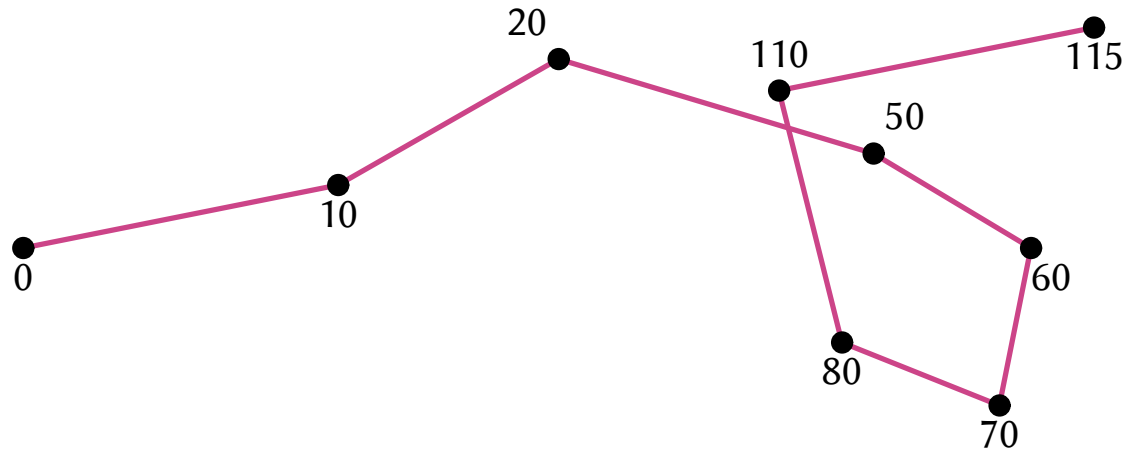
CCCG 2018

Looking for Bird Nests
Identifying Stay Points with Bounded Gaps

Ali Gholami Rudi

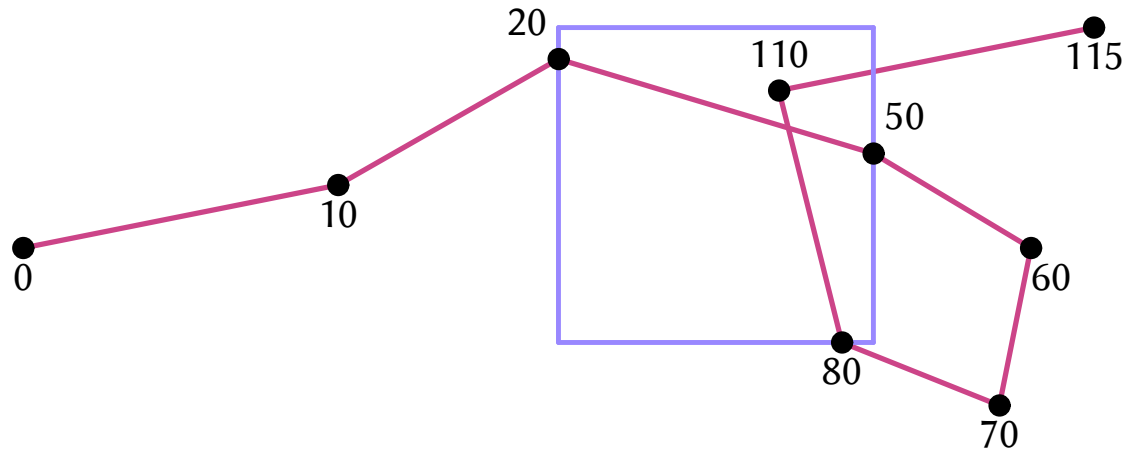
Babol Noshirvani University of Technology



Vertices: locations with timestamps
Edges: constant speed, straight line

The goal in identifying the stay points of a moving entity is finding out where it spends a significant amount of time. We model an entity as a point and record its location at certain points in time. We thus obtain a set of locations with a timestamp, which we represent as vertices. We assume that the entity moves at constant speed and in a straight line from one vertex to the next (but the speed in different edges of the trajectory may differ). We thus can map the time to the location of the entity. We call this mapping the entity's trajectory.

Hotspots, Popular Places, Stay Points 2



Stay points: where an entity spends a significant amount of time.

