Finding Popular Places

Marc Benkert[1], Bojan Djordjevic[2], Joachim Gudmundsson[2], and Thomas Wolle[2]

[1] Department of Computer Science, Karlsruhe University, Germany.
mbenkert@ira.uka.de
[2] 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, 7, 11]. 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 transport 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 [13, 14]) among geospatial lifelines. The input is a set E of n moving point objects $\Lambda_1, \ldots, \Lambda_n$ whose locations are known at τ consecutive time-steps $t_1, \ldots, t_\tau$, 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 Λ 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 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

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

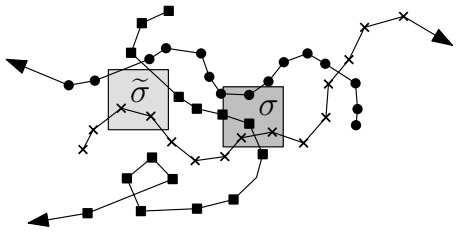

Fig. 1: An example where three entities Λ_1, Λ_2 and Λ_3 are traced during 16 time steps. For $k = 3$ the square $\widetilde{\sigma}$ is a popular place only in the continuous model. While σ is a popular place for $k = 3$ in both the discrete and continuous model.

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 [5, 8] 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 σ of side length r is a* popular place *in the* discrete model *if σ contains input points from at least k different entities. In the* continuous model *σ is a* popular place *if it is intersected by polylines from at least k different entities.*

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 Săltenis et al. [16] and Hadjieleftheriou et al. [12]. From a data mining perspective Verhein and Chawla [17] 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. [15] studied periodic patterns, e.g. yearly migration patterns or daily commuting patterns. There have been several papers considering the problem of detecting flock patterns and leadership patterns [2–4, 9].

Precursory to this work Laube et al. [13, 14] 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. The aim is to find or forecast a popular place (assuming the entities do not change their direction).

As mentioned in earlier work [4, 9, 10], 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 Section 2 we present an algorithm for the discrete model, followed by an $O(\tau^2 n^2)$ time algorithm in the continuous model. And in Section 4, we present lower bounds and hardness results. We omitted proofs and some details due to space constraints in this extended abstract.

2 A fast algorithm in the discrete model

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

The idea of our algorithm is to use a vertical sweep line ℓ sweeping the points from left to right. Together with the sweep line we sweep a vertical strip str_ℓ of width 1 whose right boundary is ℓ. Each of the τn input points induces two event points, one when the point enters str_ℓ and one when the point leaves str_ℓ. We refer to these types as *start* and *end* events.

For a start event, say that an input point p belonging to entity Λ enters str_ℓ, we update our data structures and check for the largest popular place located in str_ℓ that is visited by Λ. 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.

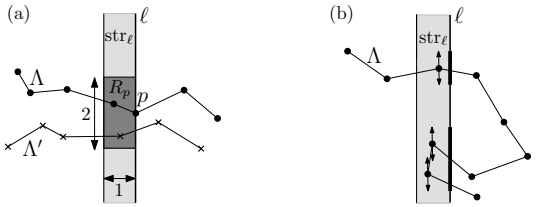

Fig. 2: (a) Finding the popular places in R_p visited by Λ. (b) The set I_Λ is indicated by the bold, solid line segments on ℓ.

For an end event, say a point p belonging to Λ is about to leave str_ℓ, we simply remove it from the current data structures.

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 then we show how we can merge this information into one tree T_{int} that can be queried and updated in $O(\log \tau n)$ time.

One structure for each entity. During the sweep, we maintain a set of disjoint y-intervals I_Λ, for each entity Λ, such that I_Λ contains exactly the y-intervals for which a square s in str_ℓ with centre in an interval in I_Λ contains a point of Λ, 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.

We will maintain two trees B_Λ and T_Λ for each entity Λ. The tree B_Λ is a balanced binary search tree on the points of Λ currently within str_ℓ and ordered with respect to their y-coordinates. The tree T_Λ will store the set I_Λ of intervals w.r.t. the current position of ℓ. The leaves of T_Λ store the endpoints of the intervals in I_Λ ordered on their y-coordinates. Each leaf also contains a pointer to the leaf in T_Λ 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 $p_\Lambda = (x, y)$. Let $I = [y - 1/2, y + 1/2]$ and note that any unit square within str_ℓ with centre in I will contain p_Λ. The point is inserted into B_Λ and then a range query is performed in T_Λ that reports the intervals of I_Λ intersecting I. Since the intervals in I_Λ are disjoint and have length at least one, I may intersect at most two intervals in I_Λ. 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_Λ. If I intersects one interval I_1 then I_1 is deleted and the interval $I \cup I_1$ is inserted, and if two intervals I_1 and I_2 are intersected then they are deleted and $I \cup I_1 \cup I_2$ is inserted.

