Detecting Change in the Shape of Moving Point Sets

van Mulken, Max

Award date:
2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Algorithms, Geometry & Applications

Detecting Change in the Shape of Moving Point Sets

Master's Thesis

Max van Mulken

Supervisors:

Dr. Kevin Verbeek
Prof. Dr. Bettina Speckmann
Prof. Dr. Kevin Buchin (TU Dortmund)

Eindhoven, November 2021

Abstract

Sets of moving entities can form groups that travel closely together for longer periods of time. Analyzing the shape and the collective movement properties of such groups of people, vehicles, or wildlife animals allows us to make inferences about the underlying movement behavior. An example of such a property is the density of the group, which has been shown to be influenced by external stimuli in applications from, for example, wildlife ecology. Motivated by such applications, this thesis investigates how to find and maintain the location of dense and sparse areas of a given group of moving entities. We achieve this by maintaining the maxima and minima of a probability density estimate of a moving input point set P, which represent the dense and sparse areas of the data, respectively. We use the *kinetic data structure* framework to build a data structure that maintains the extrema of the probability density estimate by utilizing the slope of the function. The kinetic data structure is compact, local, responsive and weakly efficient. We also investigate how we can use an ε-approximation of the kernel density estimate to make the kinetic data structure more efficient. This approximation brings the number of processed events down from $O(n^2)$ to $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$ and maintains the most relevant extrema of the density function. Using the approximation does, however, negatively impact the locality and the responsiveness of the data structure. In addition to the work on density, we also demonstrate how to classify movement phases of a group of moving entities. We do so by combining an existing a grouping approach with an existing approach for the classification of movement phases of a single entity. In a proof-of-concept implementation, the resulting method is able to classify movement phases of a group mostly based on the velocity of the entities.

Contents

Chapter 1

Introduction

Over the past decade, technological advances in tracking technology have greatly increased the amount of movement data that can be collected. Tracking such moving entities can be useful in a large range of applications such as wildlife ecology, transportation sciences and sports analysis. This large amount of collected data has also led to a higher demand of methods to analyze the behavioral movement patterns that underlie the movement.

Such movement data sets are often represented by discrete sets of timestamped positions called *trajectories*. Trajectories are can be interpreted as polygonal curves interpolated over these sampled locations (see Figure 1.1). Analyzing the movement behavior behind such trajectories is an important task. There is a large body of work exploring and analyzing trajectories of individual entities which, for example, focuses on segmenting and classifying individual trajectories to distinguish different movement phases [2, 7, 15, 19].

We focus on the collective analysis of sets of related entities. There are many concepts that describe when a set of entities form a cohesive collective, and there exists a large body of work that attempts to identify and maintain them. Some examples of such concepts are moving clusters, herds, flocks, and groups. Many of these concepts differ slightly in how they are defined. The concept of *moving clusters* [23] attempts to, given any standard static-clustering algorithm, capture how these clusters move in a setting with discrete time steps. Two clusters at two subsequent time steps are considered equal if a large enough fraction of their entities overlap. *Herds* [20, 31] further build on this concept by specifying, given a set of moving clusters, how these clusters expand, shrink, merge and split over time. The concept of *flocks* [14, 31] has been proposed to identify sets of entities that travel together for an extended period of time without the need of a static clustering algorithm. The concept of flocks is based on three parameters: size parameter m, time parameter k and distance parameter r. A set of at least m entities is said to form a flock if there are at least k consecutive time steps for which there is a disk of radius r that contains all m entities.

In this thesis, we use the concept of *groups* [6, 21] to define when and where entities travel

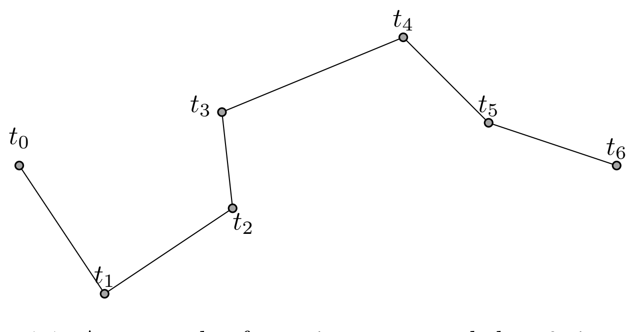


Figure 1.1: An example of a trajectory, sampled at 6 time steps.

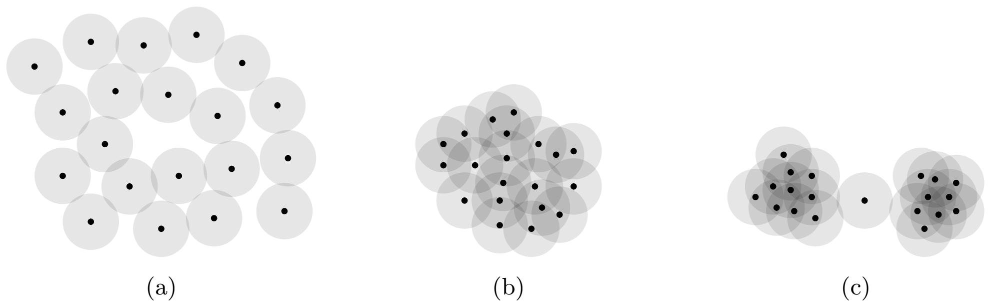


Figure 1.2: Three different groups of 15 points. Let the gray discs denote the ε-discs of the points.

together in a continuous setting. Groups work in a way that is very similar to flocks: we have a size parameter m, a temporal parameter δ, and a spatial parameter ε. A set of entities is considered a group during time interval I if it consists of at least m entities and I is of length at least δ. Unlike flocks, however, groups do not require all entities to be within a single disc. There are two different definitions of groups depending on how the spatial parameter is used. Let the ε-disc of a point p be the disc centered at p with radius ε. For a set of entities to form a group, Hwang et al. [21] require all entities p in the group to lie within the ε-disc of all other entities in the group. Conversely, Buchin et al. [6] require that, for each entity p in the group, the ε-disc of p overlaps the ε-radius disc of *at least one* other point in the group. In this thesis we use the latter definition by Buchin et al., which is discussed in more detail in Chapter 4.

Using this definition, we are able to find groups in a set of moving entities. In some applications it is useful to be able to further distinguish these groups from one another based on their shape. In wildlife ecology, for example, animals often clump together when not under threat and respond to immediate danger by spreading out [13, 26]. This means that being able to discern the shape of such a group allows us to make inferences about external stimuli affecting the movement behavior of the group. Consider the three groups in Figure 1.2. These groups all consist of 15 points, and are all grouped using the same spatial parameter ε, yet they differ in shape. Since the spatial parameter ε is fixed, we are unable to distinguish these groups from one another. We desire a way to differentiate differently shaped groups as they evolve over time. In this thesis, we aim to provide a way to evaluate the shape of a group by identifying its particularly dense and sparse areas. The location of these dense and sparse areas can give us a deeper understanding of the shape of a given group.

In order to identify dense and sparse area in groups of entities, we must first be able to quantify the density. We estimate the density of our point set using a technique from statistics called *kernel density estimation* [28]. Kernel density estimation attempts to estimate the probability density function of a random variable using a sample of points from the random variable. We can use it to get an estimate of a function describing the density of our group of entities (see Figure 1.3). The maxima and the minima of this function indicate areas where the density is estimated to be high or low, respectively. Thus, to identify dense and sparse areas in the input data, we aim to find and maintain the maxima and minima of this function as the underlying points move.

Identifying the extrema of a function suggests the use of *computational topology* [12]. Computational topology describes a set of tools concerned with the shape of data, and has been used before to study groups [6]. In particular, storing the extrema of a function is closely related to storing the *Morse-Smale complex* of the function. A Morse-Smale complex aims to capture the topology of a function by storing the *critical points* of the function, i.e. points at which the derivative of the function is 0. Additionally, it describes the relations between these critical points as integral lines which intuitively represent the “flow” from a maximum to a minimum. See Figure 1.4 for an example of a Morse-Smale complex on a one-dimensional function. There has extensive work on efficiently computing the Morse-Smale complex for static functions [17, 18, 34]. Furthermore, Ophelders et al. [27] describe how to maintain the Morse-Smale complex of a discrete function of which the heights of the vertices are time-varying. In our setting, however, since

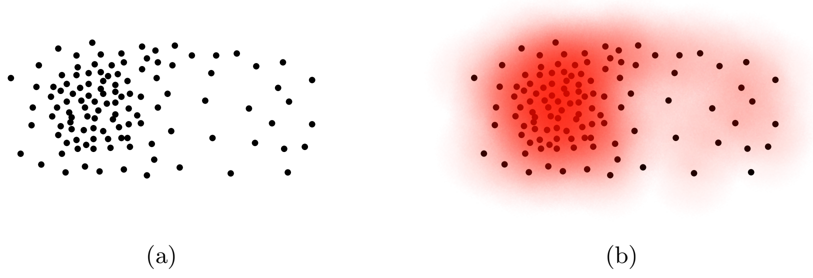


Figure 1.3: An example group of entities, represented by points, as well as a visualization of its kernel density estimate. The red areas indicate high values of the estimated density function.

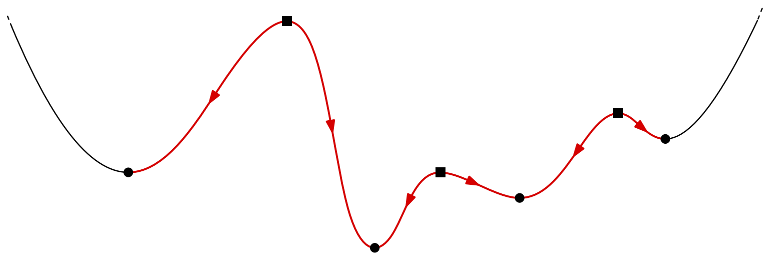


Figure 1.4: An example of a Morse-Smale complex on a function over the one-dimensional domain. Squares indicate maxima, discs indicate minima, and the red integral lines indicate the relation between these extrema.

the entities underlying the function are moving, the vertices are not only time-varying in height but also in position. Since this problem is quite complex, we will for now consider functions over the one-dimensional domain. Note that a one-dimensional Morse-Smale complex can simply be represented by an ordered list of extrema.

