

# Finding Popular Places

Marc Benkert<sup>1</sup>, Bojan Djordjevic<sup>2</sup>, Joachim Gudmundsson<sup>2</sup>, and Thomas Wolle<sup>2</sup>

<sup>1</sup> Department of Computer Science, Karlsruhe University, Germany.  
mbenkert@ira.uka.de

<sup>2</sup> NICTA\* Sydney, Australia  
{joachim.gudmundsson,bojan.djordjevic,thomas.wolle}@nicta.com.au

**Abstract.** Widespread availability of location aware devices (such as GPS receivers) promotes capture of detailed movement trajectories of people, animals, vehicles and other moving objects, opening new options for a better understanding of the processes involved. We investigate spatio-temporal movement patterns in large tracking data sets. Specifically we study so-called ‘popular places’, that is, regions that are visited by many entities. We present upper and lower bounds.

## 1 Introduction

Technological advances of location-aware devices, surveillance systems and electronic transaction networks produce more and more opportunities to trace moving individuals. Consequently, an eclectic set of disciplines including geography, market research, data base research, animal behaviour research, surveillance, security and transport analysis shows an increasing interest in movement patterns of entities moving in various spaces over various times scales [1, 10, 17]. In the case of moving animals, movement patterns can be viewed as the spatio-temporal expression of behaviours, e.g. in flocking sheep or birds assembling for the seasonal migration. In a transportation context, a movement pattern could be a traffic jam.

In this paper we will focus on the problem of computing ‘popular places’ (also called ‘convergence patterns’ in [19, 20]) among geospatial lifelines. The input is a set  $E$  of  $n$  moving point objects  $A_1, \dots, A_n$  whose locations are known at  $\tau$  consecutive time-steps  $t_1, \dots, t_\tau$ , that is, the trajectory of each object is a polygonal line that can self-intersect, see Fig. 1. For brevity, we will call moving point objects *entities* from now on, and when it is clear from the context, we use  $A$  to denote an entity or its trajectory. It is assumed that an entity moves between two consecutive time steps on a straight line and the velocity of an entity along a line segment of the trajectory is constant. Given a set of  $n$  moving entities in the plane, an integer  $k > 0$  and a real value  $r > 0$ , a *popular place* is a square of side length  $r$ , that is visited by at least  $k$  entities. Throughout the article we will for simplicity assume  $r = 1$ . Note that the entities do not have to be in the square simultaneously. Spatio-temporal patterns have traditionally been considered in two settings: the discrete case where only the discrete time steps are considered and the continuous case where the polygonal lines connecting the input points are considered. Recently it has been argued [6, 13] that the continuous model is becoming more important since trajectories will have to be compressed (simplified) to allow for fast computations. Nowadays it is not unusual that the coordinates are recorded with a frequency of one second. A popular place in the two different models is defined as follows (see Fig. 1).

**Definition 1.** *Given a set of  $n$  moving entities in the plane, an integer  $k > 0$  and a real value  $r > 0$ . An axis aligned square  $\sigma$  of side length  $r$  is a popular place in the discrete model if  $\sigma$  contains input points from at least  $k$  different entities. In the continuous model  $\sigma$  is a popular place if it is intersected by polylines from at least  $k$  different entities.*

---

\* National ICT Australia is funded through the Australian Government’s Backing Australia’s Ability initiative, in part through the Australian Research Council.

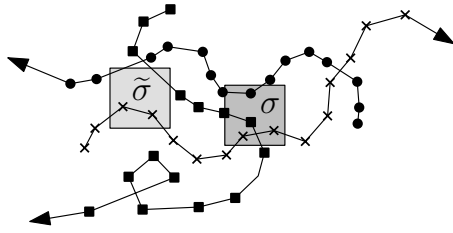


Fig. 1: An example where three entities  $\Lambda_1, \Lambda_2$  and  $\Lambda_3$  are traced during 16 time steps. For  $k = 3$  the square  $\tilde{\sigma}$  is a popular place only in the continuous model. While  $\sigma$  is a popular place for  $k = 3$  in both the discrete and continuous model.

Recently there has been considerable research in the area of analysing and modelling spatio-temporal data. In the database community research has mainly focussed on indexing databases so that basic spatio-temporal queries concerning the data can be answered efficiently. Typical queries are spatio-temporal range queries, spatial or temporal nearest neighbours, see for example the work by Sältenis et al. [23] and Hadjieleftheriou et al. [18]. From a data mining perspective Verhein and Chawla [24] used association rule mining to detect patterns in spatio-temporal sets. They defined a region to be a *source*, *sink* or a *thoroughfare* depending on the number of objects entering, exiting or passing through the region. Mamoulis et al. [21] studied periodic patterns, e.g. yearly migration patterns or daily commuting patterns. Recently there have been several papers considering the problem of detecting flock patterns and leadership patterns [2, 3, 5, 14].

Precursory to this work Laube et al. [19, 20] proposed the REMO framework (Relative Motion) which defines similar behaviour in groups of entities. They defined patterns such as ‘flock’, ‘convergence’, ‘trend-setting’ and ‘leadership’ based on similar movement properties such as speed, acceleration or movement direction, and gave algorithms to compute them efficiently. They proposed an input model where a ray was drawn from the current position of each entity that corresponds to its direction. That is, the coordinates and direction of the entities are known at the initial time step and the aim is to find or forecast a popular place (assuming the entities do not change their direction).

As mentioned in earlier work [5, 14, 15], specifying exactly which of the patterns should be reported is often a subject for discussion. For the discrete model we design a general algorithm that can generate the following output:

- the popular place with the most number of entities (detect maximum),
- a set of rectangles of width 1 and height 2 such that each reported rectangle contains a popular place and all popular places are covered by the reported rectangles (approximate),
- a set of polygons  $\mathcal{H}(E)$  such that any axis-aligned unit square with centre in a polygon of  $\mathcal{H}(E)$  is a popular place (report all).

In the continuous model we only describe how to find the set  $\mathcal{H}(E)$ . However, one can easily modify it to any of the output models listed above.

In the Section 2 we present an algorithm for the discrete model, followed by an  $\Omega(\tau n \log \tau n)$  time lower bound. In Section 4 we present an  $O(\tau^2 n^2)$  time algorithm in the continuous model. Finally, in Section 5, we prove an  $\Omega(n^2)$  lower bound in the continuous model.

## 2 A fast algorithm in the discrete model

We start with the discrete version of the problem. A set of  $n$  entities is traced over a period of  $\tau$  time steps, generating  $\tau n$  points in the plane that correspond to the positions of the tracked entities. We will refer to the  $\tau n$  points as *input points*. The input parameter  $k > 0$  defines the minimum number of entities defining a popular place, see Definition 1.

We design an algorithm that reports the popular place with the largest number of entities in  $O(\tau n \log \tau n)$  time. In Section 2.4 we show how to modify it to produce more general output. This problem closely resembles the coloured range counting problem [16] where the input is a set of  $\bar{n}$  points, each point having one of  $\bar{m}$  possible colours, and the aim is to preprocess the points such that the number of different colours inside a given query range can be reported.

**Fact 1** (Theorem 5.1 in [16]) *A set  $S$  of  $\bar{n}$  coloured points in the plane can be preprocessed into a data structure of size  $O(\bar{n}^2 \log^2 \bar{n})$  such that for any axis-parallel query rectangle  $q = [a, b] \times [c, d]$ , the number of distinctly-coloured points in  $q$  can be reported in  $O(\log^2 \bar{n})$  time.*

This result can be used to obtain a simple approximation algorithm. Let  $k_{\max}$  be the largest number of entities contained in a square of side length 1, i.e.  $k_{\max}$  is the size of a maximum popular place. We say that an algorithm is an  $\alpha$ -approximation algorithm if it returns a square of side length  $\alpha$  that contains at least  $k_{\max}$  entities. A 2-approximation algorithm can be obtained by performing  $n$  coloured range counting queries where each query square has side length 2 and is centred at an input point. The algorithm requires  $O(\tau^2 n^2 \log^2 \tau n)$  time and space. The main drawback is the space usage. In the applications considered the size of the input may be very large as noted in the introduction. We will present an exact algorithm that only uses  $O(\tau n)$  space and runs in  $O(\tau n \log \tau n)$  time.

