Pattern extraction in trajectories and its use in enriching visualisations

Ingo van Duijn

Abstract

Nowadays location aware devices are commonplace and produce large amounts of trajectories of moving objects like humans, cars, etc. To analyse this data there is need for proper visualisation techniques and methods to automatically extract knowledge from the data. We propose to enrich existing visualisations of trajectories by indicating significant occurrences of specific patterns.

We give definitions of two patterns, junctions and stop areas, and study the geometric properties of these formalisations. We also give efficient algorithms to compute a given number of significant locations, and we implemented a variation of these to demonstrate the proposed technique.

Contents

1 Introduction

With advancing technologies in recent years, more and larger sets of *trajectories* have become available. A trajectory in this sense is the path described by objects, such as humans, aeroplanes, animals, hurricanes, etc., moving around in their respective environment. These paths are typically recorded by some mobile GPS-enabled device sampling the moving object's location at some rate. Because of this, a trajectory is usually considered to be a series of timestamped georeferenced locations. In its raw form, trajectories – especially in large quantities – are not suited to be analysed by hand. There is therefore a need for techniques to facilitate the analysis of trajectories. This topic has received attention[2] from geographic information science, which is concerned with gathering, storing and analysing geographic data in general. Because of the wide range of origins of the trajectories and its explicit relation to the physical world, it has also sparked interests in other communities such as ecologists[14] and urban planners[36]. In this thesis we concern ourselves with various steps in the analysis process, providing some basic tools which can be used fairly independently. We look at two specific *patterns* in trajectories and give algorithms to process them in order to extract information which is used in a final *visualisation* step.

We distinguish between two different – albeit related – approaches to analyse trajectories. Visualisation is an intuitive way to inspect the trajectories. By employing our natural visual analytical skills, humans can analyse the trajectories by looking at the visualisations. The other approach is to reason about trajectories before (thoroughly) inspecting the data. By defining a desired pattern or set of constraints, computers can be put to work to determine if these can be found in the input.
Visualisations usually need some definition of what to visualise, if only that it has to connect the timestamped locations in the right order, and are therefore not entirely devoid of some information processing. Conversely, if an instance of a predefined set of constraints has been found, then this information needs to be conveyed – preferably with some context. In this case it is only natural to fall back to some sort of visualisation.

1.1 Visual Analytics

A variety of techniques collectively known as *visual analytics* aim to analyse data (such as trajectories) by combining the strength of human and electronic data processing[24]. As the name suggests, an important part of the techniques is concerned with visualising the data. For trajectories, a basic visualisation method is to plot their paths over some representation of their origin (e.g. a cartographic map). This particular method however, quickly suffers from clutter or occlusion and does not visualise the temporal aspect of the trajectory. One of the challenges in visual analytics of movement is to overcome such shortcomings while still conveying relevant information[5].

The other part of visual analytics is the analysis of the visualisations. Interactive visualisation tools are used by domain experts – and with increasing availability of these tools regular users – to explore the data. The interaction allows the users to manipulate the data on the fly by for instance zooming or selecting a subset of the data. By having multiple "views" on the data, subsets deemed interesting can be analysed based on what the different visualisations have to offer. This kind of analysis thus stands to benefit from a diverse set of visualisations, together giving a comprehensive view on the data. Both the visualisations and the user's analytical skills are important for discovering new knowledge and ultimately formulating new hypotheses.

1.2 Quantitive Analysis

Trajectories are inherently geometric objects and therefore their analysis and knowledge discovery can also be facilitated by a data mining approach based on their geometric properties. This approach aims to quantify[29] these properties to allow for queries on the data, or optimisation problems. For example, a similarity measure can be defined between trajectories. This allows for queries such as finding the most similar trajectory to some input, or alternatively using it to cluster trajectories[32]. Simple properties such as speed, acceleration, or turning rate of a trajectory are fairly straightforward to define and do not allow very different definitions. For more complicated properties or patterns however, there is usually no single way to formulate these. At the two extreme ends of the spectrum of modelling such patterns, we find domain specific models[10] and general purpose models[27]. The latter approach usually assumes little to no extra information besides the trajectories, while specialised models sometimes assume additional information such as extra attributes for the timestamped location or similar metadata. In this thesis we take the latter approach and try to give simple definitions based on only the trajectories.

1.3 Contributions

Cluttered areas in visualisations of trajectories can become practically void of useful information. Possibly there are discernible patterns around the clutter, but in it information is for the most part occluded. The clutter indicates the presence of a substantial number of trajectories, possibly containing an interesting pattern. Because the cluttered area does not convey very useful information, this space can be utilised for a different kind of visualisation. Based on this observation, we propose a method to enrich basic visualisations of trajectories by indicating prominent occurrences of predefined patterns with symbols. We specifically target patterns in multiple trajectories, such that the symbols give an abstract representation of some information contained in the trajectories it overlays. To investigate this idea, we study the following two patterns:

- First we look at *junctions* of trajectories. In a simple plot, a junction of trajectories from e.g. pedestrians in a city looks like bundles of trajectories meeting on the location of the actual road intersection. Although the junction might be very apparent, it can be very hard to judge how many pedestrians took a left or right turn on the junction. In this case a useful symbol would be an indication of the turning behaviour of the pedestrians, giving valuable insight for e.g. urban planners.
- The other pattern we study are *stop areas*. A major limitation of a simple plot of trajectories is that the temporal element is lost. This means that if a tracked object remains at the same location for some period of time, this is invisible on the plot of its path. Even if the stop were somewhat visible, in areas with many stops – and thus many trajectories – this information is quickly occluded. Indicating the presence of a significant concentration of *stops* provides context to an otherwise seemingly meaningless cluttered region

Note that for the latter pattern the idea of *enriching* visualisations becomes clear. It is possible to visualise stops by for instance colouring them, but this removes the option of colour-coding something else. By utilising the cluttered regions to place symbols, we leave other methods to visually encode information open.

To demonstrate our proposed method, we have implemented a proof of concept which overlays a basic plot of trajectories with crude symbols. For this we first modelled both patterns and analysed their geometric properties. We prove that there are at most $\mathcal{O}(\alpha(n)n^3)$ different junctions, where n is the number of timestamped locations in all trajectories combined and α the inverse Ackermann function. With our algorithm to "measure" a junction in $\mathcal{O}(n)$ time, we give an algorithm that computes the junction with highest measure in at most $\mathcal{O}(\alpha(n)n^4)$ time. For the stop areas we give an algorithm that identifies all individual stops in linear time. We then give an algorithm to find the area with the most unique stops in $\mathcal{O}(n^2)$ time. Finally, we describe how to determine k significant junctions (or stop areas) by iteratively applying a simple selection procedure, where k is the desired number of symbols.

In the next section we discuss a selection of works related to this thesis. After that, in Section 3.2 and 3.3, we define our models for junctions and stop areas respectively. For both we give a method to assess the quality of either around a point in the plane, and give algorithms to compute them for every point. In Section 3.4 we give a selection procedure to find the most significant points. In Section 4 we describe our implementation of previously given algorithms and present the results from our test data. Finally we conclude in Section 5 and discuss the merits and possible continuations of our work.

2

Related Work

The study of trajectories, or spatio-temporal movement data, has been an active area of research[5, 29]. We first discuss some existing work on trajectory visualisation techniques and then related work on feature extraction, including junctions and stop patterns.

2.1 Trajectory Visualisation

Multiple techniques have been developed to visualise trajectories, with varying focus on visualising temporal patterns and spatial patterns. In the early 1970s Hägerstrand proposed the space time cube[22], a three dimensional visualisation technique which plots time on the z-axis. By doing so the temporal element of trajectories becomes visible. Kraak[25] suggests that with proper interaction (e.g. rotation), the STC is suitable to reveal complex spatio-temporal patterns in trajectories. To alleviate the clutter in the STC, Demšar et al.[13] visualise the space-time density of trajectories to reveal spatio-temporal patterns. Trajectories can appear distorted inside the STC, and questions have been raised[26] about the efficacy to convey temporal patterns. The STC nonetheless remains a popular tool and its use in combination with new technologies such as Google Earth keeps being explored[30]. Another approach to explicitly visualise temporal patterns is to drop the original spatial layout and instead plot time against some derived spatial property[12].

Massive sets of trajectories can be hard to visualise; a standard technique to handle large data sets is to first simplify it by clustering or aggregating. Andrienko and Andrienko[4] apply this principle to create a map which abstractly depicts the flow of trajectories between contiguous regions on the map; they automatically infer interesting regions, and cluster segments of trajectories moving between the same two regions. There exists a variety of clustering techniques, but the complex nature of trajectories make it hard to determine a suitable technique for visualisations. Several authors [3, 32] propose methods to interactively cluster and visualise trajectories; combining different techniques and evaluating the clusters on the fly. Scheepens et al.[34] visualise massive sets of trajectories by first creating a density field and subsequently aggregating this information.

2.2 Trajectory Analysis

Trajectory data is being subjected to different kinds of analyses. There is the more data mining oriented approach observed in [17, 18, 28, 39]. These kinds of methods tend to discretise the problems or use general data mining or querying techniques to discover patterns. Another approach is focused on the geometry of trajectories and how they are related in time[6, 19, 20]. These methods define patterns in terms of the trajectories' geometric properties, and consequently the described algorithms usually are more exact in nature. The same kind of patterns are studied by both approaches, including patterns in movement [39, 16, 17], places of interest [18, 6, 20], and trajectory similarities [7, 28].

Different kinds of movement patterns have been defined. To give an overview of the types of movement patterns in trajectories, Dodge et al.[16] categorise a number of parameters extractable from trajectories and describe generic movement patterns in terms of these. Laube et al. [27] define several patterns such as leadership, flocks, and convergence, and give algorithms to detect them.

Trajectory data can contain interesting places, a concept with varied usage. Gianotti et al. [18] detect patterns in trajectories visiting the same interesting places. They assume the interesting places are given, or alternatively propose a simple density based method to identify them. Verhein et al. [39] assume the space is divided in arbitrary regions. Depending on how many trajectories move between the regions, they determine regions with interesting temporal characteristics, including thoroughfares and stationary regions. In [6], Benkert et al. define a place of interest as a region containing enough vertices or edges from different trajectories, and give algorithms to find them.

2.2.1 Stops

The notion of a stop, or stationary region, in trajectory data has had different treatments in various works. Spaccapietra et al. [35] treat a trajectory as a sequence of stops and moves, and assume this information is given by the application. The practice of dividing a trajectory in different parts based on some common property (e.g. a stop) is called segmenting. In [8], Buchin et al. cover the automatic segmentation of trajectories based on various properties more generally. Alvares et al. [1] detect stops in trajectories by taking a set of regions as input, and check if parts of trajectories stay within those regions long enough. Two variations on this method also detect stops which do not lie within the predefined regions. One clusters based on speed and then filters based on duration [31], and another detects stops by clustering based on a minimal change in direction [33].

2.2.2 Junctions

Guo et al. [21] developed a visualisation tool and used it to study trajectory data on a road intersection. In their visual analysis of the trajectories, they look at patterns in the trajectories based on which turn the moving objects took. They also try to uncover information about individual trajectories (e.g. potentially dangerous situations) hidden in the clutter.

2.3 Symbol Placement

In 1975 Imhof[23] gave an overview of important criteria when adding labels to a cartographic map, a process which used to be done by hand. Since then, automated methods to place labels on maps [40, 41] have been developed, as well as means to evaluate these [15]. To place a label, the bounding box or at least the curve on which the label is placed has to be carefully calculated. For symbols, the geometry involved can be even more complicated as it can have shapes other than curves or (curved) rectangles. In [37] van Kreveld et al. give algorithms to place square and circular symbols on maps while avoiding overlap, and in [9] the authors try to find the least unfavourable overlap of symbols by maximizing the minimal visible boundary of all the symbols. Automatically drawing aesthetically pleasing symbols becomes increasingly harder when the information portrayed is more complex, yet successful efforts have been made. For example, in [38] Verbeek et al. draw flow maps displaying complex flows of entities between regions.

3

Methodology

In this section we give models for junctions and stop areas in a set of trajectories, give algorithms to compute them, and analyse their running time. First we formally define trajectories and give additional notation used in this section. For both patterns, we discuss important properties and give definitions until we arrive at a way to quantify them. Since the goal is to find the most significant occurrences, we analyse how to find a representative (finite) subset of locations such that it contains the best occurrence. For both patterns we conclude by giving the algorithms to compute this best occurrence and analyse their running time. Finally, we describe a simple method to determine not just the best occurrence, but the best k occurrences – where k is some small constant.