To maintain the Morse-Smale complex of a discrete, time-varying function, Ophelders et al. [27] use the *kinetic data structure* (KDS) framework. Generally, kinetic data structures are data structures that aim to efficiently maintain some attribute of a moving geometric system. The KDS framework assumes the movement paths of the entities are known beforehand. This assumption is used to predict the timestamps at which the maintained attribute changes. Using these predicted time stamps we can guarantee that the data structure is updated only when strictly necessary. This ensures that the maintained attribute is always correct, and avoids any superfluous calculations. In this thesis we will also utilize the KDS framework, which is discussed in more detail in Chapter 2.

In addition to the work on group density, we also investigate how to detect when the movement patterns of groups of two-dimensional entities changes. For individual entities, a number of methods that identify movement patterns already exist. The methods we focus on here assume that the movement of an entity in between two observations can be described relatively well by a random walk. As such, these methods aim to fit *parameterized movement models* to the data, which are described by a Gaussian process. These models include a parameter p that characterizes some aspect of the movement data modelled by the process (e.g. velocity or movement direction). Using such models, sub-trajectories can be assigned a value of parameter p, as well as a likelihood score that indicates how likely it is that the movement in the sub-trajectory is well-described by the movement model using parameter p.

An example of how such a movement model can be used in practice is demonstrated well in the method called *behavioral change point analysis* [15] (BCPA). This methods aims to identify structural shifts in the movement of an entity by fitting a movement model to the data that is parameterized by the *auto-correlation* ρ of the velocity and the movement direction. This auto-correlation parameter ρ indicates how likely the velocity and the movement direction is to change

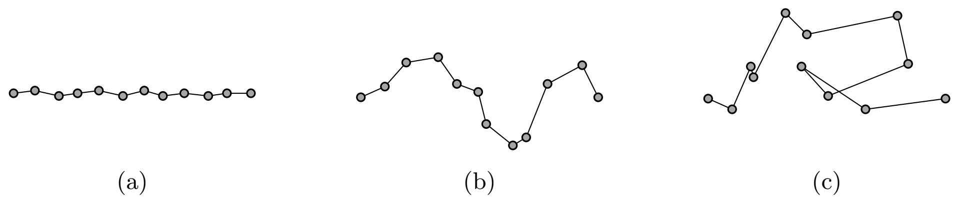


Figure 1.5: Three example trajectories with a (a) high, (b) medium, and (c) low amount of auto-correlation for the velocity and movement direction.

between subsequent observations. Examples of three trajectories with different values of this auto-correlation parameter ρ can be found in Figure 1.5. This movement model used by BCPA will be explained in more detail in Chapter 4.

Another use of such parameterized movement models can be seen in the segmentation and classification work of Alewijnse et al. [2]. In the single-trajectory setting, a *segmentation* breaks a trajectory into disjoint sub-trajectories, called *segments*, in such a way that the properties of the movement are similar within the segment and different between segments. The *classification* of a set of segments assigns segments to classes based on their movment properties. There are many known approaches to segmenting and classifying trajectory movement, generally limited to individual trajectories [5, 22, 24]. We aim to investigate whether the approach by Alewijnse et al. can be extended to groups of trajectories.

To this end, let *group segmentation* be the act of finding similar sub-trajectory clusters in a group of trajectory data. Where single-trajectory segmentation only asks to break up single trajectories into sub-trajectories depending on the movement characteristics, group segmentation aims to further group similar segments between different trajectories. Furthermore, *group classification* describes the act of, given a group segmentation, assigning the group segments to different classes based on their similar movement characteristics. We will use group classification to, given a segmentation of a group of trajectories, identify phases in the movement behavior by classifying these segments.

Similarly to BCPA, the single-trajectory segmentation and classification algorithms described by Alewijnse et al. [2] also fit a parameterized movement model to the data. To obtain a segmentation, a dynamic programming approach is used to find the optimal parameter values for which the movement model is most likely to fit each sub-trajectories. Sub-trajectories with similar parameter values are then merged into segments in order to obtain a segmentation. The classification algorithm by Alewijnse et al. works in a similar manner: dynamic programming is used to find optimal parameter values for each segment, after which similar segments are classified into a single class.

The single-trajectory segmentation approach by Alewijnse et al. [2] was recently extended by Mols [25] to work with groups of trajectories as well. Mols achieves this is three different ways: the clustering method, the incremental method, and the two-step method. The *clustering method* is a non-deterministic approach that aims to find a group segmentation by finding specific sets of sub-trajectory clusters. The method first incrementally constructs a large set of possible sub-trajectory clusters by fitting a movement model to these clusters. Thereafter, a subset of these clusters is selected such that the subset describes a valid group segmentation. The *incremental method* works by using the segmentation algorithm by Alewijnse et al. to find a segmentation for each individual trajectory separately. Then, it aims to merge these single-trajectory segmentations such that the resulting merged segmentation is a valid group segmentation. Lastly, the *two-step method* describes how to obtain a group segmentation in two separate steps. First, it finds groups in the data using the trajectory grouping structure described by Buchin et al. [6]. Then, it uses the segmentation algorithm by Alewijnse et al. to segment these groups separately. Mols concludes that this last method, the two-step method, yields the most interesting and intuitive results. We will use the two-step method to generate a group segmentation. This group segmentation will be used to investigate to what effect the single-trajectory classification algorithm by Alewijnse et al.

can be used to classify phases in a group segmentation.

1.1 Contributions

We present how to identify and maintain dense and sparse areas in a set of moving one-dimensional points by finding the maxima and minima of the kernel density estimation function. We first do so in a static setting, after which we describe how to maintain these maxima and minima through time as the input points start moving. We achieve this by using the *kinetic data structure* framework, which is commonly used to maintain attributes of moving geometric systems. A more detailed description of these techniques is provided in Chapter 2. The kinetic data structure we produce is compact, local, responsive, and weakly efficient. Furthermore, we consider possibilities to improve the efficiency of our kinetic data structure in settings where the input data set is large. We do this by finding and maintaining a *coreset* of the input point set, which is a point set of significantly smaller size than the original input point set of which the probability density function is “similar”. As we will see, the use of the coreset presents a trade-off between reducing the number of required operations and increasing the amount of time it takes to process an operation. The kinetic data structure is described in detail in Chapter 3.

In Chapter 4 we explore how we can apply the classification algorithm by Alewijnse et al. [2] to a group segmentation produced by the two-step approach described by Mols [25]. Since both of these approaches require a parameterized movement model to work, we use the model described by Gurarie et al. [15] in their description of behavioral change point analysis. We apply the classification algorithm on a real data set describing a troop of baboons moving through a wildlife reserve. These results show that the classification algorithm is able to classify movement phases relatively well in a group segmentation. The classification, however, seems based mainly on the velocity of the entities rather than the movement direction. Lastly, Chapter 5 discusses the work in this thesis and describes possible directions for future work.

Chapter 2

Preliminaries

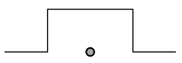
This chapter provides the background and terminology used throughout the next chapter. In Section 2.1 we describe a reliable and well-known approach of estimating the density function of a static population based on a finite sample. Then, in Section 2.2 we explain how we can efficiently extract the relevant information from the estimated density function without storing superfluous elements of the function. In Section 2.3 we describe an approach to extend these static approaches to work for moving points as well.

2.1 Kernel Density Estimation

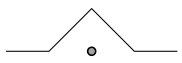
Kernel density estimation (KDE) is a robust method of estimating the continuous probability density distribution from a given finite point set. Given a static set of observations P from a domain $\mathbb{M}$, the aim is to construct a function KDE_P such that, for any input value from the domain $\mathbb{M}$, KDE_P returns an estimate of the *density* at that input value. Consider P to be an input point set of size n, where $P \subset \mathbb{R}^1$.

Kernel density estimation uses a notion of *kernels*, which (in our one dimensional case) are functions $K : \mathbb{R}^+ \to \mathbb{R}^+$. Each point in the input point set increases the value of the estimated density function according to its kernel. Given input parameter $y \in \mathbb{R}^+$, some examples of kernels are:

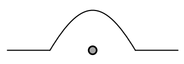
- Uniform: $K(y) = \begin{cases} 1 & \text{if } y < \sigma \\ 0, & \text{otherwise.} \end{cases}$



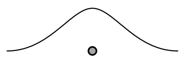
- Linear: $K(y) = \begin{cases} 1 - \frac{y}{\sigma} & \text{if } y < \sigma \\ 0, & \text{otherwise.} \end{cases}$



- Epanechnikov: $K(y) = \begin{cases} 1 - \frac{y^2}{\sigma^2} & \text{if } y < \sigma \\ 0, & \text{otherwise.} \end{cases}$



- Gaussian: $K(y) = \exp\left(-\frac{y^2}{2\sigma^2}\right)$



All of these kernels contain a parameter σ that controls the amount of the data smoothing, which we assume to be fixed. In this thesis, we will use the linear kernel. Although this kernel is less widely used than, for example, the Epanechnikov or Gaussian kernel, it produces a piece-wise linear terrain which is easy to work with and yields a sufficient approximation of the density of the input point set. An example of an estimated density function of a single point using the linear kernel can be seen in Figure 2.1. Intuitively, for the linear kernel, the smoothing parameter σ determines the width of the kernel of each point.

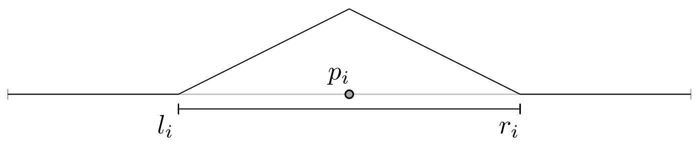


Figure 2.1: An example of a single linear kernel.

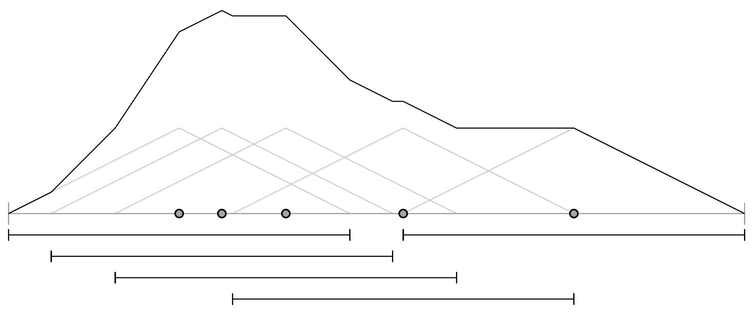


Figure 2.2: An example of the kernel density estimation function.

Given the static input point set P of size n and the kernel function K, we can define the kernel density estimate as a function

$$\mathrm{KDE}_P(x) = \frac{1}{n} \sum_{p \in P} K(|p - x|)$$

where $x \in \mathbb{R}^1$ is any value in the domain. An example of such a kernel density estimation on a larger point set can be seen in Figure 2.2.