The idea of our algorithm is to use a vertical sweep line  $\ell$  sweeping the points from left to right. Together with the sweep line we sweep a vertical strip  $\text{str}_\ell$  of width 1 whose right boundary is  $\ell$ . Each of the  $\tau n$  input points induces two event points, one when the point enters  $\text{str}_\ell$  and one when the point leaves  $\text{str}_\ell$ . We refer to these types as *start* and *end* events.

For a start event, say that an input point  $p$  belonging to entity  $\Lambda$  enters  $\text{str}_\ell$ , we update our data structures and check for the largest popular place located in  $\text{str}_\ell$  that is visited by  $\Lambda$ . Such a popular place must obviously be contained in the axis-aligned rectangle  $R_p$  having width 1, height 2 and  $p$  on the midpoint of its right vertical segment, see Fig. 2a. If we check at every start event  $p$  for a largest popular place within  $R_p$  then we will find the maximum popular place. For an end event, say a point  $p$  belonging to  $\Lambda$  is about to leave  $\text{str}_\ell$ , we simply remove it from the current data structures.

As described above, the general idea is to sweep a vertical strip  $\text{str}_\ell$  of width 1 from left to right. Then, for each start event  $p$  the aim is to find a largest popular place containing  $p$ . This could be done by sweeping all the points within  $R_p$  with a unit square  $s$ , while keeping track of the number of entities within  $s$  at any time. However, since the number of points within  $R_p$  is  $O(\tau n)$  this might take  $O(\tau n \log \tau n)$  time per event.

Instead we are going to show how we can maintain a set of trees such that given a  $y$ -interval  $[a, b]$  (the  $y$ -coordinates of the top- and bottom side of  $R_p$ ), we can find the  $y$ -value of the centre of a unit square that contains the largest number of different entities in  $O(\log \tau n)$  time per query. Below we will show how we can achieve this by maintaining a tree for each entity separately in  $O(\log \tau)$  time per event and in Section 2.2 we show how we can merge this information into one tree  $T_{\text{int}}$  that can be queried and updated in  $O(\log \tau n)$  time.

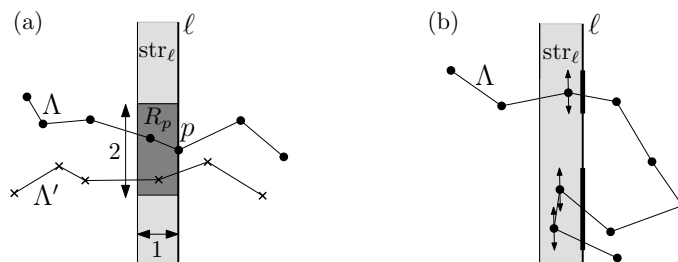


Fig. 2: (a) Finding the popular places in  $R_p$  visited by  $\Lambda$ . (b) The set  $I_A$  is indicated by the bold, solid line segments on  $\ell$ .

## 2.1 One structure for each entity

We maintain a set of disjoint  $y$ -intervals  $I_A$ , for each entity  $A$ , such that at any event point during the sweep  $I_A$  contains exactly the  $y$ -intervals for which a square  $s$  in  $\text{str}_\ell$  with centre in an interval in  $I_A$  contains a point of  $A$ , see Fig. 2b. The square containing a maximum popular place is a square whose centre is contained in a maximum number of such  $y$ -intervals.

Next we show how such a data structure can be maintained efficiently. We will maintain two trees  $B_A$  and  $T_A$  for each entity  $A$ . The tree  $B_A$  is a balanced binary search tree on the points of  $A$  currently within  $\text{str}_\ell$  and ordered with respect to their  $y$ -coordinates.  $B_A$  can be queried and updated in  $O(\log \tau)$  time using, for example, Red-Black trees (see e.g. [7]). The tree  $T_A$  will store the set  $I_A$  of intervals w.r.t. the current position of  $\ell$ . The leaves of  $T_A$  store the endpoints of the intervals in  $I_A$  ordered on their  $y$ -coordinates. Each leaf also contains a pointer to the leaf in  $T_A$  containing the other endpoint. Inserting and deleting a new interval can be done in  $O(\log \tau)$  time per update:

Assume we are about to process a start event  $p_A = (x, y)$ . Let  $I = [y - 1/2, y + 1/2]$  and note that any unit square within  $\text{str}_\ell$  with centre in  $I$  will contain  $p_A$ . The point is inserted into  $B_A$  and then a range query is performed in  $T_A$  that reports the intervals of  $I_A$  intersecting  $I$ . Since the intervals in  $I_A$  are disjoint and have length at least one,  $I$  may intersect at most two intervals in  $I_A$ . Thus, finding the intersecting intervals can be done in  $O(\log \tau)$  time. If the number of intersecting intervals is zero then  $I$  is inserted into  $T_A$ . If  $I$  intersects one interval  $I_1$  then  $I_1$  is deleted and the interval  $I \cup I_1$  is inserted. Finally, if two intervals  $I_1$  and  $I_2$  are intersected then  $I_1$  and  $I_2$  are deleted and  $I \cup I_1 \cup I_2$  is inserted.

In the case when  $\text{str}_\ell$  is about to process an end event  $p_A = (x, y)$ , update the trees in a similar manner. Report the two adjacent neighbours  $p_1 = (x_1, y_1)$  and  $p_2 = (x_2, y_2)$  of  $p_A$  in  $B_A$ , and delete  $p_A$ . Assume without loss of generality that  $y_1 < y < y_2$ . Let  $I$  be as defined above and let  $I'$  be the interval in  $I_A$  containing  $I$ . We have to distinguish three cases:

1. If  $|y_2 - y_1| \leq 1$  then we are done since  $I'$  does not change.
2. If  $\min\{|y - y_1|, |y - y_2|\} > 1$  then  $I' = I$  is deleted from  $T_A$ .
3. If  $|y - y_1| > 1$  and  $|y - y_2| \leq 1$  then  $I'$  is deleted from  $T_A$  and the interval  $I' \setminus [y - 1/2, y_2 - 1/2]$  is inserted. The case when  $|y - y_2| > 1$  and  $|y - y_1| \leq 1$  is symmetric.

We denote the set of all trees  $T_A$  and  $B_A$  by  $\mathcal{T}^{\text{ent}}$  and  $\mathcal{B}^{\text{ent}}$ , respectively. Since the total number of events is  $2\tau n$  the below corollary follows immediately.

**Corollary 1.** *Throughout the sweep, the sets  $\mathcal{T}^{\text{ent}}$  and  $\mathcal{B}^{\text{ent}}$  can be maintained in  $O(\tau n \log \tau)$  time requiring  $O(\tau n)$  space.*

## 2.2 Maintaining the status of the sweep

We store all intervals that are currently in  $\text{str}_\ell$  in a balanced binary tree  $T_{\text{int}}$ . We will use  $T_{\text{int}}$  to perform the maximum popular place query for  $R_p$ . Let  $\mathcal{I} = \bigcup_A I_A$ . For simplicity we assume that all start and end points of intervals in  $\mathcal{I}$  are pairwise disjoint. The leaf set of  $T_{\text{int}}$  corresponds to the set of start and end points in  $\mathcal{I}$  ordered w.r.t. their  $y$ -coordinates.