Shape: axis-aligned square

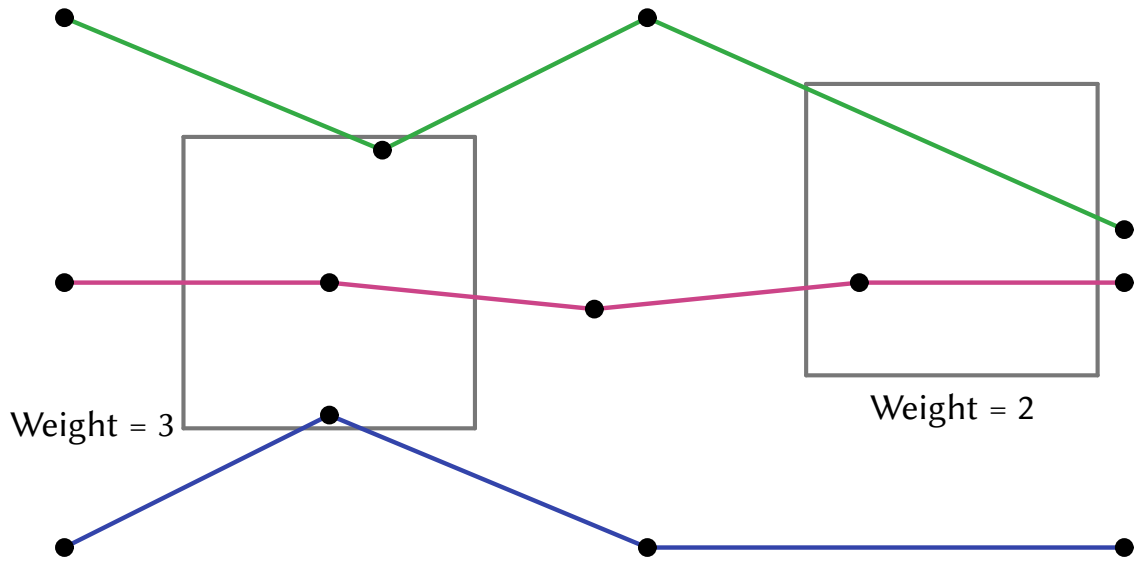
Criteria:

Number of visits (Benkert, Djordjevic, Gudmundsson, Wolle; 2010)

Duration of visits (Gudmundsson, van Kreveld, and Staals; 2013)

Our goal is identifying regions in which an entity spends a significant amount of time; these regions are called popular places, hotspots, or stay points. Several studies have been conducted for the identification of stay points. However, from a geometric perspective, there are two notable papers.

Benkert, Djordjevic, Gudmundsson, Wolle (2010)



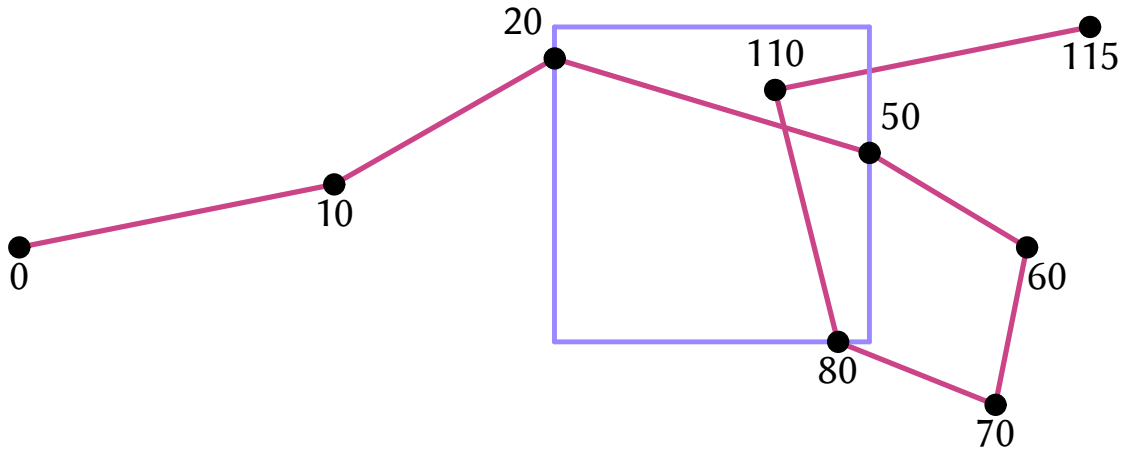
Discrete model: $O(n \log n)$.

Continous model: $O(n^2)$.