In the case when str_ℓ is about to process an end event $p_\Lambda = (x, y)$, 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_Λ in B_Λ, and delete p_Λ. 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_Λ 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_Λ.
3. If $|y - y_1| > 1$ and $|y - y_2| \leq 1$, or vice versa, then I' is deleted from T_Λ, and the interval $I' \setminus [y - 1/2, y_2 - 1/2)$ is inserted.

We denote the set of all trees T_Λ and B_Λ 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.*

Maintaining the status of the sweep. We store all intervals that are currently in str_ℓ in a balanced binary tree T_{int}. We use T_{int} to perform the maximum popular place query for R_p. Let $\mathcal{I} = \bigcup_\Lambda I_\Lambda$. We assume that all start and end points of intervals in $\mathcal{I}$ are pairwise disjoint. The leaf set of T_{int} corresponds to the set of start and end points in $\mathcal{I}$ ordered w.r.t. their y-coordinates.

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

We now show how we can store appropriate information in T_{int} in order to perform maximum popular place queries.

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

$$\begin{aligned} \mathrm{sum}(V) &:= \mathrm{sum}(V_{left}) + \mathrm{sum}(V_{right}), \quad \text{and} \\ \max_{\mathrm{pre}}(V) &:= \max\{\max_{\mathrm{pre}}(V_{left}), \mathrm{sum}(V_{left}) + \max_{\mathrm{pre}}(V_{right})\} \end{aligned}$$

Let $L_1(V), \ldots, L_m(V)$ 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 $\mathrm{sum}(V) = \sum_{j=1}^{m} \mathrm{sum}(L_j(V))$ and $\max_{\mathrm{pre}}(V) = \max_{1 \le i \le m} \sum_{j=1}^{i} \mathrm{sum}(L_j(V))$.

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

Lemma 2. *Throughout the sweep, the values* sum *and* $\max_{\mathrm{pre}}$ *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 $p = p_\Lambda$, we first perform the required updates in B_Λ and T_Λ, update T_{int} accordingly

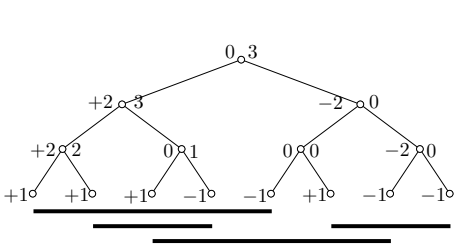

Fig. 3: The values sum (left) and $\max_{\text{pre}}$ (right) for the depicted tree T_{int}.

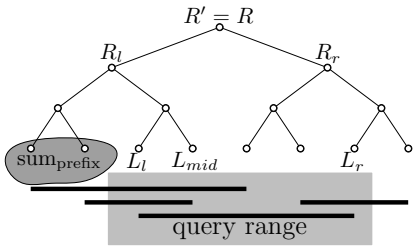

Fig. 4: Querying T_{int} for the maximum stabbing number $\max_{l,r}$.