3.1 Trajectories

Here we introduce a formal notion of a trajectory and the notation used throughout Section 3. A trajectory T is a piecewise linear continuous function mapping an interval $[a, b]$ to the plane, so the image of a trajectory will look like n connected line segments: a polyline. A point $c \in [a, b]$ is called a *breakpoint* of T if it lies on a vertex of this polyline. The value of a marks the moment in time at which the trajectory begins and b when it ends. Depending on the context, we refer to a or $T(a)$ as the *startpoint* of T, and similarly b or $T(b)$ the *endpoint*. Let $\mathrm{Path}(T) = \{T(s) | s \in [a, b]\}$ denote the path of a trajectory T and $|\mathrm{Path}(T)|$ its length.

T restricted to an interval $[s, t]$ with $a \leq s$ and $t \leq b$ is called a *sub trajectory* and is denoted $T[s, t] \subseteq T$. The *linear closure* of $T[s, t]$ is denoted $T^+[s, t]$, and is the smallest sub trajectory containing $T[s, t]$ which has its start and endpoint on breakpoints of T. We use the term sub trajectory to emphasize that it is a subset of a larger trajectory. When it is not relevant that a sub trajectory is contained in another trajectory, we simply use the term trajectory.

We discuss intersections between paths of trajectories and line segments, and as such these might coincide in more than a single intersection point. To avoid degeneracies, we therefore assume no two line segments – including those of a trajectory's path – are parallel.

3.2 Junctions

A junction is a meeting of routes: a place where the paths of moving objects diverge to a limited set of directions. Routes in this sense are a tendency for moving objects to follow the same general path, typically because of real world features like roads. In a set of paths of objects moving along such routes, the junctions can thus reflect part of an underlying structure creating them; be it a road network or some collective behaviour of the moving objects. We concern ourselves with the general case where we only have the tracked movement, meaning that this kind of additional information has to be inferred manually or extracted automatically. It can be a laborious task for an analyst to individually identify and assess all locations where trajectories meet and diverge. Therefore, to help gain insight into bare movement data, we first enrich it with automatically extracted information based on the concept of a junction.

The remainder of this section is organised as follows. We first describe the basic features of a junction, and formalise a notion of a trajectory taking a *turn* on a junction. We then argue how to measure the quality of a candidate junction, based on how many trajectories connect each pair of its turns. Following this is a detailed description of a representative set of candidate junctions. Finally, we give a step-by-step explanation of how to compute the measure for each candidate junction, concluding with a running time analysis of the final algorithm.

3.2.1 Formalisation

We consider a junction to be a local phenomenon. In some area various routes meet and then diverge, after which they lose their relevance to the junction. Beforehand it is not known which areas contain such a meeting of routes; so for now, we refer to an arbitrary region as a *tentative junction*. We introduce a set of properties characteristic of a proper junction, and subsequently a method to determine their significance as a junction. Of course there is no unique way to infer the presence of a junction from mere movement data. We therefore try to keep these properties as simple and few as possible, offering flexibility with several intuitive parameters.

The size, and to a lesser degree the shape, of a tentative junction are directly related to the *scale* of the eventual junctions that are inferred. For instance, on a small enough scale a traffic circle comprises several distinct junctions, yet on a larger scale it can be considered a single junction. In our application the goal is to visualise the junctions in the scope of a fixed size image. We assume that in such a limited scope there is a reasonable scale on which to detect junctions, and therefore let the size of the visualization determine some fixed scale λ for the tentative junctions. Furthermore, we choose to consider only circular tentative junctions; this is the most elementary shape and a natural choice for junctions.

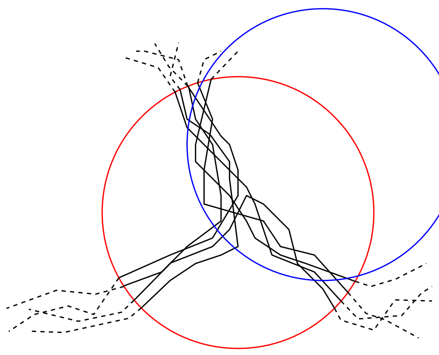

Figure 1: A junction of trajectories with two circular tentative junctions. The red circle captures the junction adequately while the blue circle does not.

Aside from this, we assume that the influence of the shape is overshadowed by the influence of the scale, and that it therefore suffices to only vary the scale and not both.

Figure 1 shows a "nice" junction and illustrates the idea of a tentative junction. As we said earlier, only the sub trajectories inside the tentative junction are considered relevant. Though both circles arguably contain the junction, the red circle clearly contains a more complete picture. We say that these sub trajectories are more *salient*, or in other words, they first pervade the inner region before leaving the area altogether. This property reflects the typical converging of routes towards the center of a junction. We use this to filter out trajectories which are not relevant to the junction, like those that only briefly skim the boundary.

Definition 1. Given a set of trajectories $\mathcal{T}$, a point p, and a scale λ; let D_p and d_p denote the discs centered at p with radius λ and $\frac{\lambda}{2}$ respectively.
A sub trajectory S, with $S \subseteq T \in \mathcal{T}$, is a **$\lambda$-salient trajectory** of p if:

- $\mathrm{Path}(S)$ lies completely within D_p
- The start and endpoints of S lie on the boundary of D_p
- S intersects d_p
- S is no strict sub trajectory of one for which the above criteria hold (S is maximal)

Let $\mathrm{Sal}_p(\mathcal{T})$ denote the set of all λ-salient trajectories of p derived from $\mathcal{T}$.

Since the scale is always fixed, we refer to elements from $\mathrm{Sal}_p(\mathcal{T})$ simply as salient trajectories. Additionally, if $\mathcal{T}$ is already clear from context, we write Sal_p. Figure 2 shows an example of a tentative junction with salient trajectories. Note that a single trajectory can induce multiple salient sub trajectories.

Earlier, we informally described a junction as the meeting of routes in an area. The first part is captured by salience; a trajectory is only relevant if it crosses the center d_p of the tentative junction. The routes themselves – or a pair of *turns* in the context of a junction – are subsequently based on how these salient trajec-

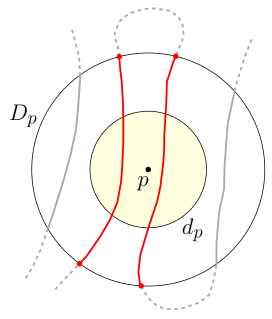

Figure 2: Two trajectories induce four sub trajectories contained in D_p, but only two of these are salient as they also cross d_p.

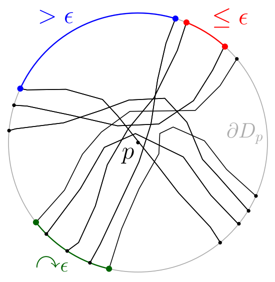

Figure 3: The red points are ϵ-adjoint and the blue and green points are not. The green points are however transitively ϵ-adjoint.

tories intersect the *boundary* ∂D_p of that area. Turns are the most characteristic feature of a junction; once they are determined a tentative junction can be assessed based on how the trajectories are distributed among them.

To illustrate how we derive the turns, consider movement data from objects moving on fixed tracks, meaning that the paths of objects moving on the same track *exactly* coincide. Thus, disregarding degeneracies, any ∂D_p will intersect the network of tracks in a fixed number of places. We consider these intersection points to be the turns in p, and two trajectories with their paths coinciding on ∂D_d are considered taking the same turn.

Former approach is of course not feasible to actually determine the turns: it will not work if there is even the slightest variation of movement in the same track. However, its simplicity is appealing and we derive a more generally applicable method by making the former more robust. Instead of letting one intersection uniquely define a turn, we *cluster* intersections with the boundary of the tentative junction.

Definition 2. Consider the intersection points between Sal_p and the boundary ∂D_p. Two such points v and u are called **ϵ-adjoint** if there is an arc on ∂D_p with length lesser or equal to ϵ, which contains both v and u.
Let $\curvearrowright_\epsilon$ denote the transitive closure of this relation.

Figure 3 shows an example of points – appearing green in the figure – which are not ϵ-adjoint, though they are transitively ϵ-adjoint. Since $\curvearrowright_\epsilon$ is an equivalence relation, it partitions – or clusters – the set of the intersection points between Sal_p and ∂D_p. Every cluster has the property that no two *consecutive* intersection points lie a distance greater than ϵ apart. To denote at which turn a trajectory enters or leaves the junction, we use the following notation. If there are m turns, let $c_1, ..., c_m$ be intersection points from each turn – essentially indexing the turns. A salient trajectory $T[s,t] \in \mathrm{Sal}_p$ is said to take the ith turn, denoted $T \Rsh^i_p$, if either $T(s) \curvearrowright_\epsilon c_i$ or $T(t) \curvearrowright_\epsilon c_i$ holds.

3.2.2 Junction measure

In this section we describe a method to measure the quality of a tentative junction. Since we derive the properties of a tentative junction from a point, we can also refer to a point's measure. Later we describe how to apply the measure to all points in the plane, which partly explains why we refer to junctions as *tentative;* it is to be expected that not all points lie on a junction. Another reason to use the term tentative is because the measure does not classify points as lying on a proper junction, it only provides a means by which to compare them. As such, the highest values measured on some input indicate locations which – according to our definition – most significantly resemble junctions.

Measuring a junction should consist of two parts: weight and quality. Weight is necessary to distinguish large from small junctions and quality to distinguish between "good" and "bad" junctions. Since we are interested in the most significant junction, we think both parts are necessary. It should be possible from two equal-sized junctions to pick the one with better quality. Similarly, the bigger of two equally well-behaved junctions should be considered more significant.

We define weight and quality in terms of *bundle* size. A bundle consists of all the trajectories in Sal_p which take the exact same turns on the junction. To make this information comprehensible we introduce the *junction matrix*.

Definition 3. For a set Sal_p of trajectories, let Q_p be the **junction matrix** with:

$$\begin{aligned} &Q_p[i,j] = |\{T \in \mathrm{Sal}_p : T \Rsh_p^i \text{ and } T \Rsh_p^j\}| && \text{if } i \neq j \\ &Q_p[i,j] = 0 && \text{otherwise} \end{aligned}$$

A cell $Q_p[i,j]$ thus contains the number of trajectories *connecting* the ith and jth turn. This matrix then represents the very basic structure of a junction, as it only preserves the number of trajectories connecting each pair of turns. By construction the trace of this matrix is zero; this way we do not consider trajectories leaving and entering the junction at the same turn to be part of this structure.

We use only this matrix to determine the quality of a junction. Therefore our measure does not depend on the way turns and bundles are defined, it only requires some (reasonable) method to derive the matrix. We proceed by giving the measure based on a junction matrix, and subsequently explain how it captures weight and quality.

Definition 4. For a junction matrix Q_p of size $m \times m$, let $\Sigma Q_p[i]$ denote the sum

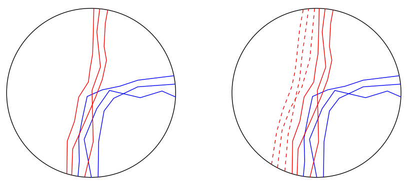
Figure 4: In both junctions, the red bundle only counts as 3 because it is limited by the size of the other bundles combined, in this case a single bundle of size 3

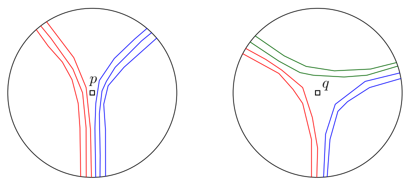

Figure 5: Two junctions with the same number of turns and trajectories, but different connectivity.

of cells in the ith row. The measure $J(Q_p)$ is defined:

$$J(Q_p) = \sum_{i=1}^{m} \left(\Sigma Q_p[i] - \max(\max_{1 \leq j \leq m} Q_p[i,j] - (\Sigma Q_p[i] - \max_{1 \leq j \leq m} Q_p[i,j]),\, 0) \right) \tag{1}$$

$$J(Q_p) = \sum_{i=1}^{m} \left(\Sigma Q_p[i] - \max(2 \cdot \max_{1 \leq j \leq m} Q_p[i,j] - \Sigma Q_p[i],\, 0) \right) \tag{2}$$

First note that we sum some value for each turn on the junction, and thus that weight and quality are calculated *per turn*. The weight term is the first occurrence of $\Sigma Q_p[i]$, which is simply the sum of the trajectories going through the ith turn. The quality term $\max(\dots)$ as it appears in (1) should be interpreted as follows: for this turn we take the sum of bundle sizes going through it *except* the largest, and subtract this value from the size of this turn's largest bundle. If this value is larger than zero, we say that the largest bundle is *dominating* and consider the resulting value a penalty for the weight term. If it is smaller or equal to zero we consider this turn well-behaved, meaning that for this turn we subtract nothing from the weight term. Thus, a turn in which every trajectory takes the same turn (e.g. a straight road) has the lowest possible quality because the quality term completely negates the weight of that turn. To illustrate this idea of quality, consider the turn with blue and red trajectories from Figure 4. Even though the red bundle in the junction on the right has more trajectories than on the left, the combined size of the other bundles (in this case the single blue bundle) limits its contribution. In this example, it results in equal measurement for both junctions.