Benkert and others define a popular place as an axis-aligned square of fixed side length, which is visited by the maximum number of entities. In their definition of a popular place for a collection of trajectories, only the number of visits by different entities is important and not the duration the entities spend inside it. To count the number of visits for a square, they consider two models, based on whether the partial inclusion of the edges of trajectories are considered as a visit or if at least a vertex should be included. For the former they give an $O(n^2)$ algorithm and for the latter they present an $O(n \log n)$ algorithm, both of which they show to be optimal.

Gudmundsson, van Kreveld, Staals (2013)

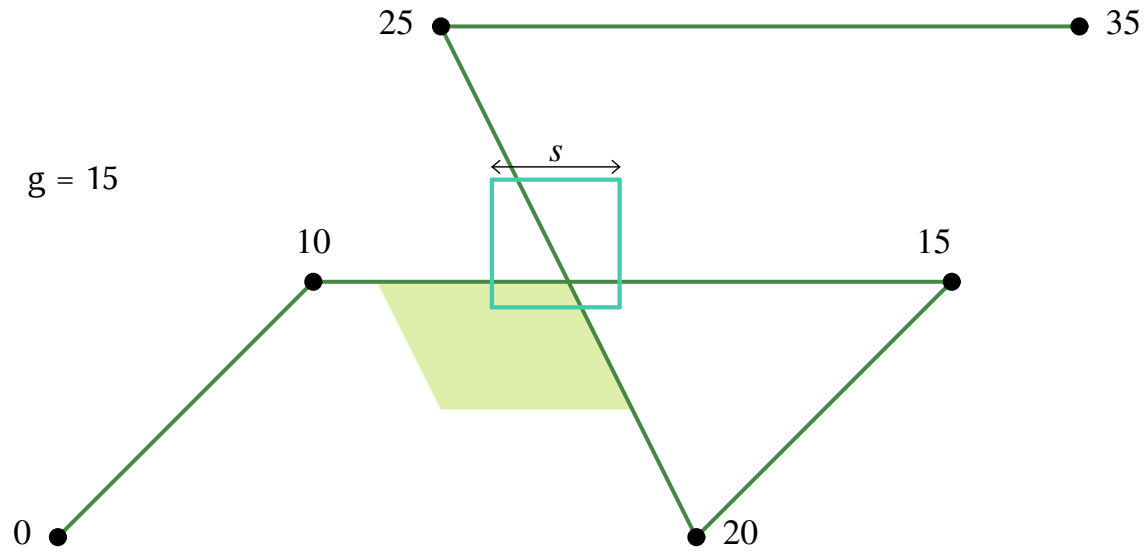
Maximum or total visit duration



Gudmundsson and others studied several different variants of trajectory popular places, in all of which the time spent in the region is taken into account. Their definitions assume a single moving entity but some of them can be extended to multiple entities. The hotspot is decided either based on the total duration in which the entity was in that region or the duration of the maximum sub-trajectory in which the entity was inside it. For instance, in this figure the entity visits the blue square twice. When considering the total duration of visits, the weight of the square is about 65 and if the maximum visit duration is considered, it is 35.

Allowing the entity to leave the region for short intervals
  A bird returning to its nest to feed its chicks.
  Leaving the cinema for the bathroom
Finding such stay points among multiple intersting places
  Arboleda, Bogorny, Patio (2017)
  Potential stay points (intersting places) are given as input

In some applications, the entity is allowed to leave a stay point, but should return after a short time. For instance, a bird regularly leaves its nest to find food for its chicks or someone at a cinema may leave it for a few minutes for the bathroom. Arboleda and others study a similar problem. They present a simple algorithm that gets a list of potential stay points (or interesting places) as input and decides which of them is a stay point, in which the entity's absence is never more than the specified limit without visiting it again.