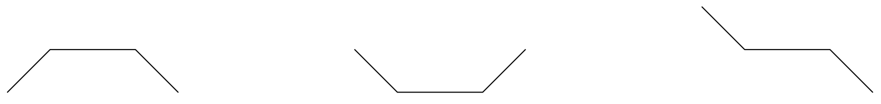
2.2 Terrains

Let $P \subset \mathbb{R}^1$ be an input point set of n points and let KDE_P denote the KDE function of P using the linear kernel, such that $\mathrm{KDE}_P(x)$ for $x \in \mathbb{R}$ describes the value of the KDE function at coordinate x. Additionally, for each $p_i \in P$, let $l_i = p_i - \sigma$ and $r_i = p_i + \sigma$ for $1 \leq i \leq n$ denote the left- and right *boundary* of the kernel of p_i (see Figure 2.1). Observe that function KDE_P is piece-wise linear and consists of line segments between *bends* at the x coordinates of l_i, p_i and r_i for $1 \leq i \leq n$. Let a bend corresponding to point p_i or boundary l_i or r_i be denoted bp_i, bl_i or br_i, respectively. At each of these bends, the slope of the function changes: at bends bp_i the slope decreases, and at bends bl_i and br_i the slope increases. Thus, we can consider the KDE_P an x-monotone, polygonal chain in $\mathbb{R}^2$ between bends bp_i, bl_i and br_i for $1 \leq i \leq n$. This x-monotone, polygonal chain constructed using P is henceforth referred to as a *terrain* T_P.

2.2.1 Extreme values

In order to track the most/least densely populated areas in the domain, we aim to find and track the *extrema* of the terrain. Generally, extrema are defined as either *maxima* or *minima*, which can be either global or local. Since we are interested in finding all dense and sparse areas in the input, we aim to identify all local extrema. Therefore, in the remainder of this thesis, whenever we mention extrema we are referring to *local* extrema. We consider a bend b in the terrain T_P a maximum if the slope of the terrain is increasing before b, and decreasing after b. Conversely, we consider bend b a minimum if the slope of the terrain is decreasing before, and increasing after b.

An important observation is that extrema in our terrain T_P may not always consist of a single bend. Since we are using the linear kernel, there are many situations in which there are line segments in the terrain that have a slope of 0, resulting in a completely horizontal piece of terrain.



(a) Plateau (b) Valley (c) Flat

Figure 2.3: Three different types of horizontal segments that can appear in the terrain.

These horizontal segments of the terrain come in three different types that depend on the slope before/after these segments. If the slope of the terrain is increasing before and decreasing after the horizontal segment, we call the segment a *plateau* (Figure 2.3a). A plateau can be described as a type of maximum that consists of two bends. Similarly, if the slope is decreasing before and increasing after the horizontal part, it is considered a type of minimum called a *valley* (Figure 2.3b). Any other horizontal line segment (with either increasing slope both before and after, or decreasing slope before and after) is called a *flat* (Figure 2.3c).

These three types also consist of different types of bends. Recall that *bp* describes a bend corresponding to an input point, and that *bl* and *br* bends describe bends corresponding to left- and right kernel boundaries, respectively. Valleys require two subsequent *bl* or *br* bends, since a valley can only exist if the two bends neighboring the valley increase the slope. Similarly, plateaus require two subsequent *bp* bends, and flats require a *bp* and either a *bl* or a *br* bend. Later, it will be useful to be able to refer to the bends that are incident on horizontal segments of the terrain. To this end, we call a bend that neighbors a horizontal segment a horizontal bend, or *h-bend* for short. Depending on whether an *h*-bend is left- or right incident on a horizontal segment, it can be either the starting- or the ending-bend of a plateau, flat or valley. We call an *h*-bend *starting* if it is the left *h*-bend of a horizontal segment, and *ending* if it is the right *h*-bend of such a segment. We call any bend that is not a maximum nor an *h*-bend a *regular* bend.

2.3 Kinetic Data Structures

The approach described above is, in principle, well-suited for static point sets. Our goal, however, is to maintain these dense and sparse areas as the point set P moves. A naive approach would be to divide the continuous temporal space in which the points are moving into a sequence of discrete observations at a fixed time step Δt. This way, one can maintain the extrema of T_P simply by recomputing them at each step. There are issues with this approach, however: if the time step is not carefully picked, we may either miss important changes in the system or perform many unnecessary computations.

To resolve these issues it would be helpful to not be restricted to a fixed time step Δt, but rather to perform updates to the system only when updates are strictly necessary. A *kinetic data structure* (KDS) is capable of maintaining an attribute of a system by anticipating when the moving components of the system will cause a change to the attribute. If we want to track the extrema of T_P, the KDS would be able to anticipate the times at which the extrema of T_P change and update the list of extrema at exactly these times.

A KDS achieves this as follows: it maintains a list of *certificates* on the system which describe conditions under which the attribute remains correct in its current state. A certificate that describes conditions on a maximum bp_{max}, for example, could take advantage of the fact that we know that any maximum bp_{max} should be higher than both of its neighboring bends. Let the *left-* and *right neighbor* of a bend b simply be the bends directly to the left- and right of b, respectively. Then, certificates that verify this maximum could be:

1. bp_{max} is higher than its left neighbor
2. bp_{max} is higher than its right neighbor

We know that bp_{max} will remain a maximum as long as these certificates are true. When one of these certificates is violated, we know that bp_{max} is no longer a maximum. If the KDS has access to the movement trajectories of P, it can predict at which time each certificate in the list is violated and maintain them in a priority queue. This priority queue is called the *event queue*, and it stores an *event* for each certificate that will be violated using its predicted timestamp as priority. If we store similar certificates for every bend in the terrain, we know that the extrema of the terrain can only change at the time stamps of events in the event queue. Then, to maintain the extrema of T_P, we wait until the first event in the event queue is reached, perform updates to the data structure at that time, and continue down the event queue.

To maintain correctness, there are a number of things that need to happen whenever a certificate violation triggers an event. The certificate violation directly implies that the list of extrema must be updated to reflect the changes, but we are also required to add or delete certificates to/from our list of certificates. This, in turn, requires us to update the event queue to reflect these certificate changes. For example, if the certificate *"bp_{max} is higher than its left neighbor"* fails, we need to:

1. Update the list of extrema by removing bp_{max} and, if necessary, the left-neighbor of bp_{max}, possibly replacing them with new bends;
2. Update the certificates of bp_{max} (and its left neighbor) to reflect the fact that bp_{max} is now *lower* than its left neighbor;
3. Update the event queue to reflect the changes to the certificates.

It is useful to note that, while not necessary in our simple example, a more complicated setting may require us to maintain a more complex internal structure. In such settings some certificate failures may not necessarily require updates to the maintained attribute, but rather trigger events that merely update the internal data structure. A distinction is made between these two types of events: *external events* are events that affect the maintained attribute directly (e.g. a change to the extrema of T_P), whereas *internal events* are events that are processed because of internal needs of the data structure without changing the maintained attribute itself directly.

2.3.1 Evaluating a KDS

After designing a KDS, we want to be able to evaluate it. To this end, there are four common quality measures with which the quality of a KDS is often quantified.

Firstly, the *compactness* of a KDS describes the worst-case size of the list of certificates. We call a KDS compact if the number of certificates it maintains in the worst-case is at most $O(n \text{ polylog } n)$, where n is the size of the input. In our example above we maintain $O(n)$ certificates (one for each pair of neighboring bends), meaning a KDS using only these certificates would be compact.

The second performance measure is the *locality* of the KDS. The locality of a KDS is determined by the maximum number of events in the event queue that depend on a single object. In our example, any event in the event queue depends on two neighboring bends. Trivially, this means that any bend always appears in at most two events at a time. We call a KDS local if the worst-case number of events in the event queue that depend on a single object is at most polylogarithmic in the size of the input.

Another key quality measure is the *efficiency* of the KDS, which is determined by the worst-case number of processed events. This is where the distinction between internal and external events becomes important. Simply measuring the absolute number of events would not make much sense: the number of external events is a lower bound on the number of events required by any structure that maintains the desired attribute. The goal is to construct a KDS where the worst-case total number of processed events is of the same order as the number of external events that must be processed in the worst-case. To compute this worst-case, we must restrict ourselves to specific classes of motion. Such motion classes can be, for example, linear or polynomial motion, in which the movement pattern is described by a linear function or a polynomial function, respectively.

Note that in the definition above we are comparing the number of events over two classes of movement patterns: one that maximizes the total number of events and one that maximizes the number of external events. However, there may still be situations in which there are many more events processed than the number of external events that take place. Therefore, we further split the efficiency of a KDS into two categories: *weak efficiency* and *strong efficiency*. A weakly efficient KDS only guarantees that the total number of events that are processed is never more than the worst-case number of external events. A strongly efficient KDS, however, guarantees that the worst-case *ratio* of total events to external events, taken over all possible allowed movements, is constant. This means we no longer necessarily look at movement classes in which the number of external events is large, but we look at movements in which the ratio of total number of events to external events is large instead.

Lastly, we define the *responsiveness* of a KDS to be the time required to repair the structure after a certificate failure. This includes repairing the certificate set, updating the maintained attribute and adding/removing events to/from the event queue. We call a KDS responsive if the worst-case cost of processing a certificate failure is at most polylogarithmic.

Chapter 3

Maintaining extrema of a kinetic density estimate

Using the tools described in the previous chapter, this chapter describes how we build a kinetic data structure that aims to track the dense and sparse areas of a moving point set. To this end, we use the same terminology put forward in the previous chapter. Given a one-dimensional point set P of size n, we construct terrain T_P by computing the kernel density estimation function KDE_P using the linear kernel.

The discussion of the data structure in this chapter is split into four parts. In Section 3.1 we discuss how we are able to efficiently find the extrema in T_P when point set P is static. We do so by describing how to compute the extrema of T_P from point set P without having to actually construct the terrain T_P itself. Section 3.2 describes what the observations made for the static setting entail for the kinetic setting, and how the extrema found in the static setting actually change when the points in P start moving. Section 3.3 details how to construct and maintain a KDS that utilizes the findings of the previous two chapters, and analyses the KDS using the quality measures described in Section 2.3.1. Lastly, in Section 3.4 we discuss how the efficiency of the KDS can potentially be improved by using an approximation of P.