The previous example showed that one interpretation of the quality term is that it limits the influence of large bundles. Because we consider each trajectory twice (once for each of its turns), the quality term can also be interpreted to favour connectivity. To demonstrate this, we explicitly give the junction matrices for the junctions in Figure 5. Note that both junctions have equal weight, but the trajectories are differently distributed. In both junctions, let the turn with blue and red trajectories have index 1, and clockwise the others 2 and 3.

$$Q_p = \begin{bmatrix} 0 & 3 & 3 \\ 3 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix} \qquad Q_q = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}$$

Evaluating $\Sigma Q[i] - \max(2 \cdot \max_{1 \leq j \leq m} Q_p[i,j] - \Sigma Q_p[i],\, 0)$ for every column gives $\{6, 0, 0\}$ and $\{4, 4, 4\}$ respectively. Summing these values gives

$$J(Q_p) = 6 < 12 = J(Q_q)$$

This shows that lower connectivity yields a lower overall quality. Therefore, between junctions with equal weight, their quality determines the more significant junction. Note that an "intermediate" form of the junctions in Figure 5, where for example one blue trajectory connects the topmost turns, also has an intermediate measure of 8. Thus, with fixed weight, the quality provides a reasonable linear order on junctions.

Lastly a small note on the influence of the number of turns on the measure. Junctions can have a different number of turns, typically a small constant like three or four. It is not clear however if a specific number of turns is more important, and if so how much more important. Arguably more turns contain more structure and should be considered more important, but intuitively this argument does not hold for arbitrarily large numbers of turns. Because of this we chose minimise the influence the number of turns has. By the way it is defined, the measure is not greatly influenced by the number of turns, though it somewhat favours more turns. The reason for this is that the quality term can more easily become zero with more turns, which is most noticeable between three and four turns. With three turns a junction needs perfect distribution of trajectories to have no dominating bundle in each turn, while with four or more it becomes much easier to lack dominating bundles in the turns.

3.2.3 Plane partition

In this section we describe when two points define essentially the same junction. We note that for "almost all" points p in the plane, points sufficiently close to p define the same turns and bundles. Intuitively this means that "moving" those points a little does not influence how the turns and bundles are determined. We show that it is therefore sufficient to measure a finite set of points which represent the whole plane in this manner. The shape of the areas represented by a single point, are determined by the shape of the trajectories, the shape of the tentative junctions, and the values of the two parameters λ and ϵ. We show that together they define a partition of the plane, such that all points in the same cell of the partition define the same junction.

For the sake of simplicity, we switch from circular tentative junctions to squares with sides of length λ. This makes the explanation of the partition simpler because we limit ourselves to straight line segments. By switching to squares, we introduce some minor technical changes in previously introduced notation. Firstly in the definition of ϵ-adjointness; instead of arc length, we generalise to length of a contiguous subset of ∂D_p. Secondly, since the time element of a trajectory is not relevant for the definition of a junction, we consider $\mathcal{T}$ to be a set of directed polylines.

$\mathcal{T}$-equivalence. The junction measure of a point is based on the turn clusters induced by the salient trajectories around it. When these properties are

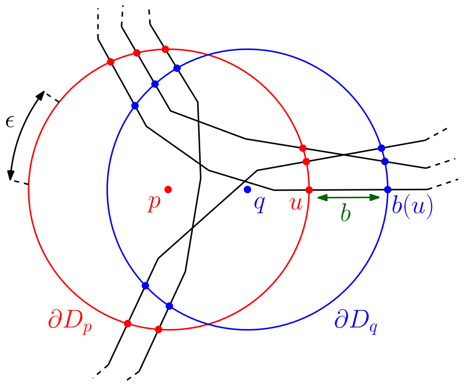

Figure 6: The linear closure of Sal_p and Sal_q are the same and indicated by the solid black line segments. The green arrow shows how the one-to-one correspondence b between intersection points on ∂D_p and ∂D_q is naturally induced.

the same for two points, their measure will consequently be the same as well. Based on this observation, we define an equivalence relation on points under which the junction measure is invariant.

Definition 5. Given a set of trajectories $\mathcal{T}$, two points p and q are $\mathcal{T}$-equivalent if the following conditions hold:

(i) Sal_p and Sal_q are equivalent up to linear closure:

$$\{T^+ : T \in \mathrm{Sal}_p\} = \{T^+ : T \in \mathrm{Sal}_q\}$$

This induces a natural one-to-one correspondence between startpoints of trajectories from Sal_p and Sal_q, and similarly for endpoints.

(ii) Given two intersection points $u, v \in \partial D_p$ and the mapping b based on the above correspondence, then $u \frown_\epsilon v$ if and only if $b(u) \frown_\epsilon b(v)$.

Figure 6 shows an example configuration $\mathcal{T}$ with two $\mathcal{T}$-equivalent points. The correspondence between Sal_p and Sal_q given in condition (i) can be recognised in the figure by the fact that ∂D_p and ∂D_q intersect the same line segments. The correspondence between trajectories naturally translates to a one-to-one correspondence between the intersections on ∂D_p and ∂D_q, as shown by the green arrow. Condition (ii) then states that this one-to-one correspondence preserves clustering. This means that p and q have the same turn matrix, which directly implies that p and q have the same measure.

It is easy to see that this relation is in fact an equivalence relation. It is trivially reflexive, symmetry follows from the one-to-one correspondence in condition (i) and (ii), and transitivity immediately follows by composing the correspondences. Since it is an equivalence relation, it partitions the plane into sets of equivalent points. We call a connected component of an equivalence class a *cell* of the partition. Earlier we stated that for most points, a sufficiently small area around them contains only points which are measured the same. These points are said to lie in the *interior* of a cell. Conversely, there are points for which every neighbourhood around them contains at least two points which are not

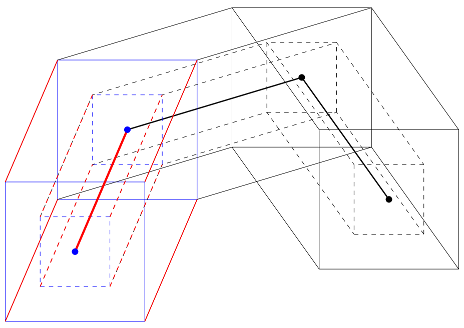
Figure 7: A vertex-critical arrangement of a polyline P. Red and blue line segments are due to segments and vertices of P respectively. Solid and dashed segments are due to D_p and d_p respectively.

equivalent; these are called *critical* points and lie on the *boundary* of cells.

We show that a point is critical if it has a specific property related to condition (i) or (ii). Since condition (ii) depends on (i), we first describe the critical points based on condition (i). The reason for this is that critical points for condition (ii) are based on salient trajectories, and not just any trajectory. Therefore if we fix a set of points equivalent under condition (i), this subset of points share the same salient trajectories (up to linear closure). We can then ignore all non-salient trajectories and subsequently describe the subset of critical points induced by condition (ii). Since this subtle point is not a problem, it is ignored in the description of the critical points.

Vertex-critical line segments. Condition (i) of $\mathcal{T}$-equivalence states that up to linear closure, Sal_p and Sal_q are equivalent. Firstly this means that d_p and d_q intersect the same salient trajectories. Secondly, as seen in Figure 6, D_p and D_q intersect the same line segments of $\mathcal{T}$.

Now imagine that we move p, then these properties can change in one of two ways accordingly. ∂d_p crosses a vertex in $\mathcal{T}$, which means a trajectory can lose its salience. Similarly, when ∂D_p crosses a vertex it intersects a different segment. Alternatively, the same changes can happen when a vertex from D_p or d_p crosses a line segment of $\mathcal{T}$. Such changes occur on points with the following *critical conditions*:

Definition 6. Given a polyline P, a point p is called vertex-critical if at least one of the following holds.

- There is a vertex of ∂D_p or ∂d_p which coincides with P
- There is a vertex of P which coincides with ∂D_p or ∂d_p

The arrangement of all vertex-critical points is simply the trace of p acquired by moving the vertices of ∂D_p (and similarly d_p) along $\mathcal{T}$, and additionally moving the edges of ∂D_p (and d_p) along the vertices of $\mathcal{T}$. Figure 7 shows an example of such a vertex-critical arrangement.

ϵ-critical line segments. Consider a cell of the vertex-critical arrangement. Moving a point p around in such a cell does by construction not change Sal_p. It can however change the ϵ-adjointness of two intersection points between Sal_p and D_p, and thus influence the turn clusters. Because clusters are based on the transitive closure of ϵ-adjointness, not all changes in ϵ-adjointness are relevant. Only when two *adjacent* intersection points lose or gain ϵ-adjointness do the clusters change. Two intersection points are adjacent if there is no intersection point strictly between them. Thus, the critical conditions for property (ii) are easily defined as follows:

Definition 7. Given a set of polylines $\mathcal{T}$, a point p is called ϵ-critical if two adjacent intersection points on ∂D_p lie exactly ϵ apart (along ∂D_p).

Though the definition of ϵ-critical line segments is simple enough, Figure 8 illustrates that they appear to be more complex than vertex-critical segments. For reference, it also shows vertex-critical and intersection-critical line segments; the latter are analogous to vertex-critical segments but based on intersections instead of vertices. By construction this arrangement of ϵ-critical segments has the following property: when the center of a square (the size of the blue intersection-critical line segments) is placed anywhere on an ϵ-critical segment, then $\mathcal{T}$ "cuts" the boundary of the square in contiguous pieces of which at least one has length exactly ϵ. An example placement p is shown, with corresponding dashed purple part of the square's boundary with length ϵ.

Let u and v be adjacent intersections points between trajectories and ∂D_p. There are three ways such a pair can be formed, each describing a different kind of ϵ-critical segment:

i u and v lie on the same edge of ∂D_p. This defines line segments parallel to edges of ∂D_p; in Figure 8 recognised as red line segments parallel to the purple ϵ sized line segments. These line segments have length $\lambda - \epsilon$ because the edge of ∂D_p of length λ must always overlap the same two points which lie ϵ apart.

ii u and v lie on the same line segment of some trajectory but on different edges of ∂D_p. This creates line segments parallel to the trajectory's line segments, appearing in the figure as parallel red and green line segments.[1]

iii u and v do not lie on the same line segment of some trajectory and lie on different edges of ∂D_p. This results in all the red line segments which connect the former two kinds.