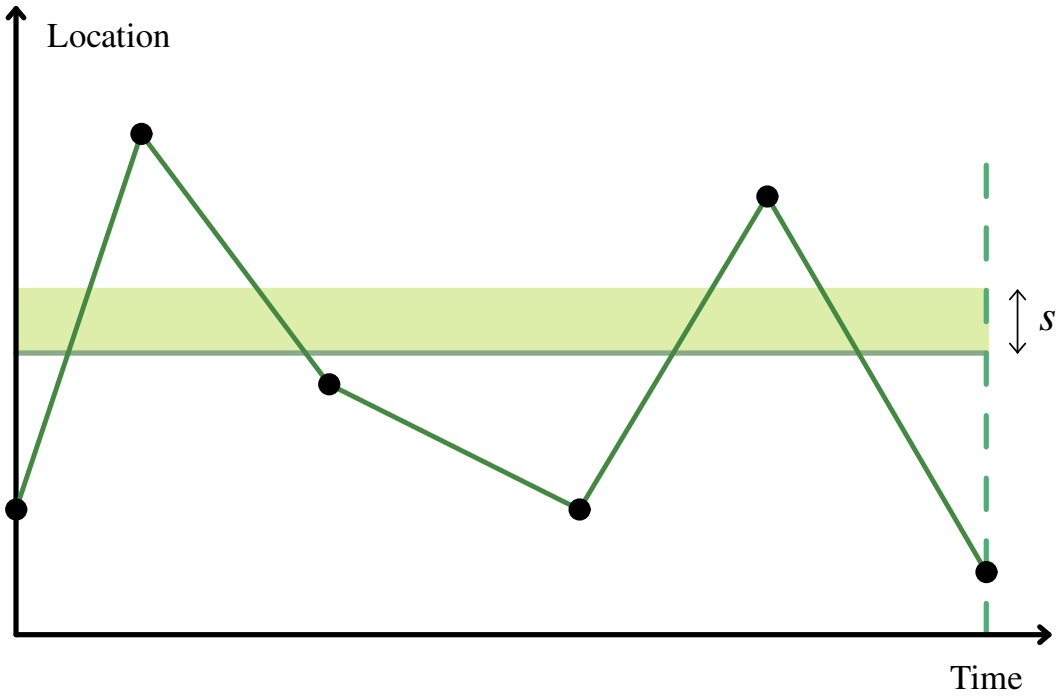


Stay points: axis-aligned squares with fixed side length (s)

The entity should never be outside the regions for more than the maximum allowed absence time (g)

Stay maps: the lower left corner of all stay points

In this paper our goal is to find all stay points of a trajectory. We model each stay point as an axis-aligned square with fixed side length (s) and the entity cannot be outside a stay point unless for a time smaller than g. The region containing the lower left corners of all stay points of a trajectory is its stay map, which is what we try to compute in this paper. In this figure, g is 15, the green region is the stay map, and the green square is a stay point.



Lemma: The stay map of a trajectory in R^1 is continuous

We begin with one-dimensional trajectories.

We start with one-dimensional trajectories. This figure shows the location of an example entity on the line. The stay map of a one-dimensional trajectory is continuous. The informal proof is that an entity visits the region between two stay points regularly.

Event points: points in R^1

i) a trajectory vertex lies on that point

ii) the time gap between two visits to that point is exactly g

Lemma: The set of event points of a trajectory can be computed in $O(n \log n)$ time.

Sweeping the time-location plane vertically.

Lemma: The stay map of a trajectory in R^1 starts and ends at an event point or at distance s from one.

Otherwise, we can move the leftmost (similarly, rightmost) stay point to the left (right) to obtain a new leftmost (rightmost) stay point.

We define the event points of a one-dimensional trajectory as points on the line, on which either a trajectory vertex exists or a visit to that point happened after exactly g from its previous visit. We can find all event points of a trajectory by sweeping the time-location plane vertically and keeping track of the length of the segments that result from cutting the sweep line by trajectory edges. The next important observation is that the stay map of a one-dimensional trajectory starts and ends at an event point or at distance s from one. This can be proved by moving the left-most or right-most stay point in the stay map slightly to the left or right.

Lemma: we can answer in $O(n)$ time whether a point is in the stay map or not, and if not, whether the stay map is on its left side or on its right side.

Algorithm for trajectories in R^1 (with the time complexity $O(n \log n)$):

- Obtain the event points and points at distance s from them.
- Perform a binary search to find the left end point.
- Perform a binary search to find the right end point.

If we are given a point on the line, we can tell whether it is in the stay map, and if not, if the stay map is on its left or on its right. This can be done by linearly scanning trajectory edges. With this lemma, we can use the following algorithm to find the stay map of a trajectory. We first obtain all event points and points at distance s from them and sort them. Then we use binary search to find the left and right end points of the trajectory.

Notation:

- $T(a, b)$: the sub-trajectory from time a to time b
- $P(a, b)$: the lowest left corners of all squares of side length s that contain any part of $T(a, b)$.
- $M(0, t)$: the stay map of $T(0, t)$.

Unfortunately the one-dimensional algorithm does not work for two-dimensional trajectories, because the stay map may no longer be continuous. We use these notations in the rest of this presentation.