Since there are  $\tau n$  points,  $T_{\text{int}}$  contains at most  $O(\tau n)$  leaves and thus at most  $O(\tau n)$  vertices at all times. During the sweep  $T_{\text{int}}$  is maintained as follows: every time a tree  $T_A$  is updated we perform the corresponding update operation in  $T_{\text{int}}$ . One update in  $T_A$  requires the deletion and insertion of a constant number of leaves in  $T_{\text{int}}$ . Thus, one update operation in  $T_A$  induces update operations in  $T_{\text{int}}$  that can be performed in  $O(\log \tau n)$  time. Since the sweep conducts  $2\tau n$  update operations, we have the following:

**Lemma 1.** *Throughout the sweep, the tree  $T_{\text{int}}$  can be maintained in  $O(\tau n \log \tau n)$  time requiring  $O(\tau n)$  space.*

Next, we show how we can store appropriate information in  $T_{\text{int}}$  in order to perform maximum popular place queries.

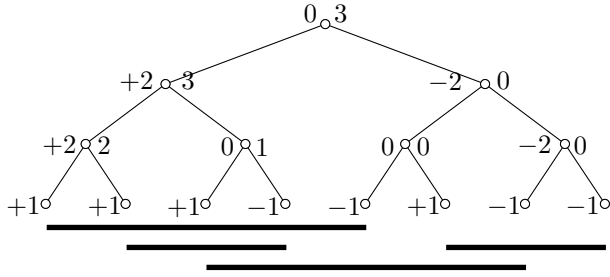


Fig. 3: The values  $\text{sum}$  (left) and  $\text{max}_{\text{pre}}$  (right) for the depicted tree  $T_{\text{int}}$ .

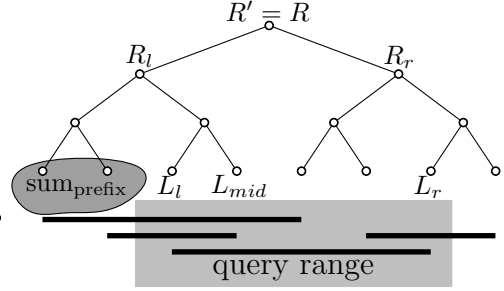


Fig. 4: Querying  $T_{\text{int}}$  for the maximum stabbing number  $\text{max}_{l,r}$ .

### 2.3 Extending $T_{\text{int}}$ to allow for efficient queries

A point  $p = (x, y)$  is said to *stab* an interval  $[a, b]$  if  $y \in [a, b]$ . The *stabbing number* of  $p$  w.r.t. a set of intervals  $I$  is the number of intervals in  $I$  that  $p$  stabs. Note that for a start event  $p$  we have to find the point in  $\ell \cap R_p$  having maximum stabbing number w.r.t.  $\mathcal{I}$ . Ideally we could store the number of intervals that a point corresponding to a leaf  $L$  in  $T_{\text{int}}$  stabs. However, one event could require the update of  $O(\tau n)$  of these numbers which would be too costly to maintain. Instead we maintain this information implicitly in order to do updates as well as queries in  $O(\log \tau n)$  time.

For this we store two values,  $\text{sum}(V)$  and  $\text{max}_{\text{pre}}(V)$ , with each vertex  $V$  in  $T_{\text{int}}$ , see Fig. 3. For leaves  $L$  we define  $\text{sum}(L)$  and  $\text{max}_{\text{pre}}(L)$  to be  $+1$  if  $L$  corresponds to an interval start point and to be  $-1$  otherwise. For inner vertices  $V$  let  $V_{\text{left}}$  and  $V_{\text{right}}$  denote the left and the right child of  $V$ , respectively. Then,  $\text{sum}$  and  $\text{max}_{\text{pre}}$  are defined as follows:

$$\text{sum}(V) := \text{sum}(V_{\text{left}}) + \text{sum}(V_{\text{right}}), \quad \text{and}$$

$$\text{max}_{\text{pre}}(V) := \max\{\text{max}_{\text{pre}}(V_{\text{left}}), \text{sum}(V_{\text{left}}) + \text{max}_{\text{pre}}(V_{\text{right}})\}.$$

Let  $L_1(V), \dots, L_m(V)$  be the sequence of all leaves contained in the subtree rooted at  $V$  enumerated from left to right. The intuition of the definitions is that

$$\text{sum}(V) = \sum_{j=1}^m \text{sum}(L_j(V)) \quad \text{and} \quad \text{max}_{\text{pre}}(V) = \max_{1 \leq i \leq m} \sum_{j=1}^i \text{sum}(L_j(V)).$$

When performing an update operation in  $T_{\text{int}}$ , i.e. deleting or inserting a leaf  $L$ , updating  $\text{sum}$  and  $\text{max}_{\text{pre}}$  is only required on the path from  $L$  to the root. Hence, updating  $\text{sum}$  and  $\text{max}_{\text{pre}}$  takes  $O(\log \tau n)$  time per update operation.

**Lemma 2.** *Throughout the sweep, the values  $\text{sum}$  and  $\text{max}_{\text{pre}}$  can be maintained in  $O(\tau n \log \tau n)$  time requiring  $O(\tau n)$  space.*

Recall the processing of the sweep. When we arrive at a start event  $p = p_A$ , we first perform the required updates in  $B_A$  and  $T_A$ , update  $T_{\text{int}}$  accordingly and then query  $T_{\text{int}}$  for the most popular place in  $R_p$ . Next, we show how the query can be done in  $O(\log \tau n)$  time.

Let  $L_1, L_2, \dots$  be the set of leaves in  $T_{\text{int}}$  ordered from left to right. Each of the leaves in  $T_{\text{int}}$  is associated with the  $y$ -value of the stored start or end point. By performing two searches in  $T_{\text{int}}$  we can find the leftmost leaf  $L_l$  in  $T_{\text{int}}$  whose  $y$ -value is at least  $y_p - 1/2$  and the rightmost leaf  $L_r$  whose  $y$ -value is at most  $y_p + 1/2$  in  $O(\log \tau n)$  time. This defines our query range within  $T_{\text{int}}$ : the goal is to find the leaf between (and including)  $L_l$  and  $L_r$  whose stored point stabs the maximum number of intervals among these leaves. We denote this maximum stabbing number by  $\text{max}_{l,r}$ . We have that  $\text{max}_{l,r} = \max_{l \leq m \leq r} \{\sum_{j=1}^m \text{sum}(L_j)\}$ . We claim that  $\text{max}_{l,r}$  can be calculated by walking along the search paths from  $L_l$  and  $L_r$ , respectively, to the root  $R$  of  $T_{\text{int}}$ . We sketch how this is done. Note that the sketch comprises that a leaf with stabbing number  $\text{max}_{l,r}$  can be found and reported in  $O(\log \tau n)$  time.

Let  $R'$  be the least common ancestor of  $L_l$  and  $L_r$  and let  $mid$ , with  $l \leq mid < r$ , be the index such that  $L_{mid}$  is the rightmost leaf in the left subtree of  $R'$ , see Fig. 4. Set  $\text{sum}_{\text{prefix}} := 0$ , and consider the traversal of the search path from  $R$  to  $L_l$ . Every time the search path descends

into the right subtree of a vertex  $V$  add  $\text{sum}(V_{\text{left}})$  to  $\text{sum}_{\text{prefix}}$ , see Fig. 4. When the search path reaches  $L_l$  we have  $\text{sum}_{\text{prefix}} = \sum_{j=1}^{l-1} \text{sum}(L_j)$ .

Let  $\max_l = \max_{l \leq m \leq \text{mid}} \sum_{j=l}^m \text{sum}(L_j)$  and  $\max_r = \max_{\text{mid}+1 \leq m \leq r} \sum_{j=\text{mid}+1}^m \text{sum}(L_j)$ . Once we know  $\max_l$  and  $\max_r$ , the maximum stabbing number can be computed by  $\max_{l,r} = \max\{\text{sum}_{\text{prefix}} + \max_l, \text{sum}(R'_{\text{left}}) + \max_r\}$ . It remains to show how to compute  $\max_l$  and  $\max_r$ .