Final arrangement and its complexity. The partition induced by this notion of $\mathcal{T}$-equivalence can thus explicitly be described in terms of vertex- and ϵ-critical line segments, denoted C_v and C_ϵ respectively. Because every two points in the same cell of the *critical arrangement* $\mathcal{A}(C_v \cup C_\epsilon)$ have identical measure, we abstract over individual points. Instead of considering the junction measure of a single point, we consider the measure of a complete cell. As such, it is

[1]The distance between them depends on the shape of ∂D_p and the slope of the trajectory's line segment

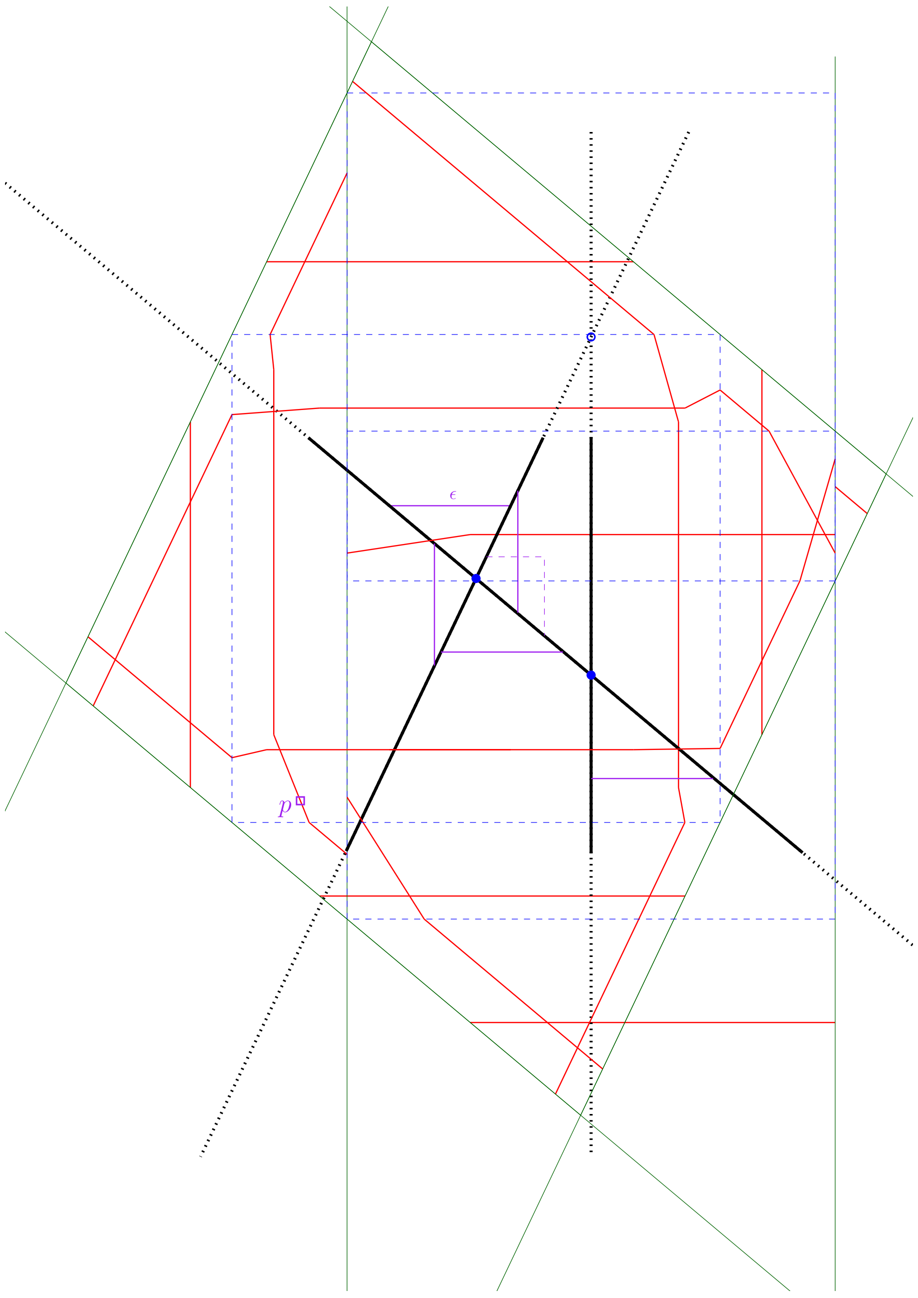

Figure 8: A set of black polylines $\mathcal{T}$ with an arrangement of red ϵ-critical segments. For reference there are green vertex-critical segments, blue intersection-critical segments and purple line segments with length ϵ.

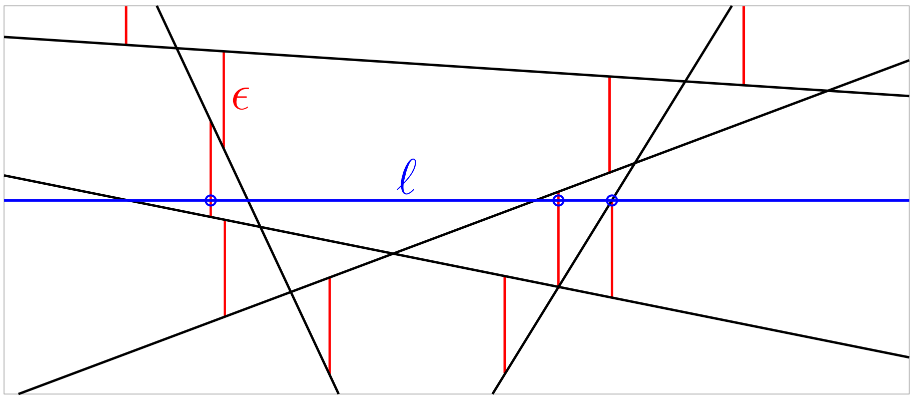

Figure 9: Any line ℓ intersects at most two red ϵ-pieces per cell in the arrangement of lines.

interesting to know how many cells the critical arrangement can have in terms of the size of the input. The size of $n = |\mathcal{T}|$ is the total number of line segments in the polylines combined. Recall that the parameters λ and ϵ are assumed to be fixed and thus not dependent on n. We prove that under this assumption, the number of cells in $\mathcal{A}(C_v \cup C_\epsilon)$ is bounded by $\mathcal{O}(\alpha(n)n^3)$. We demonstrate by a counterexample that this assumption is very important; it shows that the complexity of $\mathcal{A}(C_v \cup C_\epsilon)$ is $\Omega(n^4)$ if λ and ϵ can be chosen freely.

To achieve the $\mathcal{O}(\alpha(n)n^3)$ bound with constant ϵ and λ, we show that any line through the critical arrangement intersects at most $\mathcal{O}(\alpha(n)n)$ line segments. Since there are a constant number of critical line segments per pair of line segments from the input, there are at most $\mathcal{O}(n^2)$ ϵ-critical segments. Combined, this gives the desired upper bound.

We prove this property separately for vertex-critical segments and ϵ-critical segments of all three types, so let C_v, C_{i}, C_{ii}, and C_{iii} denote the four types of line segments respectively. Since there are only $\mathcal{O}(n)$ segments in C_v, any line through it trivially intersects at most a linear number of segments. For the ϵ-critical line segments we first assume they are derived from a set of lines instead of line segments and prove the desired property. Afterwards we show how to alter the proofs such that they also work with line segments. As noted before, the ϵ-critical lines are based on salient trajectories, but here we prove upper bounds for the general case where all trajectories can induce ϵ-critical lines. Since the arrangement based on salient trajectories is a refinement of the general case this gives no problems for the upper bounds.

Lemma 1. Given C_{i} based on a set of lines $\mathcal{L}$, an arbitrary line intersects at most $\mathcal{O}(n)$ line segments of C_{i}.

Proof. Because there is only a constant number of directions of line segments in C_{i}, it is sufficient to prove the lemma for the case where all line segments in C_{i} are parallel. Every segment in C_{i} is derived from a hypothetical line segment of length ϵ with endpoints in $\mathcal{L}$, called an *ϵ-piece* (see Figure 9). Now consider all the segments an arbitrary line ℓ through C_{i} intersects. Because all line seg-

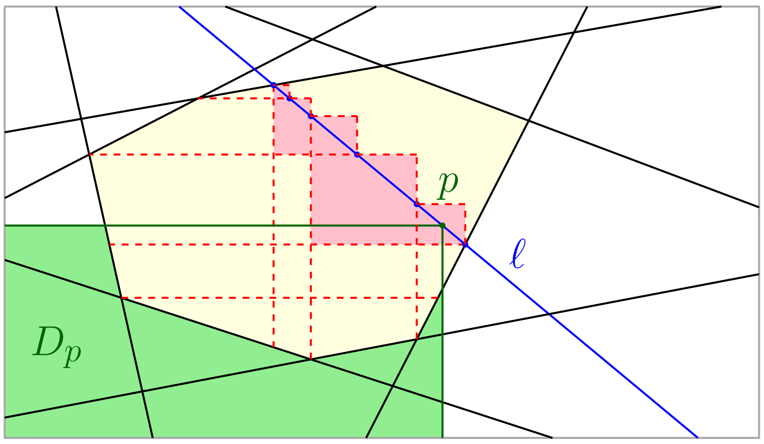

Figure 10: Per indicated red rectangle, there is at most one placement of p on ℓ such that the part of the boundary of D_p contained in this cell has exactly length ϵ.

ments of type i have length $\lambda - \epsilon$, the ones intersecting ℓ are all contained in a strip around ℓ with width 2λ. Similarly in $\mathcal{L}$, there is such a strip containing all corresponding ϵ-pieces. Now consider $\frac{2\lambda}{\epsilon}$ equally spaced parallel lines in this latter strip and note that all ϵ-pieces in the strip are intersected by at least one of these lines. We further note that any line through $\mathcal{A}(\mathcal{L})$ intersects at most n cells, and there are at most 2 ϵ-pieces in a cell. Therefore, the $\frac{2\lambda}{\epsilon}$ lines intersect at most $\mathcal{O}(n)$ ϵ-pieces, and hence ℓ intersects at most $\mathcal{O}(n)$ segments. □

Lemma 2. Given C_{ii} based on a set of lines $\mathcal{L}$, an arbitrary line intersects at most $\mathcal{O}(n)$ line segments of C_{ii}.

Proof. For any $s \in \mathcal{L}$, the critical placements of type ii induced by this line all lie at the same distance from s, and are therefore collinear. Any line through C_{ii} thus intersects at most 1 critical placement per $s \in \mathcal{L}$. □

Lemma 3. Given C_{iii} based on a set of lines $\mathcal{L}$, an arbitrary line intersects at most $\mathcal{O}(n)$ line segments of C_{iii}.

Proof. Every ϵ-critical segment from C_{iii} is induced by a pair of edges from ∂D_p. Since it has only a constant number of edges, it is sufficient to prove this lemma for a fixed pair. Let D_p be parametrised from the intersection of this pair, and ℓ be an arbitrary line intersecting the arrangement $\mathcal{A}(\mathcal{L})$. Consider an arbitrary cell (with k edges) of the arrangement, cut in half by ℓ as in Figure 10. For one of the halves (and repeat the argument for the other half), take the horizontal and vertical decomposition as in the figure. ℓ intersects at most $\mathcal{O}(k)$ of the new cells induced by both decompositions. By construction, every such new cell contains at most one ϵ-critical segment of type iii. Therefore, ℓ intersects at most $\mathcal{O}(k)$ critical segments per cell it intersects; and by the zone theorem, ℓ intersects at most $\mathcal{O}(n)$ critical segments in total. □

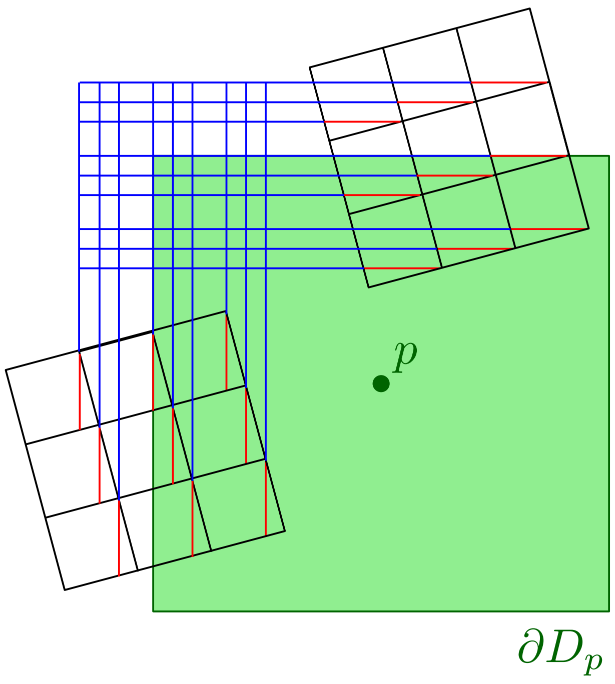

Figure 11: Each red ϵ-piece induces an ϵ-critical segment similar to the blue line segments. Because of the placement of the grids in relation to the shape of ∂D_p, the arrangement of the ϵ-critical line segments form $\Omega(n^4)$ cells.

The proof for Lemma 2 does not change if line segments are used. However, for Lemmas 1 and 3 we use that a cell in an arrangement of lines has complexity $\mathcal{O}(n)$, which does not hold for cells in an arrangement of line segments.

For Lemma 1 we make two minor alterations in the proof. The proof is already for parallel line segments, and we further assume without loss of generality that they are horizontal. Then, instead of considering $\mathcal{A}(\mathcal{L})$ we first take its horizontal decomposition. The same arguments now hold, except that a line through this horizontal decomposition can intersect $\mathcal{O}(\alpha(n)n)$ cells, where $\alpha(n)$ is the inverse Ackermann function. For Lemma 3 the same arguments still hold, but the bound changes to $\mathcal{O}(\alpha(n)n)$ similarly. To summarise the results we give our main theorem:

Theorem 1. Given a set $\mathcal{T}$ of polylines, the critical arrangement $\mathcal{A}(C_v \cup C_\epsilon)$ contains at most $\mathcal{O}(\alpha(n)n^3)$ cells.