and then query T_{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 $L_1, L_2, \ldots$ be the set of leaves in T_{int} ordered from left to right. Each of the leaves in T_{int} is associated with the y-value of the stored start or end point. By performing two searches in T_{int} we can find the leftmost leaf L_l in T_{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_{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 $\max_{l,r}$. We have that $\max_{l,r} = \max_{l \le m \le r}\{\sum_{j=1}^{m} \text{sum}(L_j)\}$. We claim that $\max_{l,r}$ can be calculated by walking along the search paths from L_l and L_r, respectively, to the root R of T_{int}. We sketch how this is done. Note that the sketch comprises that a leaf with stabbing number $\max_{l,r}$ 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 $l \le mid < r$, 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 $\text{sum}(V_{left})$ 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 $\max_l = \max_{l \le m \le mid} \sum_{j=l}^{m} \text{sum}(L_j)$ and $\max_r = \max_{mid+1 \le m \le r} \sum_{j=mid+1}^{m} \text{sum}(L_j)$. Once we know $\max_l$ and $\max_r$, the maximum stabbing number can be computed by $\max_{l,r} = \max\{\text{sum}_{\text{prefix}} + \max_l,\ \text{sum}(R'_{left}) + \max_r\}$. 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'_{left}. Initially, we set $\max_l := \max_{\text{pre}}(L_l)$, and a helper variable $\text{sum}_{\text{seen}} := \text{sum}(L_l)$. Let $L_l = V^0, V^1, \ldots, R'_{left}$ be the sequence of nodes that are traversed when walking from L_l to R'_{left} in T_{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. $\max_l := \max\{\max_l,\ \text{sum}_{\text{seen}} + \max_{\text{pre}}(V^i_{right})\}$ and update sum_{seen} to $\text{sum}_{\text{seen}} := \text{sum}_{\text{seen}} + \text{sum}(V^i_{right})$. 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'_{right} to L_r, let $R'_{right} = V^0, V^1, \ldots, L_r$ 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. $\max_r := \max\{\max_r, \text{sum}_{\text{seen}} + \max_{\text{pre}}(V_{left}^{i-1})\}$ and update sum_{seen} to $\text{sum}_{\text{seen}} := \text{sum}_{\text{seen}} + \text{sum}(V_{left}^{i-1})$. Arriving at L_r we get $\max_r$ by the final update $\max_r = \max\{\max_r, \text{sum}_{\text{seen}} + \max_{\text{pre}}(L_r)\}$. Now, we are ready to summarise the results obtained in this section.

Lemma 3. *Given T_{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.*

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.*

Approximating and reporting all popular places. For a start event $p = p_\Lambda$ we have seen how we can detect a unit square in R_p that contains the maximum number $k_{\max}$ of different entities in the discrete model among all unit squares contained in R_p. If we now find that $k_{\max} \geq k$ 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 $\mathcal{H}(E)$. However, we can extend the above technique in order to show the following theorem.

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 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 we will argue that it is unlikely to find an asymptotically faster algorithm. Our algorithm takes as input a set E of n entities $\Lambda_1, \ldots, \Lambda_n$ moving in the plane over τ 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 [6] 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 Λ to denote both an entity and its trajectory. Also recall that a trajectory is a polygonal path described by τ points, and that two consecutive points are connected by a straight-line segment s. Consider the following polygon construction: for a trajectory Λ sweep an axis-aligned unit square σ along the trajectory such that its centre moves on Λ as shown in Fig. 5(a). The region swept by σ induces a polygon which we denote by $P(\Lambda)$, see Fig. 5(b). Now 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:

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

However, we do not explicitly store the polygons $P(\Lambda)$ for each Λ. Instead, the following approach will be used. For each segment s of Λ consider the region swept along s by the unit-square σ. This region is a polygon $P(s)$ with at most six edges, see Fig. 5(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 Λ. 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 ℓ. 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 an ordered string. It contains the current intersections between ℓ and the visible edges in $\mathcal{A}$ ordered along ℓ from top to bottom. Initially, S is empty. For each intersection between ℓ 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

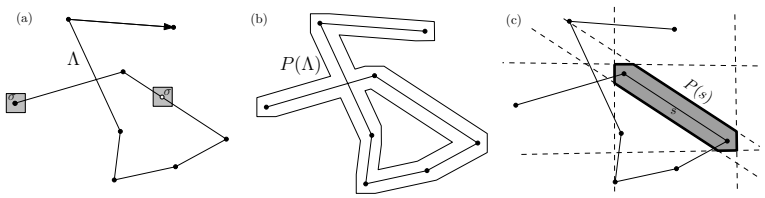

Fig. 5: (a) the square σ that sweeps along a trajectory Λ, (b) the polygon $P(\Lambda)$ obtained from the sweep, (c) the polygon $P(s)$ and the lines l_i generated by $P(s)$ (dashed)

exiting the polygon $P(s)$ as we go down ℓ. 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 $P(s)$. The *depth* value of a bracket is the depth with respect to $\{P(\Lambda_1), \ldots, P(\Lambda_n)\}$ of points immediately after this bracket until the next bracket as we go down ℓ. Note that *depth* only counts each entity once.

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 events can occur at a single event point. The moment at which the sweep line reaches an event point x the status of the sweep line is updated according to all events that happen at that event point. All events and event points are computed, sorted and stored beforehand. In the following, we categorise a line l_i that passes through an event point x into: (I) 'x coincides with the end point of e_i', (II) 'x coincides with the start point of e_i' and (III) 'other'. During the sweep, we process all event points in turn and for each event point x we will do the following.