We compute  $\max_l$  by walking along the path from  $L_l$  to  $R'_{\text{left}}$ . Initially, we set  $\max_l := \max_{\text{pre}}(L_l)$ , and a helper variable  $\text{sum}_{\text{seen}} := \text{sum}(L_l)$ . Let  $L_l = V^0, V^1, \dots, R'_{\text{left}}$  be the sequence of nodes that are traversed when walking from  $L_l$  to  $R'_{\text{left}}$  in  $T_{\text{int}}$ . Each time we encounter a vertex  $V^i$  having  $V^{i-1}$  as a left child we look for a new maximum in the right subtree, i.e.  $\max_l := \max\{\max_l, \text{sum}_{\text{seen}} + \max_{\text{pre}}(V_{\text{right}}^i)\}$  and update  $\text{sum}_{\text{seen}}$  to  $\text{sum}_{\text{seen}} := \text{sum}_{\text{seen}} + \text{sum}(V_{\text{right}}^i)$ . It is not hard to see that in the end of the traversal  $\max_l$  holds the right value. There are  $O(\log \tau n)$  vertices on the path from  $L_l$  to  $R'$ , and each vertex is processed in constant time. Thus, the time to compute  $\max_l$  is  $O(\log \tau n)$ .

In a similar fashion we compute  $\max_r$ . Here, we walk along the path from  $R'_{\text{right}}$  to  $L_r$ , let  $R'_{\text{right}} = V^0, V^1, \dots, L_r$  be the sequence of traversed vertices. Initially, we set  $\max_r := 0$  and  $\text{sum}_{\text{seen}} := 0$ . Each time we encounter a vertex  $V_i$  which is a right child of its parent we look for a new maximum in the left subtree, i.e.  $\max_r := \max\{\max_r, \text{sum}_{\text{seen}} + \max_{\text{pre}}(V_{\text{left}}^{i-1})\}$  and update  $\text{sum}_{\text{seen}}$  to  $\text{sum}_{\text{seen}} := \text{sum}_{\text{seen}} + \text{sum}(V_{\text{left}}^{i-1})$ . Arriving at  $L_r$  we get  $\max_r$  by the final update  $\max_r = \max\{\max_r, \text{sum}_{\text{seen}} + \max_{\text{pre}}(L_r)\}$ .

**Lemma 3.** *Given  $T_{\text{int}}$  and a  $y$ -interval  $[l, r]$  one can, in  $O(\log \tau n)$  time, return the highest stabbing number  $\max_{l,r}$  for any point in  $[l, r]$  together with a point  $p \in [l, r]$  having this stabbing number.*

Now we are ready to summarise the results obtained in this section.

**Theorem 1.** *Given a set  $E$  of  $n$  moving point objects in the plane the unit square containing the maximum number of different entities in the discrete model can be computed in  $O(\tau n \log \tau n)$  time using  $O(\tau n)$  space.*

*Proof.* The proof follows by putting together Corollary 1, Lemma 1, Lemma 2 and Lemma 3. That is, all updates and queries can be done in  $O(\log \tau n)$  per event, thus, in  $O(\tau n \log \tau n)$  time in total.  $\square$

## 2.4 Approximating and reporting all popular places

For a start event  $p = p_A$  we have seen in Section 2.3 how we can detect a unit square in  $R_p$  that contains the maximum number  $k_{\text{max}}$  of different entities in the discrete model among all unit squares contained in  $R_p$ . If we now find that  $k_{\text{max}} \geq k$  we can report  $R_p$  as an approximation for (potentially) all popular places that are contained in  $R_p$ . This leads directly to the following result.

**Theorem 2.** *Given a set  $E$  of  $n$  entities in the plane we can report rectangles of width 1 and height 2 such that each reported rectangle contains a popular place and all popular places are covered by the reported rectangles. This requires  $O(\tau n \log \tau n)$  time and  $O(\tau n)$  space.*

It gets more involved when we want to report the set of polygons  $\mathcal{H}(E)$  such that any axis-aligned unit square with centre in a polygon of  $\mathcal{H}(E)$  defines a popular place and each centre of a popular place is contained in  $\mathcal{H}(E)$ . Say that we process a start event  $p_A$  and find a popular place  $s \subset R_p$ . Then,  $s$  will—most likely—still be a popular place if we shift it slightly to the right. To find out the rightmost position of  $s$  still being a popular place makes the difficulty of the report-all problem.

We first show how we can find all popular places in  $R_p$  when processing a start event  $p$ . We will report *all* popular places as an interval set  $\mathcal{I}_{p,\text{start}}$  such that for each  $I \in \mathcal{I}_{p,\text{start}}$  it holds that  $I \subseteq [a, b]$  where  $a$  and  $b$  are the bottom and topmost  $y$ -coordinates defining  $R_p$  and, furthermore, the squares with centres on any  $y \in I$  give all popular places. The task is to find all leaves between

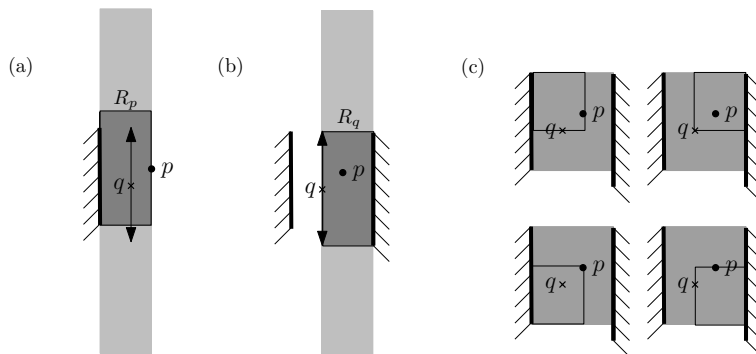


Fig. 5: (a) setting start flags, (b) setting end flags, and (c) the reported polygon (grey) in  $\mathcal{H}(E)$ .

$L_l$  and  $L_r$  (that define the query range induced by  $R_p$ ) whose stabbing number is at least  $k$ . Using the above techniques, we can find a leaf that induces a popular place in  $O(\log \tau n)$  time, meaning that we can find all leaves in  $O(M \log \tau n)$  time per event, where  $M$  is the number of all leaves between  $L_l$  and  $L_r$  whose stabbing number exceeds  $k$ .

Note that each leaf  $L_j$  is associated with a  $y$ -value  $y_j$ , which means if we recognise  $L_j$  as a popular place we will actually report the line segment  $x_p \times [y_j, y_{j+1}]$  as a popular place defining interval. To find out if the leaves  $L_l$  or  $L_r$  induce popular places we actually have to be a bit more careful and determine whether we have to extend the reported line segment to the lower end  $a$  or the upper end  $b$  of the query range, by checking how many intervals the leaf stabs, and in case this number is exactly  $k$  whether an interval starts or ends at  $L_l$  or  $L_r$ , respectively.

Next, we make the following observations: let  $\mathcal{I}_{p,\text{start}}$  be the popular place defining interval set that we have found for a start event  $p$  and let  $s$  be a popular place associated with  $\mathcal{I}_{p,\text{start}}$ . Let  $s'$  denote the position of the unit square  $s$  when we slightly shift it to the left. If  $s'$  is also a popular place, we will have recognised this popular place earlier on when we processed a start event to the left of the current sweep line position. So, finding the left boundary of the polygons  $\mathcal{H}(E)$  can simply be done by setting start flags whenever we find the set  $\mathcal{I}_{p,\text{start}}$  for a start event  $p$ : for each  $I \in \mathcal{I}_{p,\text{start}}$  we set the start flag  $x_p - 1 \times I$ , see Fig. 5.

However, we still have to determine the right boundary of the polygons  $\mathcal{H}(E)$ . We do this by setting end flags: for an end event  $q$  we also compute the set of popular place defining intervals. Now, let  $\mathcal{I}_{q,\text{end}}$  be the complement of this set in  $R_q$  extended with the popular place defining intervals whose stabbing number is exactly  $k$  ( $q$  is about to leave). For each  $I \in \mathcal{I}_{q,\text{end}}$  we set the end flag  $x_p \times I$ , see Fig. 5. Now, we simply have to connect the start and end flags accordingly to obtain the polygon set  $\mathcal{H}(E)$ .