Algorithm: incrementally compute $M(0, D)$

- Let $M(0, g) = P(0, g)$;
  $P(0, g)$ is the union of polygons $P(u, v)$ for all edges uv in $T(0, g)$.
- Compute $M(0, b)$ from $M(0, a)$, in which $M(0, a)$ is the last computed stay map and b is the smallest value after a such that $b - g$ or b is the timestamp of a trajectory vertex.
  Let V be the difference between $M(0, a)$ and $M(0, b)$;
  we compute V to obtain $M(0, b)$.

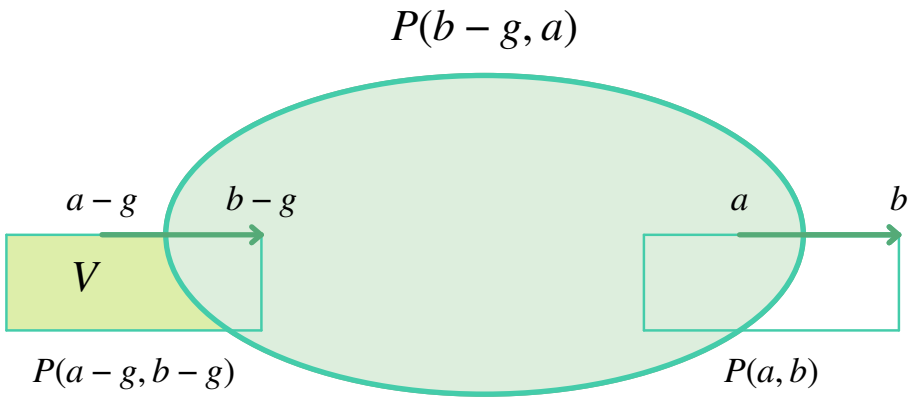
We incrementally compute $M(0, D)$ as follows. [Description of the algorithm]. V contains the lower left corners of all squares which were a stay point in $T(0, a)$ but not in $T(0, b)$; the entity has not visited them in $T(a, b)$ within the limit g.

Need to compute V, the difference between $M(0, a)$ and $M(0, b)$.

- Let $V = V' \setminus P(b - g, a)$, where

$$V' = \bigcup_{0 \le \delta \le g} P(a - g, a - g + \delta) \setminus (P(a - g + \delta, b - g) \cup P(a, a + \delta))$$

- The shape of V' depends on $T(a, b)$ and $T(a - g, b - g)$.



- If $P(a - g, b - g)$ and $P(a, b)$ are disjoint, $V' = P(a - g, b - g)$.

The challenging part of this incremental algorithm is the computation of the difference V. We can extract $P(b - g, a)$ to make its computation easier. Here, V' depends on $T(a, b)$ and $T(a - g, b - g)$ and is a polygon of constant complexity. When $P(a - g, b - g)$ and $P(a, b)$ are disjoint, V' is $P(a - g, b - g)$, as this figure shows.

- $P(b - g, a)$ is the union of $O(n)$ simple polygons.
- The union of the differences (V) for all iterations of the algorithm, containing $O(n^2)$ simple polygons.
- An $O(n^2)$ implementation seems unlikely.

Definition:

- $(1 + \varepsilon)$-approximate stay point: the entity is never outside the region for more than $g + \varepsilon g$ time.
- $(1 + \varepsilon)$-approximate stay map: all exact stay points and possibly some of its $(1 + \varepsilon)$-approximate stay points.

We now discuss a faster, approximation algorithm. We define approximate stay points and stay maps as follows.

A snapshot: $P(t, t + g)$, in which $0 \leq t \leq D - g$

The lowest left corner of every stay point should appear in each snapshot.

Approximation algorithm:

- Let $\lambda = \varepsilon g$
- Compute snapshots $P(t, t + g)$ for $t = i\lambda$, for integral values of i from 0 to D/λ.
- Compute the intersection of these snapshots.