- Let L be the set of lines of $\mathcal{A}$ that go through x. We have $\Theta(|L|^2)$ events.
- Let S' be the substring of S that contains all the brackets that correspond to lines that go through x.
- Let S'' be a new string that contains copies of exactly those brackets contained in S'. From now on, we will modify (brackets of) S''.
- For all pairs of lines l_1 and l_2 in L, let l_1 and l_2 be associated with an edge of $P(s_1)$ and $P(s_2)$ for some segments s_1 and s_2, respectively. We distinguish the following cases, which are illustrated in Fig. 6:
  - (a) Both l_1 and l_2 are of category (II). If $P(s_1) = P(s_2)$ then we encountered a new polygon $P(s)$ and we insert a pair of brackets (anywhere) into S''.
  - (b) Both l_1 and l_2 are of category (I). If $P(s_1) = P(s_2)$ then we finish sweeping a polygon $P(s)$ and delete the corresponding brackets in S''.
  - (c) l_1 is of category (I), l_2 is of category (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.
  - (*) For all other cases, we do not make any changes.

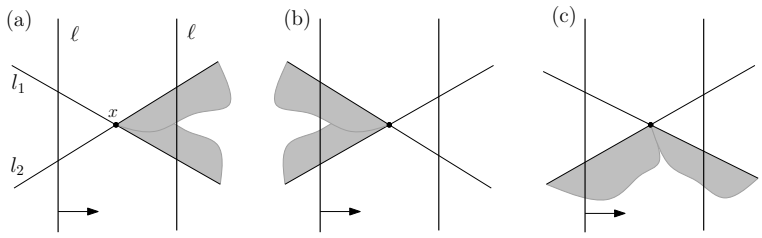

Fig. 6: Cases at an event point x during the line-arrangement sweep.

- We sort the brackets in S'' in non-increasing order according to the slope of their corresponding lines.
- Recalculate all the *level* and *depth* values from scratch for all the brackets in S''. This can be done by looping over S'' using the information in S'.
- Replace S' by S'' in S.

All this can be done in time linear in the number of events at x.

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.*

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 $\{P(\Lambda_1), \ldots, P(\Lambda_n)\}$.

Whenever we have a bracket br where the two faces above and below the edge e corresponding to br have depth $k-1$ and k then we call that bracket a *boundary bracket*. A boundary bracket br corresponds to a *boundary edge*, which is a part of the edge e 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. 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, which needs to be handled as above.

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 ℓ. 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 $\mathcal{H}(E)$. Recall that the first sweep processed $O(\tau^2 n^2)$ events, so together we have:

Theorem 4. *Given a set E of n entities moving in the plane over τ 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. 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 [6]. As a result the above bounds can be improved.

Theorem 5. *Given a set E of n entities moving in the plane over τ 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.*

4 Lower Bounds and Hardness Results

We present a lower bound for the popular places problem in the discrete model, and we argue that the popular places problem in the continuous model is at least as hard as 3-SUM, i.e. it is likely that every algorithm in the continuous model of the problem requires quadratic time in the worst case.

Theorem 6. *Let T be a set of n trajectories over τ time-steps, for any $\tau \geq 1$. The problem to decide in the discrete model whether there exists a popular place in T has an $\Omega(\tau n \log \tau n)$ lower bound.*

Theorem 7. *Let T be a set of n trajectories over τ time-steps, for any $\tau \geq 1$. There exists no $o(n^2\tau^2)$ time algorithm to decide whether there exists a popular place in the continuous model with input T, unless there exists an $o(N^2)$ time algorithm to decide 3-SUM 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. 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.
5. H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15(3):211–228, 2006.
6. H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38:165–194, 1989.
7. 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.
8. J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. Compressing spatio-temporal trajectories. To appear at ISAAC, 2007.
9. 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.
10. J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of motion patterns in spatio-temporal sets. *GeoInformatica*, 11(2):195–215, 2007.
11. R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann Publishers, 2005.
12. M. Hadjieleftheriou, G. Kollios, V. J. Tsotras, and D. Gunopulos. Indexing spatio-temporal archives. *The VLDB Journal*, 15(2):143–164, 2006.
13. 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.
14. 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.
15. 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.
16. 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.
17. 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.