Proof. Above arguments show that any line through the vertex-critical arrangement and the ϵ-critical arrangement intersect at most $\mathcal{O}(\alpha(n)n)$ line segments. In particular, every segment in $\mathcal{A}(C_v \cup C_\epsilon)$ intersects at most $\mathcal{O}(\alpha(n)n)$ other segments. Since there are $\mathcal{O}(n^2)$ critical segments, there are at most $\mathcal{O}(\alpha(n)n^3)$ intersections between segments and similarly at most $\mathcal{O}(\alpha(n)n^3)$ cells in the arrangement. □

Free choice of λ and ϵ To demonstrate the importance of constant λ and ϵ for our previous result, we show by a counterexample that with free choice of these parameters the complexity of the arrangement can be much larger. For the worst case example, fix $\epsilon = 1$ and let $\lambda = 2n$ for some positive integer n. Note that the argument from Lemma 1 that $\frac{\lambda}{\epsilon}$ is constant does not hold. Now create a regular $n \times n$ grid with cells of size 1×1 and rotate it a little such that each cell contains one ϵ-piece and no two ϵ pieces are collinear. Then, by placing two such grids in the right orientation along two non-parallel edges of D_p, every ϵ-critical segment from one of the grids intersects with every ϵ-critical segment induced by the other. Figure 11 shows an example of such a construction. Since there are n^2 cells in the grids, and every cell induces an ϵ-critical segment which intersects with n^2 other segments, the total number of cells in the critical arrangement is $\Omega(n^4)$. Note that to describe the construction in Figure 11 we used line segments and not trajectories. If the individual line segments are considered trajectories, then not all of them are salient. This is however easily fixed by adding a few extra line segments, so the argument still holds for a construction with salient trajectories.

3.2.4 Algorithm

In this section we present an algorithm to compute the junction with the highest measure in a set of trajectories. We shortly recapitulate the problem and then describe the separate parts of the final algorithm, together with an analysis of their respective running time.

Problem definition. For a set of trajectories $\mathcal{T}$, a scale $\lambda > 0$, and an $\epsilon > 0$, every point in the plane defines a *junction* consisting of *turns* and *bundles* captured in the junction matrix of Definition 3. $\mathcal{T}$-equivalent points define the same turns and bundles, and thus the same junction. This equivalence subdivides the plane into contiguous regions with $\mathcal{T}$-equivalent points, called *junction areas*. The problem at hand is to calculate the all the junction areas with their measure, such that we can determine the area of the junction with the highest measure.

The algorithms consists of two parts. The first part is to compute all junction areas based on the critical arrangement described in Section 3.2.3. The second part consists of the routine to compute the measure of a junction area. These two parts are then combined by computing the measure of each junction to determine the highest value.

Algorithm: ComputeJunctionAreas. This algorithm simply involves explicitly determining all vertex-critical and ϵ-critical line segments, and then calculating their arrangement by a standard technique [11].

The vertex-critical line segments are trivially determined as they are simply translated copies of the trajectories or copies of ∂D_p centered at breakpoints in the trajectories. The ϵ-critical segments of type i can be determined by doing a brute force calculation to determine all potential ϵ-pieces between trajectories,

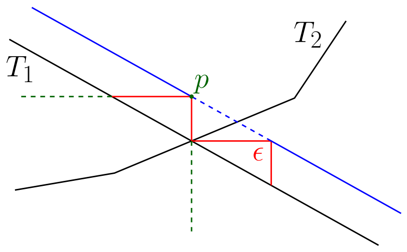

Figure 12: The blue ϵ-critical line segment associated with T_1 has a hole induced by T_2.

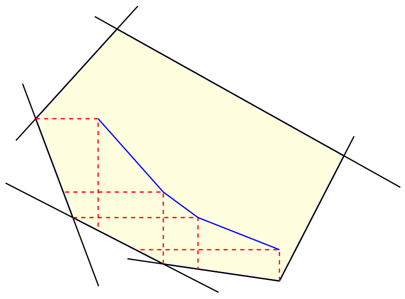

Figure 13: For convex cells, ϵ-critical line segments are found by scanning the boundary of the cell.

and eliminating those intersecting a line segment. Given an ϵ-piece which does not intersect anything, the corresponding critical segment of length $\lambda - \epsilon$ is parallel to it at a distance $\frac{\lambda}{2}$. The ϵ-critical segments of type ii are essentially copies of the trajectories, but with some "holes" induced by another segment as in Figure 12. We first calculate the "uninterrupted" critical segment, and then by brute force check for each trajectory segment if it creates a hole.

For the critical segments of type iii we first calculate the horizontal and vertical decomposition for every non-convex cell. Every new sub cell induced by the decompositions thus has two associated line segments, one for the horizontal and one for the vertical decomposition. With this information we calculate if the cell contains an ϵ-critical segment. For convex cells we, we scan the boundary of the cell for each corner of ∂D_p as demonstrated in Figure 13.

Running time. For the segments of type i, we first calculate $\mathcal{O}(n^2)$ potential ϵ-pieces and check these for intersections with $\mathcal{O}(n)$ line segments. Thus, we find the segments of type i in $\mathcal{O}(n^3)$. For type ii, we calculate the "uninterrupted" critical segment for all n trajectory segments per corner of ∂D_p. For each uninterrupted segment, we traverse all the trajectories' segments to determine its possible holes. Determining a single hole can be done in constant time. Since there are at most $\mathcal{O}(n^2)$ holes combined, we find the segments of type ii in $\mathcal{O}(n^2)$ time.

For the segments of type iii, we can calculate the sub cells induced by the horizontal and vertical decompositions for every non-convex cell. There are $\mathcal{O}(n)$ non-convex cells, and an individual non-convex cell can have complexity of $\mathcal{O}(\alpha(n)n)$, yielding $\mathcal{O}((\alpha(n)n)^2)$ sub cells. However, k endpoints inside a non-convex cell create at most a factor $\mathcal{O}(\alpha(k))$ additional sub cells – compared to a convex cell which has at most $\mathcal{O}(n^2)$ sub cells. Since there are only $\mathcal{O}(n)$ endpoints, the combined number of sub cells they induce is at most $\mathcal{O}((\alpha(n)n)^2)$. Thus, disregarding the sub cells induced by endpoints, each non-convex cells each with complexity $\mathcal{O}(n)$ can still induce $\mathcal{O}(n^2)$ sub cells. Per sub cell we use only 2 line segments to determine if it contains an ϵ-critical segment, thus finding all ϵ-critical line segments in non-convex cells in $\mathcal{O}(n^3)$ time. For convex cells, we scan the boundary once per corner of ∂D_p. There are $\mathcal{O}(n^2)$ convex cells, thus we find the remaining ϵ-critical line segments in $\mathcal{O}(n^3)$.

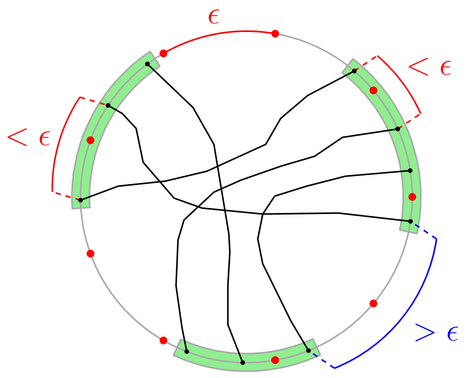

Figure 14: ∂D_p is divided in nine arcs of length ϵ. The final turn clusters are indicated in green.

Algorithm: ComputeJunctionMeasure. Given a point p and a set of trajectories $\mathcal{T}$, we need to calculate the measure of the junction defined by this point. The four steps that comprise this algorithm are: determine salient trajectories, determine turns, calculate the size of each bundle, and calculate the measure.

The first step involves checking for intersections between trajectories and the boundaries of D_p and d_p. After this the salient trajectories are easily determined, as well as their intersections with ∂D_p. Suppose for now that we have determined the m turn clusters, then we determine the bundle sizes by iterating through all salient trajectories and increasing the correct cell of the $m \times m$ turn matrix. Finally, the measure is trivially evaluated from this matrix.

The hardest step in this algorithm is to determine the turns. Let C be the set of intersections between the salient trajectories and ∂D_p that were found after the first step. To find the turns we need to know for every pair of adjacent points in C if their distance is more or less than ϵ. To do this, we divide the boundary of D_p in pieces of length ϵ as in Figure 14 (one piece can be smaller). Every such piece is a *bin*, and we determine for each $c \in C$ to what bin it belongs. During this process, we keep track of the (at most) two points lying closest to the edge of the bin. Once all points are binned, we know that points in the same bin belong to the same turn. Finally, we check the distance between the closest points in all adjacent bins; if this distance is smaller than ϵ, then all points from these bins belong to the same cluster.

Running time. We analyse the running time of the four steps separately. Let n be the number of line segments in $\mathcal{T}$.

1. Every segment in $\mathcal{T}$ is checked for intersecting one of the four sides of ∂D_p or ∂d_p. A trajectory might produce $\mathcal{O}(n)$ salient trajectories, so we traverse every trajectory once to determine the correct start and endpoints for the salient trajectories. Thus we find the salient trajectories in $\mathcal{O}(n)$ time.

2. Since ϵ and λ are constants, only a constant number of bins are created. Therefore checking in which bin a point belongs can be done in constant

time. Keeping track of the points nearest to the boundary of the bin also does not depend on n. Finally we merge a constant number of bins to determine the turns. Since there are at most $\mathcal{O}(n)$ points in C, this step runs in $\mathcal{O}(n)$ time.

3. There are at most a constant number of turns, and hence the turn matrix has constant size. We iterate through $\mathcal{O}(n)$ salient trajectories while updating the turn matrix, taking $\mathcal{O}(n)$ time.
4. The turn matrix is of constant size, therefore determining the measure can be done in constant time.

Since all steps are sequential and every step takes at most linear time, the junction measure can be calculated in linear time as well.

Algorithm: ComputeBestJunction. We compute the critical arrangement by running ComputeJunctionAreas for the critical line segments, and a standard algorithm to compute their arrangement. Next we run ComputeJunctionMeasure on each cell in the arrangement and keep track of the best cell.

Running time. Since the critical arrangement has $\mathcal{O}(\alpha(n)n^3)$ cells, and per cell we spend $\mathcal{O}(n)$ time, we find the best junction in $\mathcal{O}(\alpha(n)n^4)$ time.

3.3 Stops

The other pattern we study are *stops*, a feature in a trajectory which corresponds to the respective moving object coming to a halt. We are interested in areas where this pattern occurs frequently, so we can eventually indicate these locations in our visualisation. As with the previous pattern, we present a method to assign each point in the plane an importance value: higher values indicate more significant constellations of stop patterns. For this we first formalise a notion of stop patterns, and discuss how significant a single occurrence is in the presence of others – particularly stop patterns originating from the same trajectory. Then we describe a simple method to assign importance values to points in the plane. Finally we give two algorithms to efficiently compute each step.

3.3.1 Formalisation

In an ideal trajectory, a stop is easily recognised as it are exactly those points where the velocity is zero and one would only need a duration threshold to decide which stops are considered significant enough. However, this notion of stops is not robust with respect to small movements. This is especially undesirable since real world data is far from being ideal and will most certainly introduce artifacts which look like minor movements. For this reason, instead of considering the velocity, we look at the trajectory's path. More specifically, we consider a sub trajectory to be a stop pattern if its path remains inside a

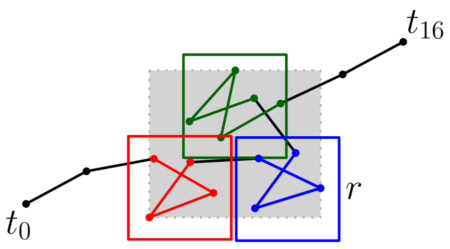

Figure 15: A single trajectory (segments with duration 1) with three stop patterns indicated. The parameter r is indicated and $d = 3$. The grey area indicates an ambiguous stop pattern.

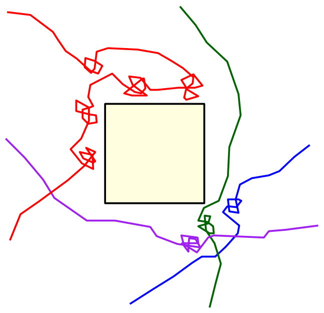

Figure 16: The southeast corner of the square is a more important stop area than the northwest, because it contains stops from *different* trajectories.

small enough region for a long enough period of time:

Definition 8. Given a trajectory T, a size r and a duration d. A sub trajectory $T[t, t+d]$ is a *stop* if there exists a point p such that $\mathrm{Path}(T[t, t+d])$ is contained inside the square centered around p with sides of length r. We call t the *halt* and $s \in [t, t+d]$ a *rest* of the stop.