**Theorem 3.** *Given a set  $E$  of  $n$  moving point objects in the plane the polygons  $\mathcal{H}(E)$  can be reported in  $O(\tau n \log \tau n + M \log \tau n)$  time using  $O(\tau n + M)$  space, where  $M$  is the number of all popular place defining intervals that we find throughout the algorithm.*

### 3 A Lower Bound in the Discrete Model

One of the first problems shown to have an  $\Omega(N \log N)$  lower bound was the MIN-GAP problem [4]. The decision version of the MIN-GAP problem also has an  $\Omega(N \log N)$  lower bound [12].

*Problem 1.* (MIN-GAP, decision version)

Given a vector  $x \in \mathbb{R}^N$  of reals and a positive real value  $\delta$ , is  $\min_{i \neq j} |x_i - x_j| > \delta$ ?

Now consider the decision version of the discrete popular place problem in one dimension for  $n$  entities,  $\tau$  time step and  $k = 2$ .

*Problem 2.* (POP-PLACE1, decision version)

Given a set  $Y$  of  $n$  vectors  $y \in \mathbb{R}^\tau$  of reals and a positive real value  $\delta$ , is there a popular place of side length  $\delta$  and  $k \geq 2$ ?

**Theorem 4.** *Problem 2 has an  $\Omega(\tau n \log \tau n)$  lower bound.*

*Proof.* Consider an algorithm  $\mathcal{A}(Y)$  that solves Problem 2 in  $T_{\mathcal{A}}(Y)$  time. We show how algorithm  $\mathcal{A}$  can be used to solve Problem 1.

Let  $1 \leq a \leq N$  be a fixed integer. We transform an instance of Problem 1 into an instance of Problem 2. Suppose we are given a vector  $x \in \mathbb{R}^N$  of reals and a positive real value  $\delta$  as input to Problem 1. Place the values in  $x$  in a matrix  $M$  with  $a$  columns and  $b = \lceil \frac{N}{a} \rceil$  rows, in any order. (In the case that  $\frac{N}{a}$  is not an integer, fill up the matrix with appropriate dummy values.)

Consider the columns in  $M$  as a set of  $a$  vectors of length  $b$ . This set of vectors can be interpreted as a set  $Y_1$  of  $n = a$  trajectories over  $\tau = b$  time steps in one dimension. Run algorithm  $\mathcal{A}$  with  $Y_1$  as input. If the algorithm  $\mathcal{A}$  returns ‘Yes’ then return ‘Yes’ as the answer to Problem 1. Otherwise, consider the rows in  $M$  as a set  $Y_2$  of  $n = b$  trajectories over  $\tau = a$  time steps. Run algorithm  $\mathcal{A}$  again, but this time with  $Y_2$  as input. If the algorithm  $\mathcal{A}$  returns ‘Yes’ then return ‘Yes’, otherwise return ‘No’ as the answer to Problem 1.

To prove the correctness of this transformation consider the pair of real numbers  $x_i, x_j$  in  $x$  with the smallest gap in  $x$ . If  $x_i$  and  $x_j$  are in different columns then algorithm  $\mathcal{A}(Y_1)$  will report ‘Yes’ if  $|x_i - x_j| \leq \delta$ . If  $x_i$  and  $x_j$  are in the same column they must belong to different rows, thus  $\mathcal{A}(Y_2)$  will report ‘Yes’ if  $|x_i - x_j| \leq \delta$ , otherwise ‘No’.

Thus, we can solve Problem 1 with algorithm  $\mathcal{A}$  in  $O(N) + T_{\mathcal{A}}(Y_1) + T_{\mathcal{A}}(Y_2)$  time. As there is a lower bound for this time of  $\Omega(N \log N)$ , this implies that  $\max\{T_{\mathcal{A}}(Y_1), T_{\mathcal{A}}(Y_2)\} = \Omega(N \log N) = \Omega(\tau n \log \tau n)$ , because  $\tau n = \Theta(N)$ .  $\square$

## 4 An Exact Algorithm in the Continuous Model

In this section we consider the continuous model and present an  $O(\tau^2 n^2)$  time algorithm using  $O(\tau n)$  space. Later in Section 5 we will argue that it is unlikely to find an asymptotically faster algorithm. Our algorithm takes as input a set  $E$  of  $n$  entities  $A_1, \dots, A_n$  moving in the plane over  $\tau$  time steps. The output of the algorithm will be a set  $\mathcal{H}(E)$  of polygons.  $\mathcal{H}(E)$  is the minimal point set in the plane such that any axis-aligned unit square whose centre lies in  $\mathcal{H}(E)$  intersects at least  $k$  trajectories of  $E$ . Note that these polygons are in general not rectilinear polygons.

We first show how to construct an arrangement  $\mathcal{A}$  of lines. The general idea is to sweep the arrangement  $\mathcal{A}$  and then building  $\mathcal{H}(E)$ . For ease of presentation, we will initially describe an algorithm using a standard sweep-line technique with running time  $O(\tau^2 n^2 \log \tau n)$  using  $O(\tau^2 n^2)$  space. This sweep-line algorithm identifies the edges that contribute to the polygons in  $\mathcal{H}(E)$ . In a second sweep over these edges, we will construct the polygons in  $\mathcal{H}(E)$ . Note that the presented algorithms work for inputs with degeneracies and are easy to implement. We then observe that our methods do not require a sweep by a straight line. Hence, we can use a topological plane sweep introduced by Edelsbrunner and Guibas [8] to improve the running time to  $O(\tau^2 n^2)$  and the used space to  $O(\tau n)$ .

### Constructing the Line Arrangement

Recall that we use  $A$  to denote both an entity and its trajectory. Also recall that a trajectory is a polygonal path described by  $\tau$  points, and that two consecutive points are connected by a straight-line segment  $s$ . Consider the following polygon construction: for a trajectory  $A$  sweep an axis-aligned unit square  $\sigma$  along the trajectory such that its centre moves on  $A$  as shown in Figure 6(a). The region swept by  $\sigma$  induces a polygon which we denote by  $P(A)$ , see Figure 6(b). Note that any axis-aligned unit square having its centre within  $P(A)$ , will intersect  $A$ . Thus, we can restrict ourselves to consider the centre locations for the popular-place defining squares.

Consider a set  $W$  of polygons and a point  $q$  in the plane. The *depth* of  $q$  with respect to  $W$  is the number of polygons in  $W$  intersecting  $q$ . This definition allows us to describe  $\mathcal{H}(E)$  as follows:

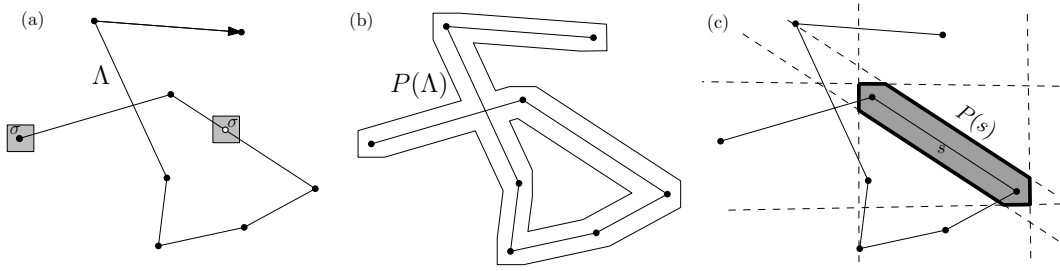


Fig. 6: (a) a trajectory  $\Lambda$  and the square  $\sigma$  that sweeps along  $\Lambda$ , (b) the polygon  $P(\Lambda)$  obtained from the sweep, (c) the polygon  $P(s)$  and the lines  $l_i$  generated by  $P(s)$  (dashed)

**Observation 1**  $\mathcal{H}(E)$  consists of the set of points having depth at least  $k$  with respect to  $\{P(\Lambda_1), \dots, P(\Lambda_n)\}$ .