3.1 Locating and storing extrema

Since our goal is merely to compute and maintain the extrema of terrain T_P through time, computing and maintaining the entire terrain T_P would be inefficient; it would not only require us to store a large amount of redundant information, but also require a large amount of superfluous computations. Therefore, it would be more efficient to be able to describe the extrema using input point set P only, and then maintain them from there. Since extrema can be determined using the slope of the terrain, we will describe how the points from P directly determine the slope of the terrain T_P.

Observe that the slope of a single linear kernel is equal to $\frac{1}{\sigma n}$. Additionally, note that since T_P is constructed directly from the kernel density estimation function using linear kernels, the slope of the terrain at any location must be an integer multiple of the slope of a single linear kernel. Let the *range* $R(x)$ of any location $x \in \mathbb{R}$ be the set of points p_i from P that are within σ distance of x, described by the interval $(x - \sigma, x + \sigma)$. The range of x can be decomposed into the *left-range* $R_\ell(x)$ and the *right-range* $R_r(x)$ of x, which describe the points in interval $(x-\sigma, x)$ and $(x, x+\sigma)$, respectively (see Figure 3.1). Note that these ranges are described by *open* intervals, and thus do not include their end points. The slope of T_P at location $x \in \mathbb{R}$ can then be described as

$$\frac{|R_r(x)| - |R_\ell(x)|}{\sigma n} \tag{3.1}$$

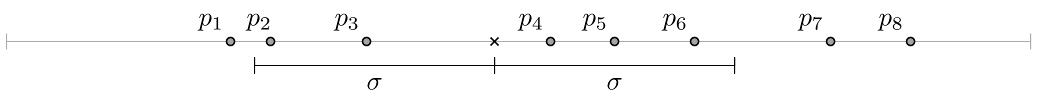


Figure 3.1: An example of the left- and right ranges of a location x indicated by $\times$. We have $R_\ell(x) = \{p_2, p_3\}$ and $R_r(x) = \{p_4, p_5, p_6\}$.

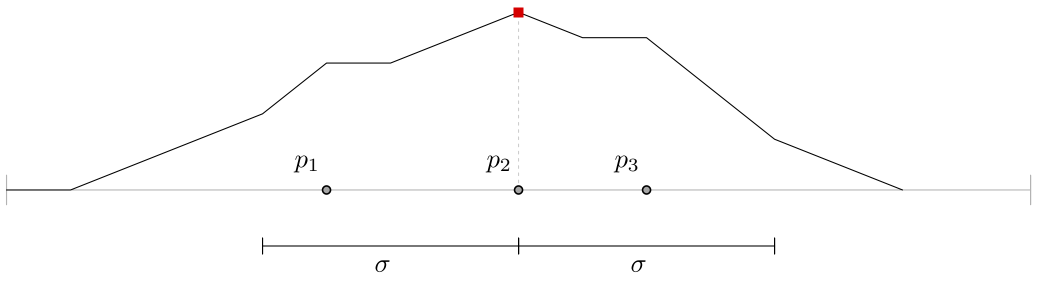


Figure 3.2: An example of a single-bend maximum at bp_2, highlighted in red. Note that $|R_\ell(p_2)| = |R_r(p_2)| = 1$.

since each point in the right-range of x increases the slope of T_P at x by $\frac{1}{\sigma n}$, and each point in the left-range of x decreases the slope by $\frac{1}{\sigma n}$. Intuitively, the points in P and their boundaries influence the slope of terrain T_P as follows: when doing a walk over terrain T_P from left to right, each bl bend we encounter increases the size of our right-range by one, and each br bend we encounter decreases the size of our left-range by one. This means that both bl and br bends increase our slope by $\frac{1}{\sigma n}$. Each bp bend we encounter decreases the size of our right-range and increases the size of our left-range by one each which decreases the slope of the terrain by $\frac{2}{\sigma n}$ in total.

When considering the locations of the extrema, it will be useful to be able to classify the slope of T_P before and after a point. To this end, let x^- and x^+ for some $x \in \mathbb{R}^1$ denote $x - \varepsilon$ and $x + \varepsilon$, respectively, for some arbitrarily small $\varepsilon > 0$. This way, x^- describes a location slightly to the left of x and x^+ describes a location slightly to the right of x. This gives us the following equations for points $p_i \in P$:

$$R_\ell(p_i^-) = R_\ell(p_i) \tag{3.2}$$

$$R_r(p_i^-) = R_r(p_i) \cup \{p_i\} \tag{3.3}$$

$$R_\ell(p_i^+) = R_\ell(p_i) \cup \{p_i\} \tag{3.4}$$

$$R_r(p_i^+) = R_r(p_i) \tag{3.5}$$

Using these equations, it is relatively straightforward to find the maxima in our terrain. We provide the arguments for situations where p_i is a single-bend maximum and, since we must find plateaus as well, where p_i is an h-bend. First, we evaluate when $p_i \in P$ is a single-bend maximum:

Lemma 1. *Bend bp_i with $p_i \in P$ is a maximum in terrain T_P if and only if $|R_\ell(p_i)| = |R_r(p_i)|$.*

Proof. Assume bp_i is a maximum. By definition, this means that the slope of terrain T_P is increasing at p_i^- and decreasing at p_i^+ . By Equation 3.1 this means that $|R_r(p_i^-)| > |R_\ell(p_i^-)|$ and $|R_\ell(p_i^+)| > |R_r(p_i^+)|$. Using Equations 3.3 and 3.4, this implies that $|R_r(p_i)| + 1 > |R_\ell(p_i^-)|$ and $|R_\ell(p_i)| + 1 > |R_r(p_i^+)|$. Combining this with Equations 3.2 and 3.5, we get $|R_r(p_i)| + 1 > |R_\ell(p_i)|$ and $|R_\ell(p_i)| + 1 > |R_r(p_i)|$. This directly implies that $|R_\ell(p_i)| = |R_r(p_i)|$.

Now assume $|R_\ell(p_i)| = |R_r(p_i)|$. Then, by Equations 3.2-3.5 we must have $|R_\ell(p_i^-)| + 1 = |R_r(p_i^-)|$ and $|R_\ell(p_i^+)| = |R_r(p_i^+)| + 1$. Therefore, by Equation 3.1, the slope of terrain T_P must be increasing directly before and decreasing directly after bp_i. Thus, bp_i must be a maximum. □

For an example of a single-bend maximum as described in Lemma 1, see Figure 3.2. Next, we investigate when point $p_i \in P$ is an h-bend:

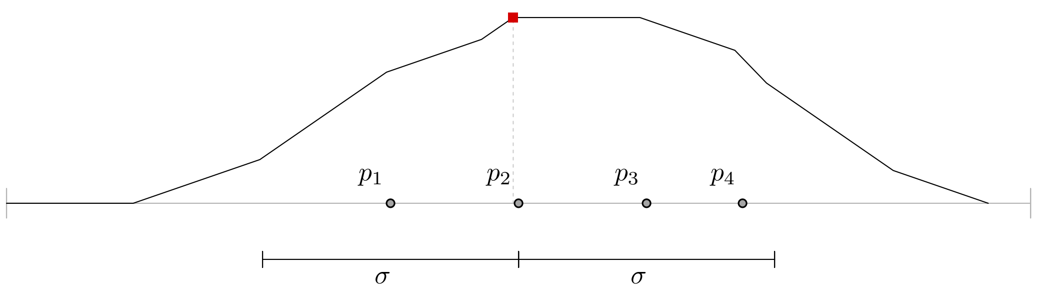


Figure 3.3: An example of a starting h-bend bp_2, highlighted in red. Note that $|R_r(p_2)| = 2 = |R_\ell(p_2)| + 1$.

Lemma 2. *Given some point $p_i \in P$:*

- *bp_i is a starting h-bend if and only if $|R_r(p_i)| = |R_\ell(p_i)| + 1$;*
- *bp_i is an ending h-bend if and only if $|R_\ell(p_i)| = |R_r(p_i)| + 1$.*

Proof. The argument below is for the statement that bp_i is a starting h-bend if and only if $|R_r(p_i)| = |R_\ell(p_i)| + 1$. The proof for the case where bp_i is an ending h-bend is analogous.

Assume bp_i is a starting h-bend. Then we know that $|R_\ell(p_i^+)| = |R_r(p_i^+)|$, since the slope of T_P is 0 at p_i^+. Using Equations 3.4 and 3.5, this yields $|R_\ell(p_i)| + 1 = |R_r(p_i)|$.

Now assume $|R_r(p_i)| = |R_\ell(p_i)| + 1$. We can use Equations 3.4 and 3.5 to show that $|R_\ell(p_i^+)| = |R_r(p_i^+)|$, meaning the slope at p_i^+ is 0. Also, using Equations 3.2 and 3.3, we get $|R_\ell(p_i^-)| + 2 = |R_r(p_i^-)|$, meaning the slope of the terrain is increasing at p_i^-. Therefore, since the slope of T_P is increasing before bp_i and horizontal after bp_i, bp_i is a starting h-bend. □

For an example of a starting h-bend as described in Lemma 2, see Figure 3.3. Before we discuss minima, it is important to observe that single-bend minima in our terrain T_P are degenerate. Because the terrain T_P is constructed using a linear kernel, the slope of T_P is always an integer multiple of $\frac{1}{\sigma n}$. For a single bend b to be a minimum, we require the slope of T_P to be decreasing before and increasing after b. This means that the slope should increase at b, which only happens if b is associated with a boundary. If this is the case, however, b only increases the slope by $\frac{1}{\sigma n}$. Since the slope is always an integer multiple of $\frac{1}{\sigma n}$, a single bl or br bend is never enough to turn a decreasing slope into an increasing slope. Therefore, minima can only exist in a single location if two boundaries coincide, which is simply a valley of length 0. Thus, we are exclusively interested in valleys, which consist of a starting- and an ending h-bend.