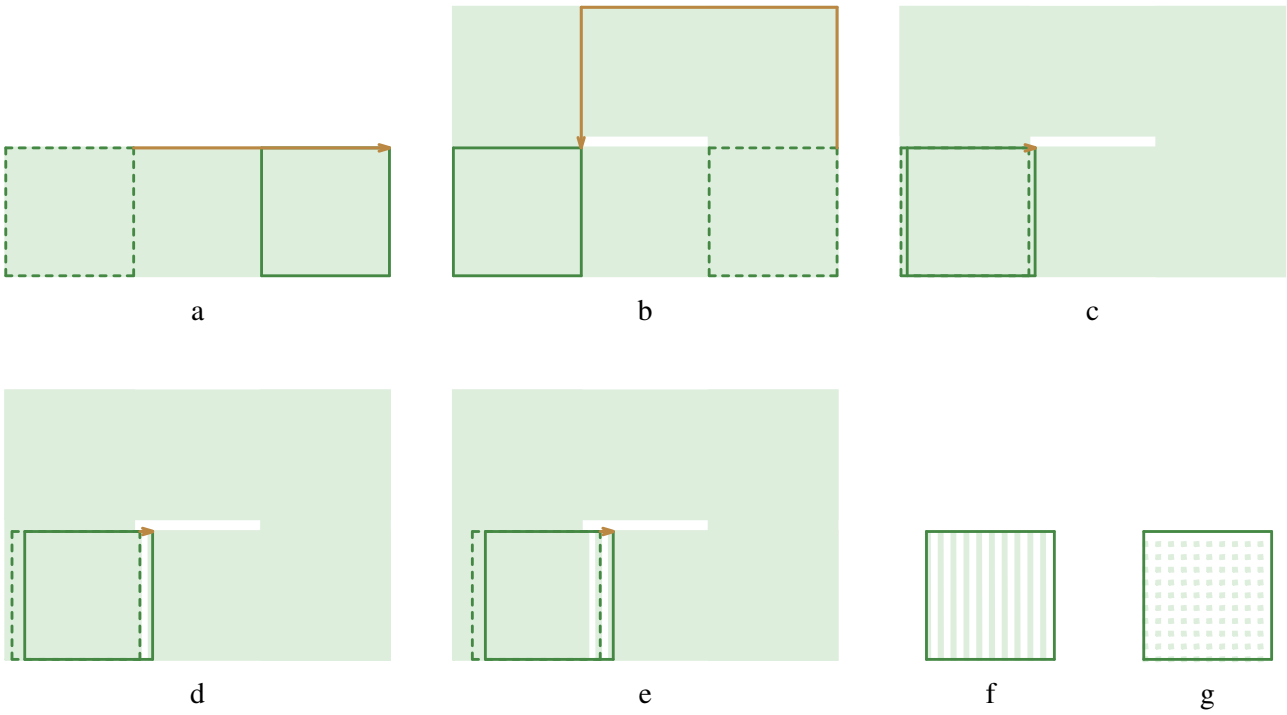
A snapshot of a trajectory is the set of lowest left corners of every stay point that contains any sub-trajectory of duration g. Clearly, the lowest left corner of every stay point should appear in every snapshot, because the entity cannot be outside any stay point for more than time g. The main idea in the approximation algorithm is to compute the intersection of many snapshots to compute an approximate stay map. For a positive constant ε, the algorithms computes the intersection of the snapshots that are εg apart.

Approximate 2D Stay Maps — Analysis 16

The output contains the lowest left corner of every stay point.

For every square whose lowest left corner is in the output:
the entity cannot be outside for more than $g + \varepsilon g$;
for any square r in the output:
- suppose the entity leaves r at t_b and reenters it at t_e
- t_1 is the snapshot immediately before t_b
- if $t_e - t_b > g + \varepsilon g$ then $t_e \leq t_b + g + \varepsilon g$

Clearly, every stay point of a trajectory is included in the intersection of the snapshots. Therefore, it is adequate to show that the square whose lower left corner is in the output of the algorithm is an $(1 + \varepsilon)$-approximate stay point. This can be done by showing that the entity cannot be outside any such square for more than $g + \varepsilon g$.



We conclude this presentation with an example trajectory with a stay map of complexity $\Theta(n^2)$. The pattern in part f for the stay map can be generated by the trajectory shown in parts a-e. The entity returns to its starting point in parts a-c in about time g. Then, m vertical strips can be generated by moving quickly s/m to the right after every g/m time. Repeating this trajectory, after rotating it 90 degrees, results in figure f. This implies that we cannot hope to find an algorithm with the time complexity $o(n^2)$. However, it may be possible to do so if we limit the output, for instance, by looking for a stay point with the maximum visit duration. Also this bound is not tight.

Multi-trajectory stay maps

Each stay point is visited by at least one of the entities in any interval of duration g

It seems interesting to study the multi-trajectory version of this problem. Each stay point should be visited by at least one of the entities in any time interval of duration g. The approximation algorithm may be modified to compute the intersection of the union of the snapshots of the entities. Fast exact algorithms or faster approximation algorithms seem interesting.