However, we do not explicitly store the polygons  $P(\Lambda)$  for each  $\Lambda$ , because  $P(\Lambda)$  might have  $O(\tau^2)$  holes, which in turn results in  $O(\tau^2)$  storage space for each  $P(\Lambda)$ . Instead, the following approach will result in  $\Theta(\tau)$  line-segments per trajectory, which guarantees linear space usage and fewer event points during the sweep. For each segment  $s$  of  $\Lambda$  consider the region swept along  $s$  by the unit-square  $\sigma$ . This region is a polygon  $P(s)$  with at most six edges, see Figure 6(c). When considering the two points that specify an edge  $e_i$  of  $P(s)$ , we call the point with the smaller  $x$ -coordinate (or smaller  $y$ -coordinate, in case the  $x$ -coordinates are equal) the *start point* of  $e_i$  and the other one the *end point* of  $e_i$ . For each edge  $e_i$  of  $P(s)$ , we construct an infinite line  $l_i$  that contains  $e_i$ . For each line  $l_i$ , we store the start and end point of  $e_i$  on  $l_i$ , and to which side  $s$  lies, and  $\Lambda$ . We refer to  $e_i$  as a *visible edge* and to the rest of  $l_i$  as the *invisible line*. The set of all lines constructed as above yields the line arrangement  $\mathcal{A}$ , which we will sweep. Note that  $\mathcal{A}$  contains  $O(\tau n)$  lines.

### The First Sweep

The algorithm will sweep the arrangement  $\mathcal{A}$  from left to right using a vertical sweep line  $\ell$ . The status structure of the sweep line is stored in a list  $S$  of size  $O(\tau n)$ . For convenience we sometimes use  $S$  as if it was an ordered string. It contains the current intersections between  $\ell$  and the visible edges in  $\mathcal{A}$  ordered along  $\ell$  from top to bottom. Initially,  $S$  is empty. For each intersection between  $\ell$  and a visible edge  $e$ , we store a *bracket*  $br$  in  $S$  with pointers between  $br$ , the corresponding edge  $e$  and the line corresponding to  $e$ . A bracket is formally a tuple  $\langle i, type, level, depth \rangle$ , where  $i$  is the entity number corresponding to the edge;  $type$  is either *open* or *closed* depending on whether we are entering or exiting the polygon  $P(s)$  as we go down  $\ell$ . The brackets will always come in open-closed pairs, because  $P(s)$ , for a segment  $s$ , is a convex region. Note that we can consider them as matching pairs of brackets for the same entity, and that the matching brackets do not have to come from the same polygon  $P(s)$ . For example in Figure 7, if going down  $\ell$  then we enter two polygons belonging to entity  $A_1$  and then exit both of them. We get the following sequence of brackets:  $(())$ . We say that the first and last are one matching pair (event though they do not correspond to the same polygon) with nesting level (*level*) equal to 1, and the second and third bracket are a different pair with nesting level 2. The *depth* value of a bracket is the depth with respect to  $\{P(\Lambda_1), \dots, P(\Lambda_n)\}$  of points immediately after this bracket until the next bracket as we go down  $\ell$ . Note that *depth* only counts each entity once, so a point that stabs many polygons belonging to the same trajectory will only get a contribution of 1 to *depth* from that entity. Therefore in  $(())$  the second bracket will have *depth* = 1 since the middle region only stabs one unique entity. An example is shown in Figure 7.

An *event* of the sweep is the intersection of exactly two lines of the arrangement  $\mathcal{A}$ . These events happen at the vertices of the arrangement  $\mathcal{A}$ , which we call *event points* from now on. By our construction it will happen that three or more lines of  $\mathcal{A}$  intersect in one point. Hence, multiple



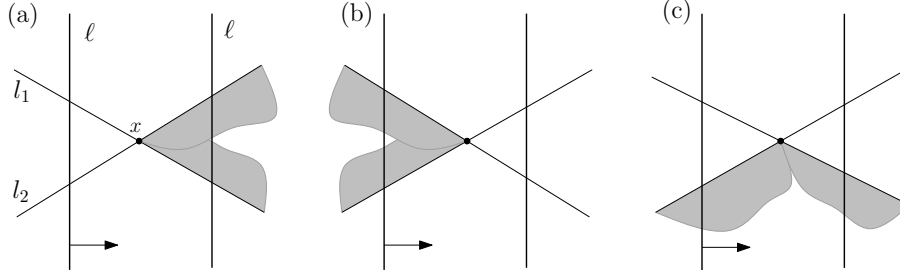


Fig. 8: Cases at an event point  $x$  during the line-arrangement sweep.

- (c)  $l_1$  is of type (I),  $l_2$  is of type (II). If  $P(s_1) = P(s_2)$  then we need to change a pointer. More specifically, let  $br$  be the bracket in  $S''$  corresponding to the visible segment of  $l_1$ . We have to change the pointer of  $br$  that points to  $l_2$  so that it now points to  $l_1$ . The symmetric opposite case can be handled in a similar way.
- (\*) For all other cases, we do not make any changes.

Note, that there are only  $O(\tau n)$  events of the first three cases in total.

- We sort the brackets in  $S''$  in non-increasing order according to the slope of their corresponding lines. This can be done in  $O(|L| \log |L|)$  time.
- Recalculate all the *level* values from scratch for all the brackets in  $S''$ . This can be done for each entity  $A_i$  independently. Let  $br'$  and  $br''$  be the first brackets that correspond to  $A_i$  in  $S'$  and  $S''$ , respectively. Note that  $br'$  and  $br''$  must have the same *level* value. All other *level* values in  $S''$  can be computed by traversing  $S''$  starting at  $br''$ . Whenever we encounter an open or closed bracket corresponding to  $A_i$ , we set its *level* to be the level of the previous bracket plus or minus one, respectively. This costs  $O(|L|)$  time per entity. Hence, in total this can be done in  $O(|L|^2)$  time.
- Recalculate all the *depth* values from scratch for all the brackets in  $S''$ . Having the *depth* values of the first brackets and all the *level* values, we can calculate the *depth* values of all brackets in a way that is similar to the previous step. Also this can be done in  $O(|L|^2)$  time in total.
- Replace  $S'$  by  $S''$  in  $S$ .

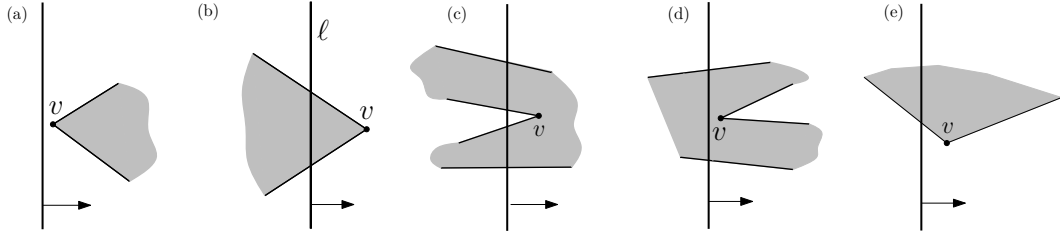
**Lemma 4.** *The status of the sweep line can be maintained during the sweep in total time  $O(\tau^2 n^2)$  and  $O(\tau n)$  space.*

*Proof.* The above procedure describes how to process an event point  $x$  in which  $|L|$  lines intersect. Note that there are  $\Theta(|L|^2)$  events happening at  $x$ , and all of them can be processed in  $O(|L|^2)$  time and  $O(|L|)$  space. Hence, each single event can be processed in  $O(1)$  amortised time. Since each trajectory induces  $6\tau$  lines in  $\mathcal{A}$  and there are  $O(\tau^2 n^2)$  events in total, the lemma follows.  $\square$

### Constructing the Output

In the above we did not describe what our algorithm outputs. In this section, we will make up for this, by adding a few steps to the previous sweep. Recall that the output of our algorithm will be a set  $\mathcal{H}(E)$  which consists of all points having depth at least  $k$  with respect to  $\{P(A_1), \dots, P(A_n)\}$ .