It is also important here to note that the left- and right-range of locations are defined as open intervals. This means they only include points from P that are *strictly* within σ distance. Therefore, the left- and right-range of r_i and l_i do not contain the corresponding point p_i. Conversely, it is easy to see that $R_r(l_i^+)$ and $R_\ell(r_i^-)$ do contain p_i, which implies the following equations:

$$R_\ell(r_i^-) = R_\ell(r_i) \cup \{p_i\} \tag{3.6}$$

$$R_r(l_i^+) = R_r(l_i) \cup \{p_i\} \tag{3.7}$$

Other than that, the left- and right-ranges of l_i^-, l_i^+, r_i^- and r_i^+ are equal to those of l_i or r_i:

$$R_\ell(r_i^+) = R_\ell(r_i) \tag{3.8}$$

$$R_r(r_i^-) = R_r(r_i^+) = R_r(r_i) \tag{3.9}$$

$$R_r(l_i^-) = R_r(l_i) \tag{3.10}$$

$$R_\ell(l_i^-) = R_\ell(l_i^+) = R_\ell(l_i) \tag{3.11}$$

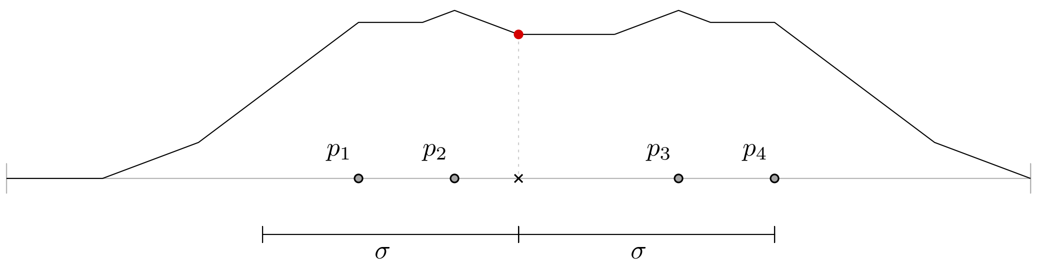


Figure 3.4: An example of a starting h-bend bl_4, highlighted in red. Here, l_4 is indicated by $\times$. Note that $|R_\ell(l_4)| = 2 = |R_r(l_4)| + 1$.

Using these observations, it is again straightforward to find valleys in our terrain by locating the corresponding h-bends:

Lemma 3. *Given some point $p_i \in P$:*

- *bl_i is a starting h-bend if and only if $|R_\ell(l_i)| = |R_r(l_i)| + 1$;*
- *bl_i is an ending h-bend if and only if $|R_\ell(l_i)| = |R_r(l_i)|$;*
- *br_i is a starting h-bend if and only if $|R_\ell(r_i)| = |R_r(r_i)|$;*
- *br_i is an ending h-bend if and only if $|R_\ell(r_i)| + 1 = |R_r(r_i)|$.*

Proof. The argument below is for the first statement that bl_i is a starting h-bend if and only if $|R_\ell(l_i)| = |R_r(l_i)| + 1$. The proofs for the other three statements are analogous.

Assume bl_i is a starting h-bend. Then, since the slope is 0 to the right of bl_i, we have $|R_\ell(l_i^+)| = |R_r(l_i^+)|$. By Equations 3.7 and 3.8 this directly implies that $|R_\ell(l_i)| = |R_r(l_i)| + 1$.

Now assume that $|R_\ell(l_i)| = |R_r(l_i)| + 1$. Again, by Equations 3.7 and 3.8, we can conclude that $R_\ell(l_i^+) = R_r(l_i^+)$. This means the slope of T_P to the right of bl_i is 0. Additionally, by Equations 3.10 and 3.11, we have $|R_\ell(l_i^-)| = |R_r(l_i)^-| + 1$, meaning the slope before bl_i is decreasing. Thus, by definition, bl_i must be a starting h-bend. $\square$

For an example of a starting h-bend as described in Lemma 3, see Figure 3.4. With these observations, it is possible compute the minima and maxima in T_P directly from a static input point set P without constructing the entire terrain T_P.

3.2 Kinetic setting

In this section we characterize how the extrema of T_P described in the previous section change when the points in P start moving through time. As such, consider a set P of n time-varying functions $p_1(t), \dots, p_n(t)$ that describe the positions of a set of n points at time t. Additionally, similarly to before, we use functions $l_i(t) = p_i(t) - \sigma$ and $r_i(t) = p_i(t) + \sigma$ to describe the left- and right boundary of the kernel of point p_i for $1 \leq i \leq n$ at time t.

We assume the movement trajectories of points from P to adhere to two properties: (i) at any given time t there is at most one pair $p_i, p_j \in P$ for which $p_i(t) = p_j(t)$, $p_i(t) = l_j(t)$, or $l_i(t) = r_j(\text{t})$; (ii) if two points/boundaries share a location at time t, the difference between their locations changes sign at time t. For the prior assumption, note that $p_i(t) = p_j(t)$ also implies $l_i(t) = l_j(t)$ and $r_i(t) = r_j(t)$. Similarly, $p_i(t) = l_j(t)$ implies $r_i(t) = p_j(t)$. This prior assumption ensures that we never handle two events at the same time. If the assumption does not hold, we can simply handle simultaneous events as if they happened sequentially.

3.2.1 Events

We will now investigate when the movement of the points in P induce a topological change to T_P. We distinguish three different types of events: *birth* events (Figure 3.11), in which one or more new extrema are created, *death* events (Figure 3.12), in which one or more existing extrema cease to exist, and *shift* events (Figure 3.8), in which the topology of T_P does not necessarily change but in which an extremum shifts from one or more bends to one or more neighboring bends. All shift events require updates to one or more labels of tracked extrema only, and do not correspond to extrema being added or removed to/from T_P.

It is easy to see from Lemmas 1-3 that changes to the extrema of T_P only happen at the timestamps where two points/boundaries share a location: these are the only times at which the ranges of a point/boundary changes. Henceforth, let a *collision* of two points/boundaries at time t denote the fact that these points/boundaries share a location at time t. This means that birth/death/shift events can only take place whenever two points/boundaries collide; in between these times extrema may still move, but the list of extrema of T_P will remain the same. Therefore, to discuss when these events happen, it is sufficient to restrict ourselves to times t at which points/boundaries collide. If a collision occurs at time t, then we will refer to time $t-\varepsilon$ and $t+\varepsilon$ as t^- and t^+ respectively, where $\varepsilon > 0$ is some arbitrarily small value.

We pick $1 \leq i < j \leq n$ such that at time t_0^- we have that $p_i < p_j$, and at t_0 at least one of p_i, l_i, r_i collides with at least one of p_j, l_j, r_j. This allows us to make an exhaustive list of possible collisions at t_0, for each of which we will discuss what events they may cause. We discuss three types of collisions: *Point-point collisions* (PP collisions), where a point p_i collides with another point p_j; *Boundary-boundary collisions* (BB collisions), where a right boundary r_i collides with a left boundary l_j; *Boundary-point collisions* (BP collisions), where boundaries r_i and l_j collide with points p_j and p_i, respectively. As we will see, PP and BB collisions can only cause shift events, whereas BP collisions can also cause birth and death events.

Type 1: Point-point collision

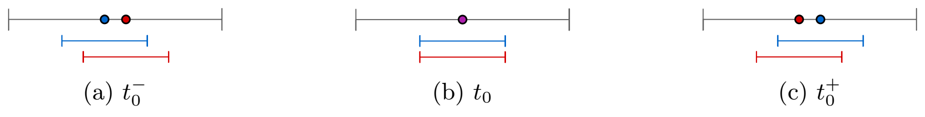

(a) t_0^- (b) t_0 (c) t_0^+

Figure 3.5: Points p_i and p_j for which $l_i(t_0) = l_j(t_0)$, $p_i(t_0) = p_j(t_0)$ and $r_i(t_0) = r_j(t_0)$.

A *PP collision* is a collision where two points collide with each other. Since σ is fixed, this also means that their left- and right boundaries collide with each other, respectively. Thus, if a PP collision takes place at t_0, we have $p_i(t_0) = p_j(t_0)$, $l_i(t_0) = l_j(t_0)$ and $r_i(t_0) = r_j(t_0)$. An example of a PP collision is illustrated in Figure 3.5.

It is easy to see that, when a PP collision happens, only shift events can take place. Note that, at t_0, l_i, p_i, and r_i switch positions with l_j, p_j, and r_j, respectively. Therefore, the ranges of $l_i/p_i/r_i$ at t_0^- will be equal to the ranges of $l_j/p_j/r_j$ at t_0^+, and vice versa. This directly implies that the only event that can happen is a shift event: any point/boundary involved in this collision that is either a maximum or an h-bend incident on a plateau/valley will cause a shift event where the label of the corresponding extremum will shift to the other point/boundary involved in the collision. No death or birth events can ever take place, because any bend involved in this collision is simply directly replaced. Examples of shift events caused by PP collisions can be found in Figure 3.8 (middle and bottom).

Type 2: Boundary-boundary collision

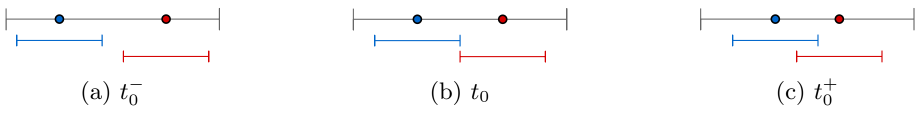


Figure 3.6: Points p_i (blue) and p_j (red) for which $r_i(t_0) = l_j(t_0)$ and $r_i(t_0^-) < l_j(t_0^-)$.

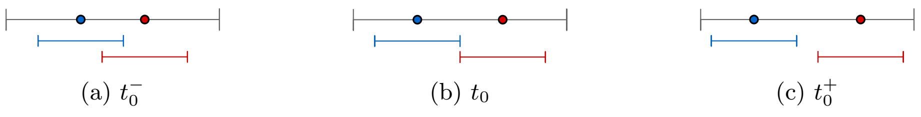


Figure 3.7: Points p_i (blue) and p_j (red) for which $r_i(t_0) = l_j(t_0)$ and $r_i(t_0^-) > l_j(t_0^-)$.

We call a collision a *BB collision* if the only colliding entities are both boundaries. This happens only when a left boundary collides with a right boundary. W.l.o.g., assume that $r_i(t_0) = l_j(t_0)$. We must have either that $r_i(t_0^-) < l_j(t_0^-)$ (Figure 3.6) or $r_i(t_0^-) > l_j(t_0^-)$ (Figure 3.7). We will look at the case where $r_i(t_0^-) < l_j(t_0^-)$; the case when $r_i(t_0^-) > l_j(t_0^-)$ is very similar. Note that, during a PP collision, exclusively the ranges of r_i and l_j change. Thus, we will investigate how the changes to bends br_i and bl_j can cause events to take place during a BB collision.