For reasonable choice of d and r, this definition certainly encompasses all intuitively identifiable stops. More important however, are the halting moments meeting this definition which are not so easily recognised as a stop. A simple example is a straight path of length larger than r with constant velocity just below $\frac{r}{d}$; it is not clear to what degree this should be counted as a stop, even though it does satisfy our criteria for a stop. A similar issue arises with "overlapping" stop patterns as demonstrated in Figure 15. Though these are arguably multiple stops, it is unclear whether these should be considered distinct. To confine these ambiguities we extend on our definition of a stop: a sub trajectory $T[t, t']$ is called a *maximal stop pattern* if every $s \in [t, t']$ is a rest of some stop, and no interval strictly containing $[t, t']$ also consists only of rests. Such a maximal pattern is a contiguous part of a *single* trajectory which can be completely covered by one or more stops. This means that it is either a single neatly defined stop, or a succession of overlapping stops spanning an arbitrary area. Reflecting on this, we argue that a variety of maximal stop patterns is more important than a few isolated bulky maximal patterns. This argument is illustrated by Figure 16, one person stopping multiple times to admire a statue from different directions is less significant than three people admiring the popular view. We therefore define the *stop number* of a point in terms of unique maximal stop patterns in some region:

Definition 9. For a set of trajectories, let M be the set of all its maximal stop patterns. The *stop number* of a point p is the size of the largest subset $N \subseteq M$ such that: for all $T[s, s'] \in N$ there is at least one halt t with $s \leq t \leq s'$ such that $T(t + \frac{d}{2})$ lies within D_p.

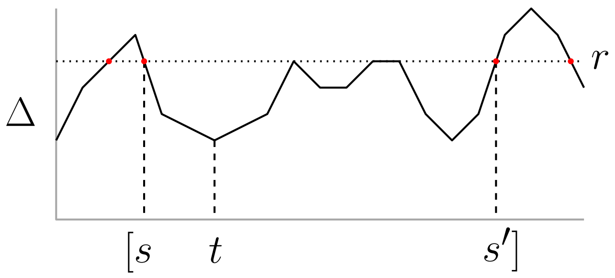

Figure 17: The graph of Δ and the constant function r with proper intersections indicated in red.

Note that we consider the point $T(t + \frac{d}{2})$ and not the point corresponding to the moment of the halt itself. The definition of a halt is convenient, it being the starting moment of a stop pattern makes it easy to reason with. However, to prevent bias in time, we identify a stop by the point lying exactly halfway through its duration.

3.3.2 Algorithm

We present an algorithm that takes a set of trajectories as input, and outputs the stop number of each point in the plane. The output is an arrangement of line segments such that for each cell it holds that every point in that cell contains the same maximal stops in its vicinity. This algorithm consists of two parts. First, given a single trajectory, determine the halts of all its stops; and second, given the halts, determine the maximal stops to calculate the planar subdivision described above.

The idea for the first algorithm is to "slide" a sub trajectory of duration d along a trajectory T and monitor the bounding box of its corresponding path. Let $\max_x(s)$ denote the maximal x-coordinate of $\mathrm{Path}(T[s, s+d])$, and $\min_x$, $\max_y$, and $\min_y$ be similar notations. Δ is a function denoting the largest dimension of the bounding box of $T[s, s+d]$:

$$\Delta(s) = \max(\max_x(s) - \min_x(s), \max_y(s) - \min_y(s))$$

We prove that halts occur in closed intervals[2], and use this to describe the first algorithm.

Lemma 4. Given a trajectory T, for every halt t in T there is an interval $[s, s']$ containing t such that the following condition holds. For a sufficiently small $\delta > 0$, $t' \in [s - \delta, s' + \delta]$ is a halt if and only if $t' \in [s, s']$.

To make this statement valid if s or s' are respectively the start or endpoint of T, a moment is never a halt if it lies outside the trajectory's time span.

Proof. Let t be a halt in T. Because T is a continuous piecewise linear function, Δ is continuous. Consider the proper intersections between the graph of Δ and

[2]This includes the degenerate case, a single point

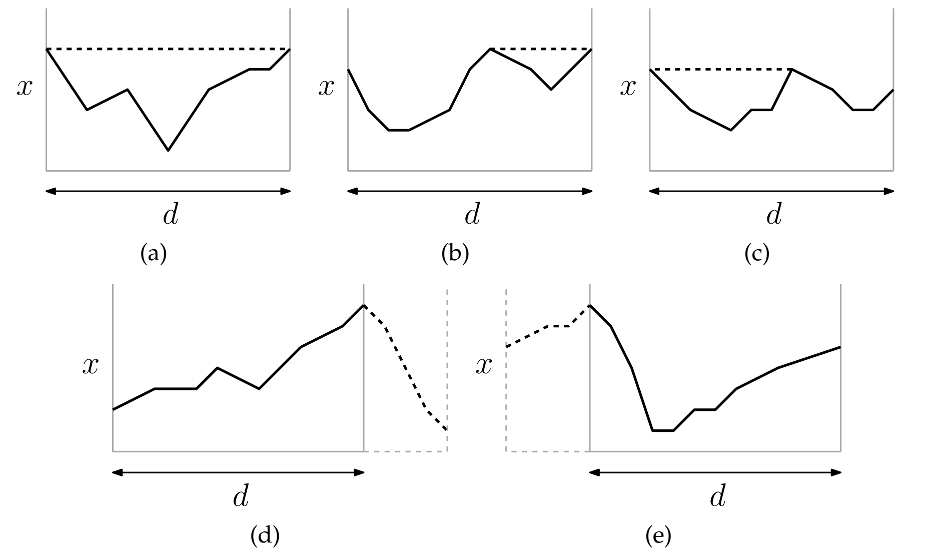

Figure 18: Different transitions of the highest x value when sliding a slab of width d to the right.

the constant function r as in Figure 17. Two adjacent intersection points s and s' either define an open interval without halts, or a closed interval containing only halts. Since t is a halt, $\Delta(t) \leq r$ and thus t lies in a closed interval $[s, s']$ containing only halts.

□

It suffices to calculate the points s where $\Delta(s) = r$ to determine all intervals containing only halts, or *halting intervals*. Note that not all moments s where $\Delta(s) = r$ is a boundary of a halting interval, s can be a local maximum/minimum or lie on a constant part of Δ. However these degeneracies are easily checked for, so from now on we assume they do not occur.

To construct Δ we need four functions which are constructed analogously, so let us consider the case of $\max_x$. As it is a continuous piecewise linear function, determining all its breakpoints suffices. Let $T_x[s]$ denote the x-coordinate of the trajectory at time s. The graph of this function looks like a polyline as in Figure 19. We slide an interval of size d along time-axis and keep track of the highest point. For a certain interval $[s, s+d]$ we distinguish three scenarios: either one of the boundary points of the interval determines the highest point, or an internal point does. Between these three, there are five ways a *transition* – or breakpoint – can occur:

1. The endpoint becomes the highest point, replacing the startpoint.
2. The endpoint becomes the highest point, replacing an internal point.
3. The startpoint is replaced by an internal point
4. The endpoint reaches a local maximum and becomes an internal point
5. An internal point reaches the start of the interval and becomes the startpoint

Figure 18 shows these five transitions respectively. Note that with three states,

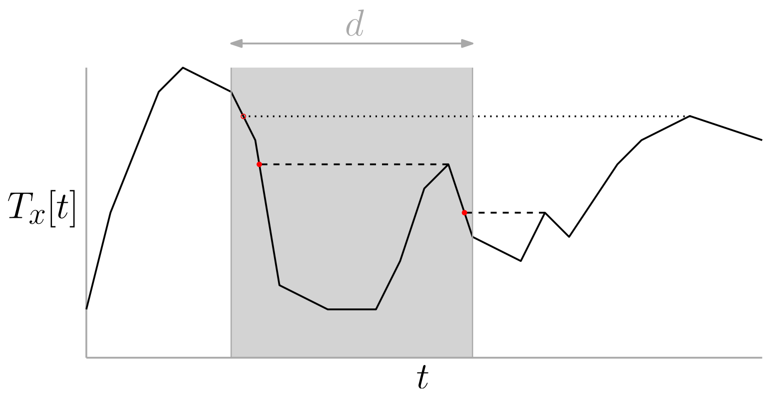

Figure 19: The x-coordinate of a trajectory is plotted against time. By sliding an interval of length d along the plot, we find the maximal x value for every such interval. Only two out of three projection points generate an event, since the dotted line segment is larger than d.

in theory there can be six transitions. However the transition from endpoint to startpoint being the highest point is impossible. It cannot occur because if the endpoint is highest in the interval, then the rest of the interval is lower and thus the startpoint cannot become higher.

When sliding the interval we keep track of the height of the start and endpoint, and if it exists the highest internal point. With this it is easy to handle the transitions in case 1 and 2. Because we can easily determine local maxima, case 4 is also no problem. For case 5 we determine when the internal point becomes the startpoint. Case 3 is the least simple to handle and requires a longer explanation. Suppose this transition occurs on time s, then we know that for some local maximum t, it holds that $T_x[s] = T_x[t]$ and that for all $s < t' < t$ we have $T_x[t'] < T_x[t]$. In other words, by shooting a ray leftwards originating from the local maximum in t, we hit the exact position of the starting point when the transition occurs. Figure 19 illustrates this idea. The figure also shows that not all *projection* points are transition points, this occurs when the ray is longer than d before hitting the projection point.

To handle case 3, we first calculate all projection points. We do this by scanning the x-monotonic polyline (derived from the function T_x) from right to left while keeping a stack of encountered local maxima. Every time we traverse a new line segment we check if it contains an x value equal to the local maximum on top of the stack. If this is true, we calculate the corresponding projection point. We then pop the stack, and repeatedly check the next local maximum on the stack until it is larger than the current segment (or until the stack is empty). Afterwards, when we slide the interval and encounter a projection point event, we check if it is a transition of case 3 by verifying that the corresponding local maximum lies within the interval d. Note that a local maximum which did not become the highest point because of its projection point, can still become the highest point because of transition case 5.

With all cases handled we can now construct $\max_x$. By analogous construction using minimal value or y-coordinate we obtain the other three functions

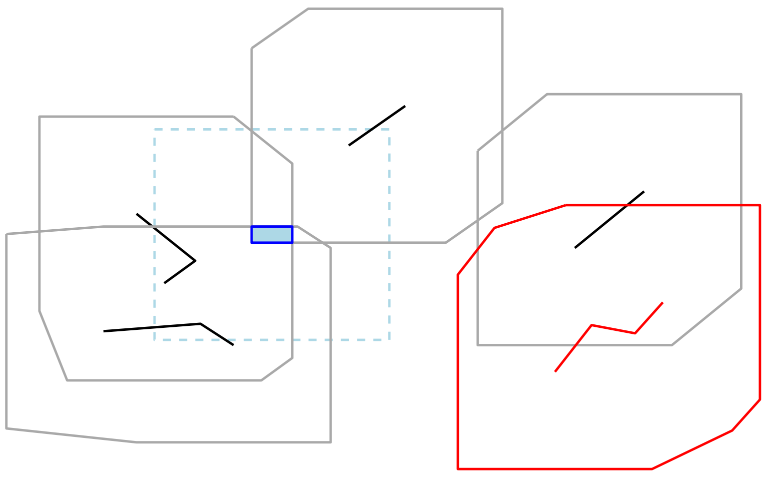

Figure 20: Five stops with their corresponding critical line segments (example correspondence shown in red). The blue area contains the most stops in its vicinity and has stop number 3, with the dashed line showing the extent of this stop area.

to construct Δ. Determining the preimage $\Delta^{-1}[H] = r$ containing the set H of boundary points of the halting intervals is now an easy task since Δ is a continuous piecewise linear function. Once the boundaries of the intervals are known, the final step is to determine if an interval contains only halting moments or none at all. To do this we evaluate a single point per interval to see if it is a halt (which can be simplified because we know the intervals alternate between containing only halts and no halts).

The second algorithm is fairly simple. The previous step identifies all halting *segments* in a trajectory. Recall that a maximal stop pattern consists of a series of such halting segments separated by at most a duration d of non-halting segments. Thus, we easily determine all maximal stop patterns by checking the time between halting segments by iterating through each trajectory. Each stop segment now induces critical segments very similar to the vertex-critical segments from Section 3.2.3 as shown in Figure 20. These *stop-critical* segments are easily calculated and we use a standard technique [11] to calculate their arrangement. Calculating the stop number of a single cell in the arrangement can be done by traversing all stops and checking if their corresponding critical segments intersect the cell. To calculate all stop numbers, it is sufficient to know how to calculate the difference between neighbouring cells. By going from one cell to a neighbour, we only need to know if we “leave” or “enter” the vicinity of a stop segment, yielding an decrement and increment of 1 respectively. Thus, by traversing all cells and keeping track of the highest stop number so far, we find the stop area with highest stop number.