Recall that by definition any edge of  $\mathcal{A}$  has a start point that is to the left of its end point. Hence, for each edge  $e$  and corresponding bracket  $br$  we also have two unique faces of  $\mathcal{A}$ , one below  $e$ , the other above  $e$ . The *depth* value of  $br$  gives us the depth of (the points in) the face that is below  $br$ , while the *depth* value of the bracket in  $S$  that is just before  $br$  gives us the depth of (the points in) the face above  $br$ . Whenever we have a bracket  $br$  where the two corresponding faces have depth  $k - 1$  and  $k$  then we call that bracket a *boundary bracket*. A boundary bracket

Fig. 9: The cases that may occur while building  $\mathcal{H}(E)$ .

$br$  corresponds to a *boundary edge*, which is a part of the edge  $e$  corresponding to  $br$  that is an edge of a polygon in  $\mathcal{H}(E)$ . The start and end points of boundary edges are the event points of the sweep where a bracket becomes a boundary bracket, or where a boundary bracket ceases to be a boundary bracket, or where a boundary bracket is inserted, or where a boundary bracket is deleted.

To identify the set  $B$  of all boundary edges, we extend the procedure of the previous section, which handles all events at a given event point. More specifically, we add the following steps to that procedure just before we replace  $S'$  by  $S''$ .

- For all brackets  $br \in S'' \setminus S'$ :
  - If  $br$  is a boundary bracket, then add a new boundary edge  $b$  to  $B$ . The start point of  $b$  will be the current event point. Associate  $b$  with the boundary bracket  $br$ .
- For all brackets  $br \in S' \setminus S''$ :
  - If  $br$  is a boundary bracket, then the current event point is the end point of the boundary edge associated with  $br$ .
- For all brackets  $br \in S' \cap S''$ :
  - If  $br$  is a boundary bracket in  $S''$  but not in  $S'$ , then add a new boundary edge  $b$  to  $B$ . The start point of  $b$  will be the current event point. Associate  $b$  with the boundary bracket  $br$ .
  - If  $br$  is a boundary bracket in  $S'$  but not in  $S''$ , then the current event point is the end point of the boundary edge associated with  $br$ .
  - If  $br$  is a boundary bracket in  $S'$  and in  $S''$ , but  $br$  corresponds to different lines in  $S'$  and  $S''$ , then  $x$  is a start and end point of boundary edges. The current event point  $x$  is the end point of the boundary edge associated with  $br$ . We also add a new boundary edge  $b$  to  $B$  with start point the current event point  $x$ . Associate  $b$  with the boundary bracket  $br$  in  $S''$ .

Having the information about all the boundary edges allows us to build  $\mathcal{H}(E)$  by performing a second sweep traversing the set  $B$  of boundary edges with a vertical sweep line  $\ell$ . Note that no two boundary edges can intersect. An event point  $x$  of this second sweep will either be (a) a start vertex, (b) an end vertex, (c) a merge vertex, (d) a split vertex or (e) a regular vertex of a polygon in  $\mathcal{H}(E)$ , see Figure 9. For an event point  $x$  it is not hard to see that we can decide which kind of polygon vertex it induces in constant time by looking at the information that the two boundary edges emanating from  $x$  yield. Hence, the sweep and the construction of  $\mathcal{H}(E)$  can be done in  $O(|\mathcal{H}(E)|)$  time, where  $|\mathcal{H}(E)|$  denotes the complexity of  $\mathcal{H}(E)$ . Note that, by the first sweep, we get the boundary edges already in order.

Recall that the first sweep processed  $O(\tau^2 n^2)$  events, so together we have:

**Theorem 5.** *Given a set  $E$  of  $n$  entities moving in the plane over  $\tau$  time steps, the set  $\mathcal{H}(E)$  can be computed in  $O(\tau^2 n^2 \log \tau n)$  time using  $O(\tau^2 n^2)$  space.*

## Using a Topological Sweep

If we examine the above sweep-line algorithms we can observe that there is no need to process the points strictly from left to right. As long as a well-defined sweep line is maintained the order in which the event points are processed is not important. Also, we only need to keep the events and event points in memory that correspond to the current sweep line. This observation suggests the use of a topological sweep line introduced by Edelsbrunner and Guibas [8]. Recall that our arrangement contains degenerate cases, such as identical/parallel lines and multiple lines intersecting in the same point. Also note that our method processes all events that happen at the same event point at once. Hence, we cannot apply the results from [8] directly. We need a topological sweep approach that correctly processes all degeneracies; especially, the approach should recognise and handle at once all events that happen at the same event point. Exactly this can be done with an approach suggested by Rafalin et al. [22], which is an extension of [8]. As a result the above bounds can be improved.

**Theorem 6.** *Given a set  $E$  of  $n$  entities moving in the plane over  $\tau$  time steps, the set  $\mathcal{H}(P)$  can be computed in  $O(\tau^2 n^2)$  time using  $O(\tau n + |\mathcal{H}(P)|)$  space.*

## 5 Hardness in the Continuous Model

We argue that it is likely that every algorithm in the continuous model of the problem requires  $\Omega(n^2 \tau^2)$  time in the worst case. We will present a transformation from the problem 3-SUM2 to a special case of our problem.

*Problem 3. (3-SUM2)*

Given three sets  $A$ ,  $B$  and  $C$  of integers with  $|A| + |B| + |C| = N$ , are there  $a \in A$ ,  $b \in B$  and  $c \in C$  with  $a + b = c$ ?

The 3-SUM2 problem is closely related to the classic 3-SUM problem and has been proven to be 3-SUM-hard [11]. This means that it is at least as hard as 3-SUM (with input size  $N$ ) for which no subquadratic time algorithm has been found yet, which is an indication for an inherent hardness of the problems. For a weak model of computation a lower bound of  $\Omega(N^2)$  for those problems exists [9]. The problem we will prove to be 3-SUM-hard is the following version of the popular places problem.

*Problem 4. (POP-PLACE2)*

Given the trajectories of  $n$  entities over  $\tau$  time steps, is there a popular place of at least three entities with side length zero?

Let  $(A, B, C)$  be a 3-SUM2 instance and let  $\tau$  be a fixed positive integer with  $1 \leq \tau \leq N$ . The transformation from 3-SUM2 to the POP-PLACE2 problem is as follows. For each integer  $s$  of the input, we create a line  $\ell_s : y = d_1 x + d_2$ , where  $d_1$  and  $d_2$  depend on  $s$  and which set  $s$  belongs to. We sort the set  $A = \{a_1, \dots, a_{|A|}\}$  such that it is indexed in increasing order of its elements. We then create a set  $L^A$  of horizontal lines  $L^A := \{\ell_a^h : y = a \mid a \in A\}$ . We do the same for the sets  $B$  and  $C$  and create a set of vertical lines  $L^B := \{\ell_b^v : x = b \mid b \in B\}$ , and a set of diagonal lines  $L^C := \{\ell_c^d : y = c - x \mid c \in C\}$ . See Figure 10(a), for an example of three such lines that intersect in one point.

**Observation 2** *There exist  $a \in A$ ,  $b \in B$  and  $c \in C$  with  $a + b = c$ , iff the three lines  $\ell_a^h$ ,  $\ell_b^v$  and  $\ell_c^d$  intersect in one point.*

Now, we transform these lines into line-segments. We compute an axis-aligned bounding box  $BB$  whose interior contains all intersections between all lines, as is illustrated in Figure 10(b). For each line, we then cut off everything outside  $BB$ , resulting in  $N$  line segments. For the sake of simplicity, we consider the sets  $L^A$ ,  $L^B$  and  $L^C$  as sets of these line segments from now on. Note that no two line segments belonging to the same set can intersect.