To see what events can take place due to a BB collision, we make two important observations. Since $r_i(t_0) = l_j(t_0)$, and t_0^- is arbitrarily close to t_0, we have:

$$|R_\ell(r_i(t_0^-))| = |R_\ell(l_j(t_0^-))| \tag{3.12}$$
$$|R_r(r_i(t_0^-))| = |R_r(l_j(t_0^-))|. \tag{3.13}$$

Then, at t_0, p_j and p_i are added to $R_r(r_i)$ and $R_\ell(l_j)$, respectively. Therefore, we get:

$$|R_\ell(r_i(t_0^+))| = |R_\ell(r_i(t_0^-))| \tag{3.14}$$
$$|R_r(r_i(t_0^+))| = |R_r(r_i(t_0^-))| + 1 \tag{3.15}$$
$$|R_r(l_j(t_0^+))| = |R_r(l_j(t_0^-))| \tag{3.16}$$
$$|R_\ell(l_j(t_0^+))| = |R_\ell(l_j(t_0^-))| + 1. \tag{3.17}$$

We can use these observations to show that the type of br_i at t_0^+ is equal to the type of bl_j at t_0^-, and vice versa. This can be argued separately for each possible type of bl/br bend (starting h-bend, ending h-bend, or regular bend). These arguments are all very similar; one such argument is given here. For completeness, the rest of these arguments can be found in Appendix A.

If bl_j is a starting h-bend at t_0^-, then by Lemma 3 we have

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| &= |R_r(l_j(t_0^-))| + 1 \\ \Rightarrow \quad & \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(r_i(t_0^-))| &= |R_r(r_i(t_0^-))| + 1 \\ \Rightarrow \quad & \text{(By Eq. 3.14 \& 3.15)} & |R_\ell(r_i(t_0^+))| &= |R_r(r_i(t_0^+))| \end{aligned}$$

By Lemma 3, this means that br_i is a starting h-bend at t_0^+. Using the analogous arguments in Appendix A, we can also show that:

- If bl_j is an ending h-bend at t_0^- then br_i is an ending h-bend at t_0^+;
- If br_i is a starting or an ending h-bend at t_0^-, then bl_j is a starting or an ending h-bend at t_0^+, respectively;

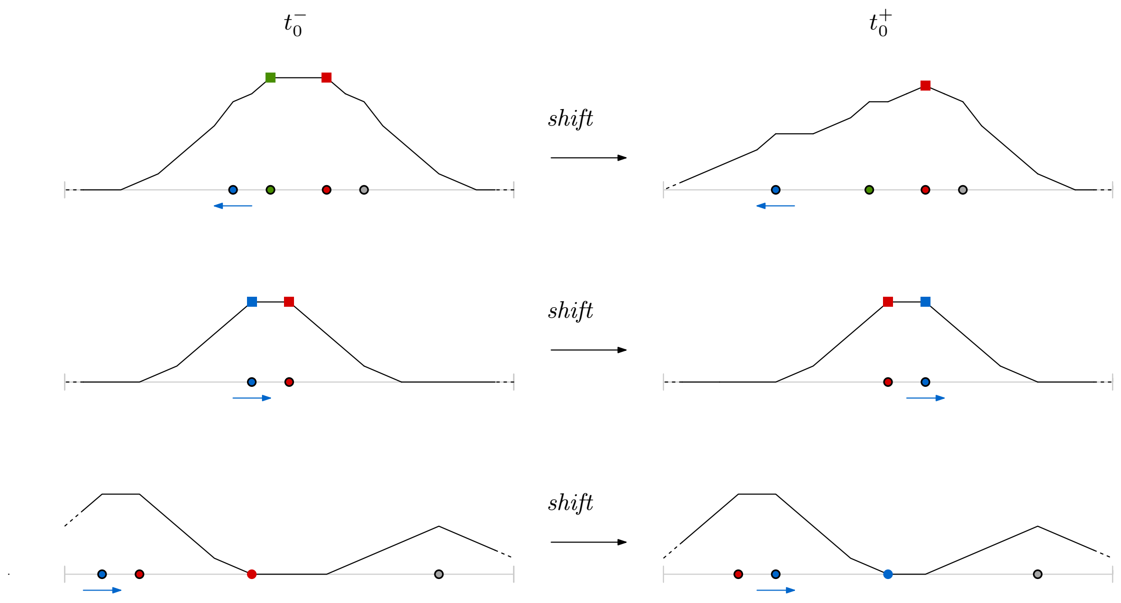


Figure 3.8: Three examples of shift events taking place at t_0 due to a BP-collision (top) or a PP-collision (middle and bottom) between the blue and the red point.

- If br_i or bl_j is a regular bend at t_0^-, then bl_j or br_i will respectively be a regular bent at t_0^+.

This means that any valley on which either br_i or bl_j is incident at t_0^- will need to be updated using a shift event to reflect this swap of br_i and bl_j. Additionally note that, since br_i and bl_j also swap positions at t_0, any valley on which either br_i or bl_j is incident cannot die as a result of a BB collision between br_i and bl_j: its incident h-bend will be replaced by another h-bend after the collision. Similarly, since each regular bend is replaced by another regular bend at t_0, BB collisions cannot cause birth events. Thus, in conclusion, BB collisions cause a *shift event* if one of the colliding boundaries is an h-bend incident on a valley, and will not cause an event otherwise.

Type 3: Boundary-Point collision

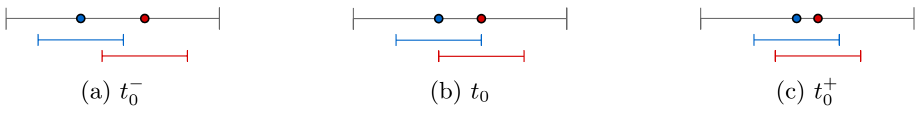


Figure 3.9: Points p_i and p_j for which $r_i(t_0) = p_j(t_0)$, $p_i(t_0) = l_j(t_0)$ and $p_i(t_0^-) < l_j(t_0^-)$.

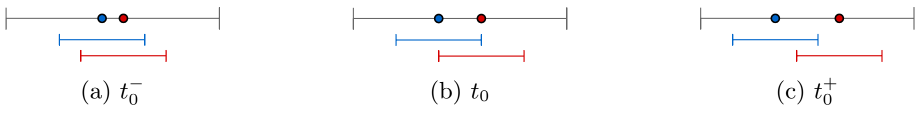


Figure 3.10: Points p_i and p_j for which $r_i(t_0) = p_j(t_0)$, $p_i(t_0) = l_j(t_0)$ and $p_i(t_0^-) > l_j(t_0^-)$.

Let a *BP collision* be a collision where two points collide with each others boundaries. W.l.o.g., assume $r_i(t_0) = p_j(t_0)$ and $p_i(t_0) = l_j(t_0)$. Then, we must have either $p_i(t_0^-) < l_j(t_0^-)$ (Figure 3.9) or $p_i(t_0^-) > l_j(t_0^-)$ (Figure 3.10). We will again exclusively look at the prior case, since the latter is very similar. A BP collision consists of two symmetric collisions: one between p_i and l_j, and one between r_i and p_j. Since these two collisions are symmetric, w.l.o.g. we will only look at the collision between p_i and l_j. We investigate how bp_i and bl_j are affected by this collision, and what events these effects can cause. To this end, we make two observations.

Since $p_i(t_0) = l_j(t_0)$, and t_0^- is arbitrarily close to t_0, we have:

$$|R_\ell(p_i(t_0^-))| = |R_\ell(l_j(t_0^-))| \tag{3.18}$$

$$|R_r(p_i(t_0^-))| = |R_r(l_j(t_0^-))|. \tag{3.19}$$

Then, at t_0, p_j is added to $R_r(p_i)$, and p_i is removed from $R_\ell(l_j)$ and added to $R_r(l_j)$. We get:

$$|R_\ell(p_i(t_0^+))| = |R_\ell(p_i(t_0^-))| \tag{3.20}$$

$$|R_r(p_i(t_0^+))| = |R_r(p_i(t_0^-))| + 1 \tag{3.21}$$

$$|R_r(l_j(t_0^+))| = |R_r(l_j(t_0^-))| + 1 \tag{3.22}$$

$$|R_\ell(l_j(t_0^+))| = |R_\ell(l_j(t_0^-))| - 1. \tag{3.23}$$

We can use these observations to investigate exactly how the type of p_i and l_j can change at t_0. We distinguish six cases, for each of which we can prove how the bp_i and bl_j change using a similar argument. One of these arguments is given here; for completeness, the rest can be found in Appendix A.

Assume $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-))$. By Equations 3.18 and 3.19, this also means $R_\ell(l_j(t_0^-)) = R_r(l_j(t_0^-))$. Thus, by Lemma 1 and 3, at t_0^- bp_i is a single-bend maximum and bl_j is an ending h-bend. This implies

$$\begin{aligned} & & |R_\ell(p_i(t_0^-))| = |R_r(p_i(t_0^-))| \\ \Rightarrow \quad & \text{(By Eq. 3.20 \& 3.21)} & |R_\ell(p_i(t_0^+))| + 1 = |R_r(p_i(t_0^+))| \end{aligned}$$

and

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| = |R_r(l_j(t_0^-))| \\ \Rightarrow \quad & \text{(By Eq. 3.22 \& 3.23)} & |R_\ell(l_j(t_0^+))| = |R_r(l_j(t_0^+))| + 2 \end{aligned}$$

This means, by Lemma 2 and 3, that at t_0^+ bp_i will be a starting h-bend and bl_j will be a regular bend. Using similar arguments, given in Appendix A, we can show that:

(Shift or death event)
If $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-))$, then bp_i is a single-bend maximum at t_0^- and becomes a starting h-bend at t_0^+. Also, bl_j is an ending h-bend at t_0^- and becomes a regular bend at t_0^+.

(Shift, birth, death, or no event)
If $R_\ell(p_i(t_0^-)) + 1 = R_r(p_i(t_0^-))$, then bp_i is a starting h-bend at t_0^- and becomes a regular bend at t_0^+. Also, bl_j is a regular bend at t_0^- and becomes a starting h-bend at t_0^+.

(Shift, birth, death, or no event)
If $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-)) + 1$, then bp_i is an ending h-bend at t_0^- and becomes a single-bend maximum at t_0^+. Also, bl_j is a starting h-bend at t_0^- and becomes a regular bend at t_0^+.

(Shift, birth, or no event)
If $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-)) + 2$, then bp_i is a regular h-bend at t_0^- and becomes an ending h-bend at t_0^+. Also, bl_j is a regular bend at t_0^- and remains regular at t_0^+.

(Shift, birth, or no event)
If $R_\ell(p_i(t_0^-)) + 2 = R_r(p_i(t_0^-))$, then bp_i is a regular bend at t_0^- and remains regular at t_0^+. Also, bl_j is a regular bend at t_0^- and becomes an ending h-bend at t_0^+.