Running time analysis. Since the second algorithm only depends on the output of the first step, the running times can be analysed independently. Let n be the number of line segments in a trajectory T. The function $\max_x$ contains

$\mathcal{O}(n)$ line segments as well, as every vertex from T only generates a constant number of transition events. Transition case 1 and 2 happen at most once per line segment, case 4 and 5 at most once respectively per local maximum, and in case 3 we note that every local maximum generates at most one projection point. Since Δ is a combination of four piecewise linear functions of $\mathcal{O}(n)$ size, it has $\mathcal{O}(n)$ size as well. Furthermore since it is a piecewise linear function it takes $\mathcal{O}(n)$ time to calculate H with $\Delta^{-1}[H] = r$. The final detail is the calculation of the projection points. Scanning the polyline takes $\mathcal{O}(n)$ time. We find at most $\mathcal{O}(n)$ local maxima, meaning the stack is popped and pushed at most $\mathcal{O}(n)$ times. Furthermore, every segment is compared to at most one local maximum whith larger x-coordinates. Thus, we find the projection points in $\mathcal{O}(n)$ time. In total the algorithm for finding all halts runs in $O(n)$ time as well.

For the second algorithm, we note that there are at most $\mathcal{O}(n)$ stop segments since Δ has at most $\mathcal{O}(n)$ segments. There are therefore $\mathcal{O}(n)$ stop-critical segments, and consequently at most $\mathcal{O}(n^2)$ cells in their arrangement. We build the arrangement of the critical segments in $\mathcal{O}(n^2)$ time. We then first determine the stop number in a single cell by iterating through all stop segments, taking at most $\mathcal{O}(n)$ time. After that, we successively determine the value of a neighbouring cell, which involves one subtraction or addition. After traversing all cells, all stop numbers are determined in $\mathcal{O}(n^2)$ time.

3.4 Symbol placement

In the previous sections we discussed junctions and stops in trajectories. We defined a junction measure and a stop number for points, indicating the *significance* of the respective pattern around that point. In this section we give the final visualization step which uses these measures to mark significant locations with abstract symbols.

Plotting the paths of trajectories gives a very basic visualization. When there is ample space between the lines, this is a suitable technique. However when the number of trajectories increases, or when the scale decreases, the visualization might start to obscure significant parts of itself in the form of clutter. When this is the case, visually encoding extra information in individual paths is undesirable, as this will most likely be partly obscured as well. However, as the information conveyed in cluttered areas diminishes, the space becomes available for other visualizations; particularly those that take multiple trajectories into account.

Above observation is the basis for our method; we overlay the plot with symbols providing information the underlying cluttered paths cannot. In a broad sense, the symbols are an abstract representation of information pertaining to the trajectories it (partially) covers. This is in contrast to a plot, which particularly visualises individual trajectories. We require the information to be significant in some sense to preclude symbols that obscure more information than they provide. Additionally, to prevent ambiguity in the areas between symbols, we require them to be sufficiently spaced. These criteria are intuitively reasonable but also very general. Consequently, they are not restrictive enough for a straightforward definition. It is obvious however that the significance mentioned in the first criterion derives from some significance function, and the second criterion means that we can not place the symbols independently.

To select k locations to place symbols, we propose the following simple procedure. Assume $\frac{\lambda}{2}$ is the size of a symbol, and we have an arrangement of line segments with some value per cell. We iteratively pick the cell with the highest value and return its centroid p as the next location for the symbol. Before the next iteration, all cells with centroid within a distance λ to p are discarded. Basically, we pick the k highest cells, but preclude points lying too close to previously selected points.

Running time. Let the input size n be the number of segments in the paths of the trajectories, and let the arrangement whose cells contain importance values (junction measure or stop number) have complexity $\mathcal{O}(f(n))$. Finding the cell with highest importance value takes $\mathcal{O}(f(n))$, as well as discarding all nearby cells. This procedure is repeated k times, giving a running time of $\mathcal{O}(f(n)k)$.

4

Implementation and Results

To test our models and demonstrate the proposed visualisation technique we have implemented variations on the algorithms described in the previous sections. The main difference between the proposed methods and the implementation is that we do not calculate the arrangements for the junction measure or the stop number. For simplicity, we discretise the space in a grid and calculate both values once per cell in the grid.

4.1 Junction Measure

Our implementation to calculate the junction measure in a point varies slightly from the proposed algorithm. To determine the turn clusters, we first determine and sort all the intersection points between Sal_p and ∂D_p and scan this list to determine the size of all the bundles. Because of the sorting, this approach does not scale linearly in the input size; however on the small inputs we used to test, this is negligible.

Figure 21 shows a set of trajectories overlain with a grid of numbers, which indicate the junction measure in that cell. The scale of D_p is indicated by the square in the upper left corner, and ϵ by the size of the line segment in the centre of this square. As can be seen in the figure, areas with a discernable junction receive nonzero measure. Junction *A* is by far the most significant in the picture, with a junction measure of 30.[1] The "junction" labelled *B* is not recognised as a junction; this was intentional, because trajectories do not turn here. Junctions *C* and *D* have roughly the same number of trajectories and consist of three turns, yet receive very different measure. This is because junction *C* is merely a splitting of one bundle, while the turns in *D* are nicely connected. Note that around junctions, many cells have the same junction measure. This indicates that the discretisation is fine enough and does not skew the results dramatically.

[1]See Section 4.4 for notes on the implementation of the junction measure

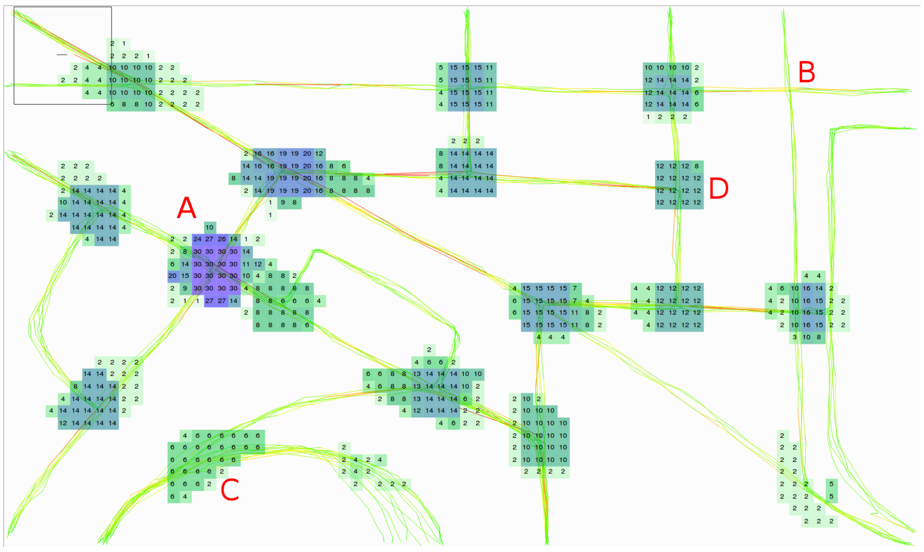

Figure 21: Idealised trajectories exhibiting various kinds of junctions. The square in the upper left corner shows the size of D_p, and the line segment in its centre has length ϵ. The numbers in the coloured squares show the junction measure in that point.

4.2 Stop Number

Two algorithms are necessary to calculate the stop number in each cell in the grid. The most important one calculates the halting moments in individual trajectories and determines all the maximal stop patterns. The other is fairly trivial as it simply counts the number of different maximal stop patterns around a point.

Again we have implemented a slight variation on the methods proposed earlier. Instead of determining all *segments* of halting moments, we determine for a discrete number of *points* on the trajectory if they are halting moments. We do this by dividing each line segment of a trajectory in equal sized sub segments – meaning that longer segments are divided in longer sub segments. For each sub segment s we then check if the shortest sequence of sub segment starting at s with combined duration at least d, is contained in a sufficiently small region. Segments for which this is true thus start with a halting moment. Note that the converse is not true, it only means that we failed to detect that halting moment. A maximal stop pattern now corresponds to sequence of halting moments lying less than a duration of d apart. Next, we create a set of points such that each point corresponds to a halting moment, and points with the same label correspond to halting moments in the same maximal stop pattern. Finally, to determine the stop number in a point, we simply check the number of unique labels in an area around it. Figure 22 shows the stop numbers of each cell in the grid.

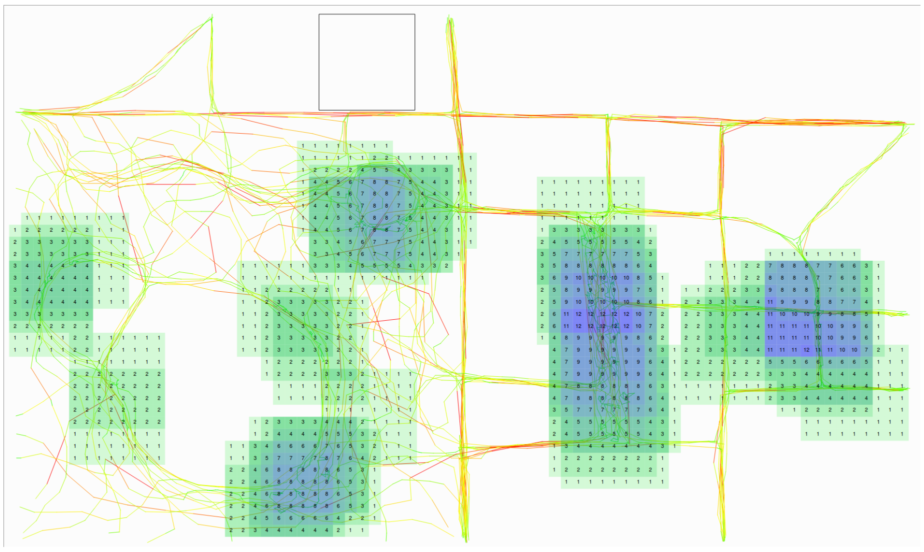

Figure 22: Green trajectories indicate low speed, but not necessarily a stop. The numbers in the coloured squares indicate the number of stops around that point. The black square indicates the extent of this area.

4.3 Point Selection

The procedure to select several significant points is almost the same as the proposed method. Instead of an arrangement with junction measures or stop numbers, the algorithm takes a grid with values as input. Also, because of the discretisation, we "merge" adjacent cells with the same value. To select k significant points, the algorithm then performs k iterations with the following steps.

- Determine the largest value amongst the cells
- Find the largest contiguous group of cells with this value
- The significant point p is the centroid of these cells
- Set all cells overlapping D_p to 0

Figure 23 shows the output of our implementation. The 'symbols' are squares with sides of length $\frac{\lambda}{2}$, thus indicating the core of each significant junction. Note that in general, two selected points lie at least λ apart. For junctions, this means that D_p can overlap, but the cores (d_p) are always disjoint.

4.4 Test Data

The data used to test the implementation was made by hand and specifically designed to contain the patterns we modelled.

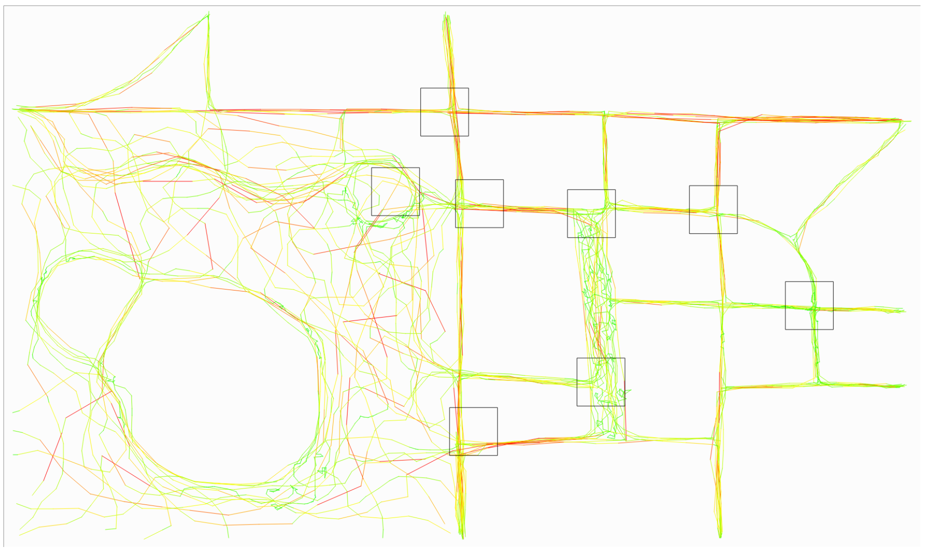

Figure 23: The 8 most significant junctions are indicated by squares with sides of length $\frac{\lambda}{2}$.