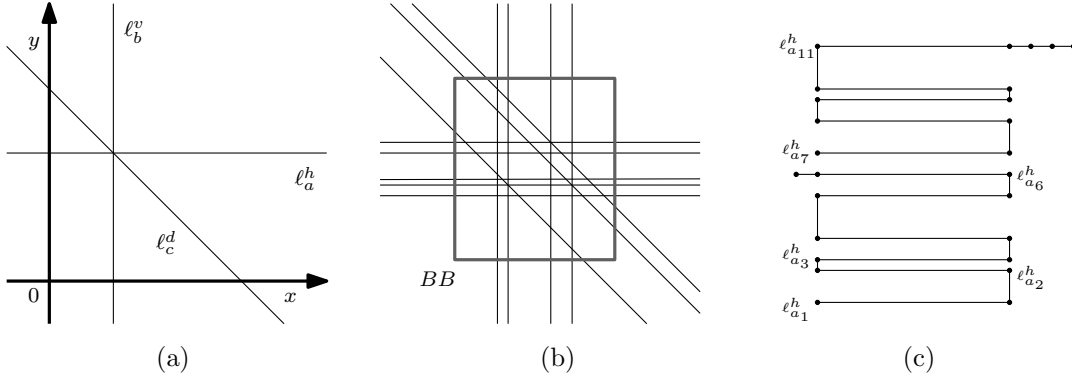


Fig. 10: Examples illustrating the transformation for the lower bound: (a) three lines  $\ell_a^h \in L^A$ ,  $\ell_b^v \in L^B$  and  $\ell_c^d \in L^C$ , which intersect in one point; (b) the bounding box  $BB$  containing all intersections between all lines; (c) constructing trajectories of length  $\tau = 12$  using the line segments of set  $L^A$  with  $|L^A| = |A| = 11$ .

As a last step we construct trajectories using the line segments. We will only describe this for the set of horizontal line segments  $L^A = \{\ell_{a_1}^h, \dots, \ell_{a_{|A|}}^h\}$ , because for the other sets it is analogous. To create a trajectory of length  $\tau$ , we start with  $\ell_{a_1}^h$  and connect it to  $\ell_{a_2}^h$  with a vertical line segment joining the right endpoints of  $\ell_{a_1}^h$  and  $\ell_{a_2}^h$ . We then join the left endpoints of  $\ell_{a_2}^h$  and  $\ell_{a_3}^h$  by another vertical connector line segment. We continue to add line segments  $\ell_{a_i}^h$  and vertical connector segments either on the left or right side until we obtain a trajectory of length  $\tau$  or  $\tau - 1$ . If  $\tau$  is even, we extend the current trajectory by adding a unit-length horizontal dummy line segment. With the remaining line segments in  $L^A$ , we create more trajectories, and we continue the process until all line segments of  $L^A$  are used in trajectories. If, for the last trajectory that was created, there are not enough line segments in  $L^A$  for it to have length  $\tau$ , we add unit-length horizontal dummy segments to it to achieve length  $\tau$ . An example is shown in Figure 10(c). Note that none of the added horizontal dummy segments and none of vertical connector segments intersects with any other line segment. The line segments of the sets  $L^B$  and  $L^C$  are used to create trajectories of length  $\tau$  in a similar way.

The result of this transformation is a set  $T$  of  $n$  trajectories over  $\tau$  time steps. From the above, the following lemma becomes evident.

**Lemma 5.** *There exist  $a \in A$ ,  $b \in B$  and  $c \in C$  with  $a + b = c$ , iff there exists a popular place in  $T$  of at least three entities with side length zero, i.e.  $3\text{-SUM2} \leq \text{POP-PLACE2}$ .*

Note that in our trajectories  $T$ , we have  $N$  segments originating from  $A$ ,  $B$  and  $C$ , at most  $N$  connector segments and at most  $3\tau \leq 3N$  dummy segments. Hence, the size of  $T$  is  $n\tau = \Theta(N)$ . As the transformation holds for any  $\tau$  and can be done in  $O(N \log N)$  time, we can conclude with the following theorem.

**Theorem 7.** *Let  $T$  be a set of  $n$  trajectories over  $\tau$  time-steps, for any  $\tau \geq 1$ . There exists no  $o(n^2\tau^2)$  time algorithm to decide  $\text{POP-PLACE2}$  with input  $T$ , unless there exists an  $o(N^2)$  time algorithm to decide  $3\text{-SUM2}$  for an input of total size  $N$ .*

## Acknowledgements

We would like to thank Hans Bodlaender, Jyrki Katajainen, Damian Merrick and Anh Pham for very useful discussions.

## References

1. Wildlife tracking projects with GPS GSM collars.  
<http://www.environmental-studies.de/projects/projects.html>, 2006.

2. G. Al-Naymat, S. Chawla, and J. Gudmundsson. Dimensionality reduction for long duration and complex spatio-temporal queries. In *Proceedings of the 22nd ACM Symposium on Applied Computing*, pages 393–397. ACM, 2007.
3. M. Andersson, J. Gudmundsson, P. Laube, and T. Wolle. Reporting leadership patterns among trajectories. In *Proceedings of the 22nd ACM Symposium on Applied Computing*, pages 3–7. ACM, 2007.
4. M. Ben-Or. Lower bounds for algebraic computation trees. In *15th Annual ACM Symposium on Theory of Computation*, pages 80–86, 1983.
5. M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. In *Proceedings of the 14th European Symposium on Algorithms (ESA 2006)*, volume 4168 of *Lecture Notes in Computer Science*, pages 660–671. Springer, 2006.
6. H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15(3):211–228, 2006.
7. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 2nd edition, 2001.
8. H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38:165–194, 1989.
9. J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete & Computational Geometry*, 13:41–57, 1995.
10. A. U. Frank. Socio-Economic Units: Their Life and Motion. In A. U. Frank, J. Raper, and J. P. Cheylan, editors, *Life and motion of socio-economic units*, volume 8 of *GISDATA*, pages 21–34. Taylor & Francis, London, 2001.
11. A. Gajentaan and M. H. Overmars.  $n^2$ -hard problems in computational geometry. Technical Report 1993-15, Department of Computer Science, Utrecht University, The Netherlands, 1993.
12. J. Gudmundsson, T. Husfeldt, and C. Levkopoulos. *Information Processing Letters*, 81(3):137–141, 2002.
13. J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. Compressing spatio-temporal trajectories. Manuscript, July 2007.
14. J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in trajectory data. In *Proceedings of the 14th ACM Symposium on Advances in GIS*, pages 35–42, 2006.
15. J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal sets. *GeoInformatica*, 11(2):195–215, 2007.
16. P. Gupta, R. Janardan, and M. Smid. *Handbook of Data Structures and Applications*, chapter Computational geometry: generalized intersection searching, pages 64–1 – 64–17. Chapman & Hall/CRC, 2005.
17. R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers, 2005.
18. M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Indexing spatio-temporal archives. *The VLDB Journal*, 15(2):143–164, 2006.
19. P. Laube, S. Imfeld, and R. Weibel. Discovering relative motion patterns in groups of moving point objects. *International Journal of Geographical Information Science*, 19(6):639–668, 2005.
20. P. Laube, M. van Kreveld, and S. Imfeld. Finding REMO – detecting relative motion patterns in geospatial lifelines. In P. F. Fisher, editor, *Developments in Spatial Data Handling: Proceedings of the 11th International Symposium on Spatial Data Handling*, pages 201–214, Berlin, 2004. Springer.
21. N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *Proceedings of the 10th ACM SIGKDD International Conference On Knowledge Discovery and Data Mining*, pages 236–245. ACM, 2004.
22. E. Rafalin, D. Souvaine, and I. Streinu. Topological sweep in degenerate cases. In *Proceedings of the 4th international workshop on Algorithm Engineering and Experiments, ALENEX 02*, volume 2409 of *Lecture Notes in Computer Science*, pages 155–156, 2002.
23. S. Sältenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 331–342, 2000.
24. F. Verhein and S. Chawla. Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications (DASFAA)*, volume 3882 of *Lecture Notes in Computer Science*, pages 187–201. Springer, 2006.