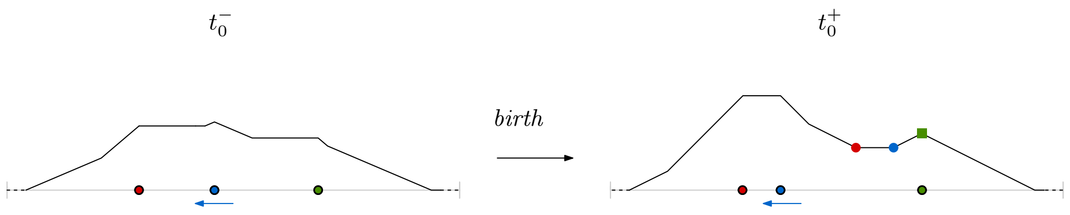


Figure 3.11: An example of a birth event taking place at t_0 due to a BP-collision between the blue and the green point.

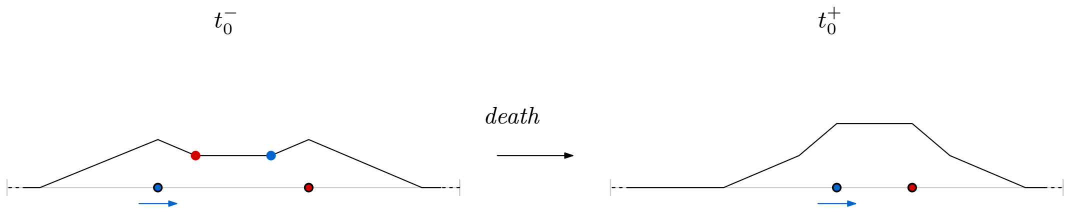


Figure 3.12: An example of a death event taking place at t_0 due to a BP-collision between the blue and the red point.

(No event)

Otherwise, both bp_i and bl_j are regular at t_0^- and remain regular at t_0^+.

Using these observations along with information about the neighbors of the changing bends, we can determine exactly what events are caused by a BP collision whenever one takes place. An example of a shift, birth and death event taking place due to a BP collision can be found in Figures 3.8 (top), 3.11, and 3.12, respectively.

3.3 KDS for maintaining the extrema

In the following section, we describe how to maintain the extrema of T_P when the points in the inducing point set P are moving. We assume the movement of points in P is characterized as described in Section 3.2. In order to maintain the extrema of T_P efficiently, we use the KDS framework as described in Section 2.3.

3.3.1 Initialization

All points $p_i \in P$ and their boundaries l_i and r_i are stored together in a single sorted list S of size $3n$, that can be maintained with a linear number of certificates using the KDS for maintaining a sorted point set [3]. We require S to be sorted, since this allows us to check for collisions between neighboring points/boundaries using a linear number of certificates (rather than a quadratic number). Because we are using the KDS framework, we need to store a list of certificates, as well as an event queue. The event queue will be stored as a priority queue such that the nearest event can be easily found. In addition, we will also maintain an auxiliary *interval tree* [10]. For each point p_i we add intervals $[p_i - \sigma, p_i + \sigma]$, $[l_i - \sigma, l_i + \sigma]$ and $[r_i - \sigma, r_i + \sigma]$ to the interval tree. Then, the nodes in the interval tree are augmented such that each node stores the type of the corresponding bend (regular, starting-/ending h-bend or maximum), as well as the number of points in the left- and right half of the corresponding interval: these represent the sizes of the left/right-ranges R_ℓ/R_r of the corresponding point/boundary. Note that, since we are merely interested in the sizes of the left/right-ranges of each point/boundary, maintaining an entire interval tree is not necessarily required. However, if we were to extend this KDS to work with, for example, different kernels, maintaining the auxiliary interval tree allows us to store additional information about these left/right-ranges.

	Type	**Certificate**
Point p_i	Maximum	$\lvert R_\ell(p_i)\rvert = \lvert R_r(p_i)\rvert$
	Left h-bend	$\lvert R_\ell(p_i)\rvert + 1 = \lvert R_r(p_i)\rvert$
	Right h-bend	$\lvert R_\ell(p_i)\rvert = \lvert R_r(p_i)\rvert + 1$
	Regular	$\lvert\lvert R_\ell(p_i)\rvert - \lvert R_r(p_i)\rvert\rvert > 1$
Left-boundary l_i	Left h-bend	$\lvert R_\ell(l_i)\rvert = \lvert R_r(l_i)\rvert + 1$
	Right h-bend	$\lvert R_\ell(l_i)\rvert = \lvert R_r(l_i)\rvert$
	Regular	$\lvert R_\ell(l_i)\rvert > \lvert R_r(l_i)\rvert + 1$ or $\lvert R_\ell(l_i)\rvert < \lvert R_r(l_i)\rvert$
Right-boundary r_i	Left h-bend	$\lvert R_\ell(l_i)\rvert + 1 = \lvert R_r(l_i)\rvert$
	Right h-bend	$\lvert R_\ell(l_i)\rvert = \lvert R_r(l_i)\rvert$
	Regular	$\lvert R_\ell(l_i)\rvert > \lvert R_r(l_i)\rvert$ or $\lvert R_\ell(l_i)\rvert + 1 < \lvert R_r(l_i)\rvert$

Table 3.1: The different types of critical certificates.

Each extremum in T_P is stored with a label indicating the indices and the types of the corresponding bend(s). They are maintained in a data structure that allows for polylogarithmic insertion and deletion operations, such as a balanced binary search tree (BST) [33]. This allows for efficient repairs to the KDS when a certificate is violated. This balanced BST can be initialized directly from the interval tree using Lemmas 1-3, and contains all extrema of T_P. To finish initialization, we create a list of certificates that hold at $t = 0$ (discussed below) and initialize the event queue accordingly.

To summarize, we maintain:

- A sorted list S representing the input point set P as well as the kernel boundaries of all points in P, maintained using the KDS for a sorted point set [3];
- A list of validating certificates;
- An event queue stored as a priority queue, such as a *heap* [9], containing an event for each certificate failure using the failure time as priority;
- An augmented *interval tree* [10], containing an interval of radius σ for each point and boundary in S. Each node in the interval tree stores the type of the corresponding bend, as well as the size of the left- and right range of the corresponding point/boundary;
- A balanced BST containing the extrema of T_P.

3.3.2 Certificates

As discussed in Section 3.2, the extrema of T_P only change at times at which collisions take place. Therefore, the KDS maintains the same certificates that are required for maintaining the sorted list S: for each pair of neighbors v, u in S, we maintain a *collision certificate* ($v \neq u$) that fails whenever these neighbors collide. When one of these certificate fails we need to update the auxiliary interval tree, certificate list and event queue to reflect these changes.

Additionally, for each node in the interval tree we maintain a *critical certificate* that compares the size of its left-range to its right-range according to Lemmas 1-3. The exact critical certificates maintained for each type of bend can be found in Table 3.1. Using these critical certificates we can detect when bends become extrema, and when extrema become regular bends. Note that these critical certificates can only fail when a collision certificate fails, as that is the only time at which the size of the left- and right-ranges change.

3.3.3 Repairing the KDS

After a collision certificate has failed, repairs to the list of certificates, the event queue, and the auxiliary interval tree must be performed. Repairs to the extrema of T_P are necessary only when

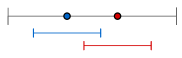
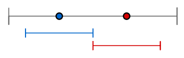
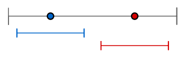
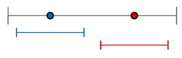
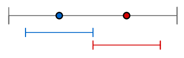
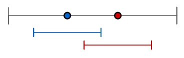
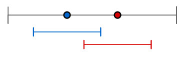
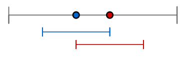
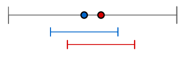
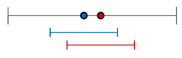
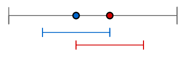
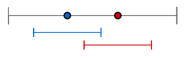
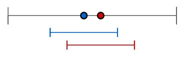
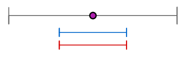
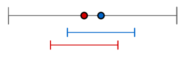
t_0^-	t_0	t_0^+	R_ℓ/R_r
			$R_r(r_1) -= 1$ $R_\ell(l_2) -= 1$
			$R_r(r_1) += 1$ $R_\ell(l_2) += 1$
			$R_\ell(p_1) += 1$ $R_\ell(p_2) += 1$ $R_\ell(r_1) += 1$ $R_r(r_1) -= 1$ $R_\ell(l_2) -= 1$ $R_r(l_2) += 1$
			$R_\ell(p_1) -= 1$ $R_\ell(p_2) -= 1$ $R_\ell(r_1) -= 1$ $R_r(r_1) += 1$ $R_\ell(l_2) += 1$ $R_r(l_2) -= 1$
			$R_\ell(p_1) += 1$ $R_r(p_1) -= 1$ $R_\ell(p_2) -= 1$ $R_r(p_2) += 1$ $R_\ell(r_1) += 1$ $R_\ell(r_2) -= 1$ $R_r(l_1) += 1$ $R_r(l_2) -= 1$

Table 3.2: A list of the 5 possible collisions that can occur, as well as the corresponding updates to left/right-ranges that need to be repaired (see Section 3.2.1).

a critical certificate fails. Therefore, collision certificate failures cause exclusively internal events, and critical certificate failures can additionally cause external events. Note that not all collision certificate failures cause a critical certificate to fail (see Section 3.2.1).

In any case, whenever a collision certificate fails we need to:

- Switch the locations of the nodes in the interval tree that correspond to the points/boundaries that collided;
- Update the sizes of R_ℓ/R_r of all points/boundaries involved in the certificate that failed according to Table 3.2;
- Update the certificates of the points/boundaries that caused the certificate failure, as well as those of their neighbors;
- Update the event queue with the newly created certificates.

During these repairs, a critical certificate may fail due to the changes to the left- and right ranges. If a certificate failed, it prompts an external event that updates the extrema of T_P (see Section 3.2.1) *unless* the certificate failure was the creation of an h-bend that is not part of an extremum, which can easily be checked by reading the type of the left/right neighbor depending on whether the new h-bend is starting or ending.

3.3.4 Analysis

Recall that, in Section 2.3, we discussed a number of quality measures that facilitate the analysis of the quality of a KDS: compactness, locality, (weak/strong) efficiency and responsiveness. We will see how the KDS described above performs in each of these four quality measures.