A slight variation was also made in the way the junction measure is determined. Because of the relatively small input size, clutter can be sparse and thus induce many turns. To compensate for this we subtract the number of turns in a tentative junction from its measure. By doing so we nullify turns determined by outliers. The altered junction measure J' of a turn matrix Q_p with size $m \times m$ is thus $J'(Q_p) = J(Q_p) - m$.

The results on the test data show that without noise, junctions are detected as expected. Figure 23 shows an example with deliberately added noise. In this example, the effect of the noise is apparent by the two junction shifted to the right away from it. Other tests showed that the models can handle noise up to some degree (also apparent from the single "noisy" junction in the figure), but eventually suffer too much and fail to detect junctions hidden in the noise.

5

Conclusion

With modern technologies generating large amounts of diverse trajectory data, there is increasing need for methods to analyse these. In this thesis we modelled junction and stop patterns in trajectories, and gave algorithms to automatically compute significant locations containing these patterns. We implemented a basic visualisation tool plotting the trajectories' paths, in which these locations are indicated.

To analyse the running time of the algorithm computing the junctions, we introduced a novel combinatorial geometric problem. We studied the arrangement of ϵ-critical line segments based on a set of n lines and later extended to n line segments. If λ and ϵ can be chosen freely (and thus depend on n), we showed that the number of cells in this arrangement can be of order $\Omega(n^4)$. We also proved that if the parameters are constant in the number of lines, there are at most $\mathcal{O}(n^3)$ cells. For trajectories we needed to consider line segments instead of lines, introducing an extra factor $\alpha(n)$ to the running time. We further showed that points in the same cell of this arrangement describe the same junction. Combined with an $\mathcal{O}(n)$ time algorithm to measure the quality of a junction, the presented algorithm to compute the best junction runs in at most $\mathcal{O}(\alpha(n)n^4)$ time.

We also presented an $\mathcal{O}(n)$ time algorithm to find all stops, or more specifically their starting moments called halts, in a trajectory with n segments. To find the best stop area, we studied a simple placement space with $\mathcal{O}(n^2)$ cells, also resulting in an $\mathcal{O}(n^2)$ time algorithm for finding the best stop area.

We analysed our algorithms based on worst case scenarios. As such the upper bound $\mathcal{O}(\alpha(n)n^4)$ does not appear to be very practical. However, the $\mathcal{O}(n^3)$ bound has not yet been proven optimal, and might be improved to $\Theta(n^2)$. Additionally, to make the algorithm appear more practical we can consider typical inputs instead of arbitrary trajectories. For instance, adding similar trajectories to a set of trajectories all resembling some underlying network will mainly increase the size of bundles and typically only influence a constant number of turns. Alternatively, we can simplify parts of the trajectories which are probably not relevant to junctions, and run the algorithm on the simplified input.

These are all interesting options for future research which improve the algorithm without drastic modifications.

Our implementation demonstrates the idea of enriching a simple visualisation with an abstract representation of hidden information. In our current implementation however, this idea has not reached its full potential since we used very crude symbols which only indicate the location and extent of patterns. To further improve upon this, symbols can be designed based on the information they represent; they can indicate the turning behaviour in case of junctions. With such sophisticated symbols it is also interesting to know if the visualisation enrichment is also perceived as helpful by users. Different types of existing visualisations can be used to test how well the symbols complement these. If the symbols are improved like this, their placement becomes more important. The current basic selection procedure was designed to demonstrate the concept. A better idea is to remove data already represented by symbols and rerunning the algorithms to determine the next important location.

The set of tools, or techniques, we described in this thesis were deliberately designed to be fairly independent. This means that isolated parts can be separately tested, improved, or if necessary even replaced. We have only tested our models of junctions and stops on our own test data, and they might need to be adapted to be suited for real data. One important aspect of the models which we have not tested is their robustness against for example the erratic behaviour of real data. Instead of improving current models, we can also look at models of other patterns to include in our method, possibly indicating different types of patterns in the same visualisation.

Alternatively, we can use the proposed models for different types of visualisations or further processing. It would be interesting to automatically extract an underlying road network from bare movement data using the notion of junctions. Such information can be used to supplement missing information on road map databases, or used as input to abstractly visualise the flow – especially on junctions – of trajectories which derive from some unknown underlying road network.

Acknowledgements

I would like to thank my advisor Marc van Kreveld for all his guidance and helping me find my way in academia; and also Maarten and Frank for always being available to answer my questions. Special thanks to Stefan van der Spek for his time to visit Marc and me in Utrecht and discuss parts of this thesis. Lastly, thanks Felix, Gabe, Hidde, and Wout for making BBL 6.82 a pleasant workplace with always someone prepared to hear me out.

Bibliography

[1] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. A model for enriching trajectories with semantic geographical information. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, page 22. ACM, 2007.

[2] Gennady Andrienko, Natalia Andrienko, Urska Demsar, Doris Dransch, Jason Dykes, Sara Irina Fabrikant, Mikael Jern, Menno-Jan Kraak, Heidrun Schumann, and Christian Tominski. Space, time and visual analytics. *International Journal of Geographical Information Science*, 24(10):1577–1600, 2010.

[3] Gennady Andrienko, Natalia Andrienko, Salvatore Rinzivillo, Mirco Nanni, Dino Pedreschi, and Fosca Giannotti. Interactive visual clustering of large collections of trajectories. In *Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on*, pages 3–10. IEEE, 2009.

[4] Natalia Andrienko and Gennady Andrienko. Spatial generalization and aggregation of massive movement data. *Visualization and Computer Graphics, IEEE Transactions on*, 17(2):205–219, 2011.

[5] Natalia Andrienko and Gennady Andrienko. Visual analytics of movement: An overview of methods, tools and procedures. *Information Visualization*, 12(1):3–24, 2013.

[6] Marc Benkert, Bojan Djordjevic, Joachim Gudmundsson, and Thomas Wolle. Finding popular places. *International Journal of Computational Geometry & Applications*, 20(01):19–42, 2010.

[7] Kevin Buchin, Maike Buchin, Marc van Kreveld, Maarten Löffler, Rodrigo I Silveira, Carola Wenk, and Lionov Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

[8] Maike Buchin, Anne Driemel, Marc van Kreveld, and Vera Sacristán. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, (3):33–63, 2014.

[9] Sergio Cabello, Herman Haverkort, Marc van Kreveld, and Bettina Speckmann. Algorithmic aspects of proportional symbol maps. *Algorithmica*, 58(3):543–565, 2010.

[10] Clément Calenge, Stéphane Dray, and Manuela Royer-Carenzi. The concept of animals' trajectories from a data analysis perspective. *Ecological informatics*, 4(1):34–41, 2009.

[11] Bernard Chazelle and Herbert Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM (JACM)*, 39(1):1–54, 1992.

[12] Tarik Crnovrsanin, Chris Muelder, Carlos Correa, and Kwan-Liu Ma. Proximity-based visualization of movement trace data. In *Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on*, pages 11–18. IEEE, 2009.

[13] Urška Demšar and Kirsi Virrantaus. Space–time density of trajectories: exploring spatio-temporal patterns in movement data. *International Journal of Geographical Information Science*, 24(10):1527–1542, 2010.

[14] Holger Dettki, Göran Ericsson, and Lars Edenius. Real-time moose tracking: an internet based mapping application using gps/gsm-collars in sweden. *Alces*, 40, 2004.

[15] Steven van Dijk, Marc van Kreveld, Tycho Strijk, and Alexander Wolff. Towards an evaluation of quality for names placement methods. *International Journal of Geographical Information Science*, 16(7):641–661, 2002.

[16] Somayeh Dodge, Robert Weibel, and Anna-Katharina Lautenschütz. Towards a taxonomy of movement patterns. *Information Visualization*, 7(3-4):240–252, 2008.

[17] Fosca Giannotti, Mirco Nanni, Dino Pedreschi, Fabio Pinelli, Chiara Renso, Salvatore Rinzivillo, and Roberto Trasarti. Unveiling the complexity of human mobility by querying and mining massive trajectory data. *The VLDB Journal—The International Journal on Very Large Data Bases*, 20(5):695–719, 2011.

[18] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–339. ACM, 2007.

[19] Joachim Gudmundsson, Marc van Kreveld, and Bettina Speckmann. Efficient detection of patterns in 2d trajectories of moving points. *Geoinformatica*, 11(2):195–215, 2007.

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

[21] Hanqi Guo, Zuchao Wang, Bowen Yu, Huijing Zhao, and Xiaoru Yuan. Tripvista: Triple perspective visual trajectory analytics and its application on microscopic traffic data at a road intersection. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 163–170. IEEE, 2011.

[22] Torsten Hägerstraand. What about people in regional science? *Papers in regional science*, 24(1):7–24, 1970.

[23] Eduard Imhof. Positioning names on maps. *The American Cartographer*, 2(2):128–144, 1975.

[24] Daniel Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Görg, Jörn Kohlhammer, and Guy Melançon. *Visual analytics: Definition, process, and challenges*. Springer, 2008.

[25] Menno-Jan Kraak. The space-time cube revisited from a geovisualization perspective. In *Proc. 21st International Cartographic Conference*, pages 1988–1996, 2003.

[26] Per Ola Kristensson, Nils Dahlback, Daniel Anundi, Marius Bjornstad, Hanna Gillberg, Jonas Haraldsson, Ingrid Martensson, Mathias Nordvall, and Josefine Stahl. An evaluation of space time cube representation of spatiotemporal patterns. *Visualization and Computer Graphics, IEEE Transactions on*, 15(4):696–702, 2009.

[27] Patrick Laube, Marc van Kreveld, and Stephan Imfeld. Finding remo—detecting relative motion patterns in geospatial lifelines. In *Developments in spatial data handling*, pages 201–215. Springer, 2005.

[28] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604. ACM, 2007.

[29] Jed A Long and Trisalyn A Nelson. A review of quantitative methods for movement data. *International Journal of Geographical Information Science*, 27(2):292–318, 2013.

[30] Gavin McArdle, Urška Demšar, Stefan van der Spek, and Seán McLoone. Interpreting pedestrian behaviour by visualising and clustering movement data. In *Web and Wireless Geographical Information Systems*, pages 64–81. Springer, 2013.

[31] Andrey Tietbohl Palma, Vania Bogorny, Bart Kuijpers, and Luis Otavio Alvares. A clustering-based approach for discovering interesting places in trajectories. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 863–868. ACM, 2008.

[32] Salvatore Rinzivillo, Dino Pedreschi, Mirco Nanni, Fosca Giannotti, Natalia Andrienko, and Gennady Andrienko. Visually driven analysis of movement data by progressive clustering. *Information Visualization*, 7(3-4):225–239, 2008.

[33] Jose Antonio MR Rocha, Gabriel Oliveira, Luis O Alvares, Vania Bogorny, and VC Times. Db-smot: A direction-based spatio-temporal clustering method. In *Intelligent Systems (IS), 2010 5th IEEE International Conference*, pages 114–119. IEEE, 2010.

[34] Roeland Scheepens, Niels Willems, Huub van de Wetering, and Jarke J van Wijk. Interactive visualization of multivariate trajectory data with density maps. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 147–154. IEEE, 2011.

[35] Stefano Spaccapietra, Christine Parent, Maria Luisa Damiani, Jose Antonio de Macedo, Fabio Porto, and Christelle Vangenot. A conceptual view on trajectories. *Data & knowledge engineering*, 65(1):126–146, 2008.

[36] Stefan van der Spek, Jeroen van Schaick, Peter de Bois, and Remco de Haan. Sensing human activity: GPS tracking. *Sensors*, 9(4):3033–3055, 2009.

[37] Marc van Kreveld, Étienne Schramm, and Alexander Wolff. Algorithms for the placement of diagrams on maps. In *Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 222–231. ACM, 2004.

[38] Kevin Verbeek, Kevin Buchin, and Bettina Speckmann. Flow map layout via spiral trees. *IEEE transactions on visualization and computer graphics*, 17(12):2536–2544, 2011.

[39] Florian Verhein and Sanjay Chawla. Mining spatio-temporal patterns in object mobility databases. *Data mining and knowledge discovery*, 16(1):5–38, 2008.

[40] Alexander Wolff, Lars Knipping, Marc van Kreveld, Tycho Strijk, and Pankaj K Agarwal. A simple and efficient algorithm for high-quality line labeling. *Innovations in GIS VII: GeoComputation*, 11:147–159, 1999.

[41] Qing-nian Zhang and Lars Harrie. Real-time map labelling for mobile applications. *Computers, environment and urban systems*, 30(6):773–783, 2006.