witness points in the following four types: topmost and bottommost intersection points of a vertical segment, and leftmost and rightmost intersection points of a horizontal segment. To be able to report all witness points inside a query rectangle Q, we are going to use four different data structures - one for each type of witness points. We are going to describe the data structure that handles the witness points which are topmost intersection points of vertical segments, the other cases are analogous. We are going to keep each witness point with coordinates (x, y) that is the topmost intersection point of a vertical segment (x, y_1, y_2) as a point in 3D space (x, y_2, y). When we want to find all witness points inside a query rectangle $Q = (x'_1, x'_2, y'_1, y'_2)$, we have to report all points (x, y_2, y) such that $(x, y_2, y) \in [x'_1, x'_2] \times (-\infty, \infty) \times [y'_1, y'_2]$. We are going to keep these witness points in a data structure $W_{points}^{topmost}$ that is yet to be described.

We are going to support two auxiliary data structures called Max_h and $NoIntersection_v$. Max_h will keep all horizontal segments as points in 3D space. For it we use a 3D AVL range tree that supports query for the element with largest z coordinate in a given box (this is a slight modification of the normal query in an AVL range tree that does not change the time complexity). $NoIntersection_v$ will keep all vertical segments that have no intersection points with horizontal segments. The vertical segments will be represented as points in 3D space and will be stored in a 3D AVL range tree.

When we are inserting a new vertical segment (x'', y_1'', y_2'') in O, we need to find its topmost intersection point first. We do this by querying Max_h, which takes $O(\log^3 N)$. If the vertical segment we are inserting is intersected by at least one horizontal segment, we take its topmost intersection point y and add (x'', y_2'', y) to $W_{points}^{topmost}$. If it is not, we add (x'', y_1'', y_2'') to $NoIntersection_v$.

When we are inserting a new horizontal segment (x_1'', x_2'', y''), we have to increase the y-coordinate to y'' of all witness points with $(x, y_2, y) \in [x_1'', x_2''] \times [y'', \infty) \times (-\infty, y'')$. This will be done using an update operation in $W_{points}^{topmost}$ that remains to be described. We also query $NoIntersection_v$ to see if our horizontal segment intersects any of the vertical segments in it and if so, we delete these vertical segments from $NoIntersection_v$ and add them to $W_{points}^{topmost}$ with the witness point obtained from their intersection with the new horizontal segment. Finally, we add the new horizontal segment to Max_h.

The requirements for $W_{points}^{topmost}$ are going to shape our formulation of the Point Projection Problem, which is as follows. We need $W_{points}^{topmost}$ to store points in 3D space and to support the following operations:

1. *Insertion:* Add a point (x, y, z) to $W_{points}^{topmost}$.

2. *Query:* Report all points in $W_{points}^{topmost}$ in a given box $[x_1, x_2] \times [y_1, \infty) \times [z_1, z_2]$.

3. *Update:* For all points (x, y, z) in $W_{points}^{topmost}$ in the box $[x_1, x_2] \times (-\infty, \infty) \times (-\infty, z_2]$, increase z to z_2.

4 Results

Theorem 1. *There exists an algorithm that solves the Point Projection Problem with amortized time complexity $O(\sqrt{n} \log(n+m) + k)$ for query operations, $O(\sqrt{n} \log^2(n) \log(n+m))$ for update operations, $O(\sqrt{n} \log^2(n) \log(n+m))$ for insertions, and space complexity $O(n \log(n) \log(n+m))$, where n is the total number of points, m is the total number of update operations, and k is the number of points reported in a query.*

Theorem 2. *There exists an algorithm that solves the Segment Intersection Problem with amortized time complexity* $O(\sqrt{N}\log(N) + K\log^3(N))$ *for query operations,* $O(\sqrt{N}\log^3(N))$ *for insertions and space complexity* $O(N\log^2(N))$*, where* N *is the number of segments and* K *is the number of reported points in a given query.*

We omit the proofs for the sake of brevity.

5 Conclusion

To summarize, we had posed ourselves the goal to design an algorithm that supports the following operations on a set of axis-aligned segments in the plane.

- *Query*: for a given rectangle Q, report all intersection points p of pairs of segments with $p \in Q$.
- *Insertion*: insert a new segment.

We achieved this goal with an amortized time complexity of $O(\sqrt{N}\log N + K\log^3 N)$ for queries and $O(\sqrt{N}\log^3 N)$ for insertions and space complexity $O(N\log^2 N)$. Our result is remarkable because it handles both queries and insertions in sublinear time, which seemed like a hopeless aim at first as each insertion can potentially add as many as N new intersection points. In the future, we are planning to consider generalizations to segments with arbitrary orientations, sequences of segments and axis-aligned rectangles.

6 Acknowledgements

We would like to thank professor Bernand Chazelle for the useful ideas and Ali D. Mehrabi, coauthor of [1], for explaining the motivation behind their work. We would also like to thank the Computer Science Department at Princeton University for supporting this research.

References

[1] Mark de Berg and Joachim Gudmundsson and Ali D. Mehrabi, Finding Pairwise Intersections Inside a Query Range, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, 2015, 236–248.

[2] Michael G. Lamoureux, An Implementation of a Multidimensional Dynamic Range Tree Based on an AVL Tree, University of New Brunswick, Faculty of Computer Science, 1995.