Compactness

The compactness of a KDS describes the worst-case size of the event queue at any one time. A KDS is considered compact if the number of events in the event queue is never more than polylogarithmic in the input size. First of all, our KDS maintains a sorted list of input points and the boundaries of their kernels. This can be done using $O(n)$ certificates [3]. These certificates for maintaining the sorted list are the same certificates as the collision certificates we use to track collisions between each pair of neighbors. Additionally, we have a single critical certificate for each point/boundary, which means there are $O(n)$ critical certificates. Since these are the only two types of certificates we need, this means we have a total of $O(n)$ certificates at any time. Since each certificate has at most one corresponding event in the event queue, the KDS is compact.

Locality

The locality of a KDS is determined by the maximum number of certificates that are dependent on a single input item. We call the KDS local if this number is at most polylogarithmic. In this case, it is relatively trivial to see that any one point appears in at most 9 certificates. Each point p_i appears in at most 6 collision certificates, since it shares a certificate with each neighbor of $l_i/p_i/r_i$. If none of $l_i/p_i/r_i$ neighbor each other, then p_i appears in a unique collision certificate for each. Additionally, p_i appears in 3 critical certificates, one for each of $l_i/p_i/r_i$. Therefore, in total, the worst-case number of certificates a single item can appear in is 9. Since this number is constant, the KDS is local.

Efficiency

The weak efficiency of a KDS depends on the ratio of the total number of events to the worst-case number of external events. A KDS is considered efficient if the total number of events processed by the structure is of the same order of (or only slightly larger than) the number of external events in the worst case. To be able to classify the worst case number of external events, we will consider the positions of the input point set to be determined by linear functions.

Lemma 4. *Let the movement of points in input point set P be described by linear functions over time. Then, in the worst case, the KDS requires $\Theta(n^2)$ external events and $\Theta(n^2)$ internal events.*

Proof. Observe that an external/internal event can only take place whenever a collision takes place. It is easy to see that, under linear motion, at most $O(n^2)$ collisions take place, since each point in P can collide with each other point in P at most once. Therefore, under linear motion, the number of external- and internal events necessary to track the extrema of T_P in the worst case is $O(n^2)$. We can actually show this bound to be tight.

We construct a scenario in which we need $\Omega(n^2)$ external and $\Omega(n^2)$ internal events to be able to correctly track the extrema of T_P. We place n points $p_1, \ldots, p_n$, such that $|p_1p_2| = \delta$ and $|p_ip_{i+1}| = |p_{i-1}p_i| + \delta$ for all $2 < i < n-1$ and some $\delta > 2\sigma$. Then, each point p_i moves linearly to the right at speed $s(p_i)$ such that $s(p_i) = s(p_{i-1})/2$ and $s(p_1) > 0$. This way, for $i < j$, all collisions between points p_i and p_j happen when $|p_{j-1} - p_j| > 2\sigma$ and $|p_j - p_{j+1}| > 2\sigma$. This means that each point p_i causes a constant number of external events every time it fully crosses another point p_j (see Figure 3.13). Since the points are placed such that each point collides with all other points, the number of external events in this situation is $\Omega(n^2)$. Therefore, the number external events to track the extrema of T_P is, in the worst case, $\Theta(n^2)$. In this same scenario, each point collides exactly once with each other point, causing a number of internal events in $\Omega(n^2)$. Thus, the number of internal events is also $\Theta(n^2)$. □

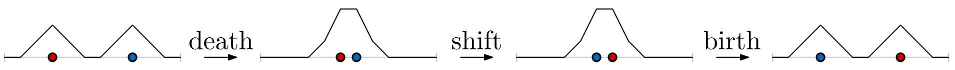


Figure 3.13: All external events that result from collisions between points p_i (red) and p_j (blue) if $|p_{j-1} - p_j| > 2\sigma$ and $|p_j - p_{j+1}| > 2\sigma$.

Since both the worst case total number of events and the worst case number of external events is $\Theta(n^2)$ under linear motion, the KDS is weakly efficient.

The strong efficiency of a KDS is the worst-case *ratio* of total events to external events in any allowed movement within the movement class we are restricted to. Recall that strong efficiency is different to weak efficiency in the sense that weak efficiency considers the worst-case number of external events in contrast to the worst-case ratio of total to external events. We show that the KDS is not strongly efficient by showing that the ratio of total to external events is $\Omega(n)$. While we may be able to find a higher bound, this linear lower bound is enough to show that the KDS is not strongly efficient.

We again restrict ourselves to linear movement. We can construct a situation where the number of external event is linear. Consider the following scenario: at $t = 0$, we place all points in P uniformly distributed such that we have $|p_{i-1}p_i| = |p_i p_{i+1}| < \frac{2\sigma}{n}$ for $2 \leq i < n$. This way, we have a single maximum at $t = 0$ that (partly) overlaps the kernels of all points. Then, each point p_i moves in a linear motion to the right at speed $s(p_i)$ such that $s(p_i) = s(p_{i-1})/2$ and $s(p_i) > 0$. This way, each point induces 2 (if single bend maximum) or 4 (if plateau) shift events when they cross the maximum. Therefore, each point induces a constant number of external events, which means that in this construction there are $\Omega(n)$ external events, while the situation requires $\Omega(n^2)$ internal events (one for each collision). Thus, the KDS is not strongly efficient.

Responsiveness

The responsiveness of a KDS is determined by the worst-case cost of resolving a certificate failure. In our setting this concerns handling changes to the interval tree, changing the extrema of T_P, and updating the event queue. The changes to the interval tree include swapping nodes by inserting and deleting nodes (at most 6 nodes are affected, corresponding to the points/boundaries of the colliding nodes) and updating the associated counters in the nodes (at most 12 counters are updated, two for each affected node). All of these only require a constant number of $O(\log n)$ time operations in an interval tree: insertion/deletion is done in $O(\log n)$ time, and since each point has pointers to and from its boundaries, all affected nodes are found in constant time. If a critical certificate is violated, either a shift of a birth/death event must happen. These two events only require a constant number of search (and update), insert and delete operations, all of which can be done in $O(\log n)$ time in our balanced binary search tree. Lastly, we need to recompute the priorities of certificates associated with each affected node in the event queue. Inserting and deleting to/from an event queue can be done in $O(\log n)$ time. Since a constant number of nodes are affected and each node has a constant number of associated certificates, we can thus update the event queue in $O(\log n)$ time. This means that we need a constant number of $O(\log n)$ operations to handle certificate failures and repair the KDS, meaning the KDS is responsive.

Theorem 1. *Let P be a moving point set of size n, where the location of each point is determined by a linear function of time. Let KDE_P denote the kernel density estimation function of P, obtained using the linear kernel. A list of extrema of KDE_P can be maintained by a KDS that is compact, local, responsive and weakly efficient.*

3.4 Coresets

In data-intensive applications, P may be significantly large. While the KDS we described previously is theoretically efficient and adheres to all quality measures, processing a number of events that is quadratic in the size of the input can be prohibitively expensive in very large data sets.

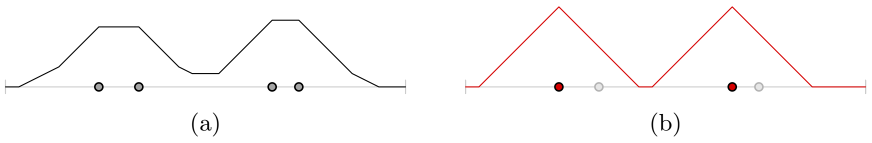


Figure 3.14: A simple example of a coreset. The example point set in (a) consists of four points. Using the coreset in (b), highlighted in red, results in a similar terrain using half the number of points.

To get around this, this section explores whether we can find and maintain a point set Q that is significantly smaller than P, of which the estimated density function is "similar" to that of P. Then, instead of maintaining the extrema of the KDE of P, we can maintain the extrema of the terrain of the smaller point set Q. This results in a lower number of events being processed (depending on the size of Q), while (due to the similarity of Q and P) the maintained extrema are guaranteed to be "similar" to the actual extrema of T_P. Such an approximation Q of P is often referred to as a *coreset* of P. A small example of a coreset can be seen in Figure 3.14. To find a coreset of P, we investigate how we can extend a common approximation method for static kernel density estimation to the kinetic setting.

3.4.1 Static coresets

We call a coreset Q an *ε-approximation* of a point set P if and only if, given error parameter $\varepsilon > 0$, we have [36]:

$$\max_{x\in\mathbb{R}} |\mathrm{KDE}_P(x) - \mathrm{KDE}_Q(x)| \leq \varepsilon.$$

There has been extensive work done on the sampling of point sets for kernel density estimation, especially in one dimension. A relatively common baseline method is simply *random sampling* (RS) [29, 36]. If we let Q be a random sample from P of size $O((1/\varepsilon^2)\log(1/\delta))$, then with probability at least $1-\delta$ the random sample Q ensures that $\max_{x\in\mathbb{R}} |\mathrm{KDE}_P(x) - \mathrm{KDE}_Q(x)| \leq \varepsilon$. The technique is widely used because of its simplicity, often as a preprocessing step for data sets at extremely large scales. However, since it is not *guaranteed* to produce an ε-approximation, it does not always work in practice.

Another, slightly more advanced sampling method is the *grid-based approximation* [37], which divides the vertex space into a grid. For the one-dimensional case, this means that we define a *grid* G_γ to divide the vertex space into a set of *grid intervals* of width γ. For a grid interval g, let P_g denote the points from P that are inside of g. Then, Q is constructed as a *weighted* sample, in which each vertex has a different kernel height. For each grid interval g we add a single vertex p_g to Q at the mean position $\frac{1}{|P_g|}\sum_{p\in P_g} p$ with weight $|P_g|$, representing all points in g. This approach, however, does not guarantee the error bound for the entire KDE function, but only for a subset above a specified height (based on the size of the grid intervals).

The approximation approach we use is *sort-selection* [36], which is a consistent ε-approximation that generates a coreset of $O(\frac{1}{\varepsilon})$ size which works well despite its relative simplicity. Contrary to random sampling, this approach is guaranteed to produce an ε-approximation. Also, unlike the grid-based approximation approach, sort-selection ensures that the error bound holds for the entire domain instead of only a subset of the domain. The sort-selection approach utilizes the following lemma [36]:

Lemma 5. *Consider a sorted, one dimensional point set $P = \{p_1, p_2, \ldots, p_n\}$ where $p_i \leq p_{i+1}$ for all $1 \leq i \leq n$. Let $P_j = \{p_i \in P | (j-1)\varepsilon n < i \leq j\varepsilon n\}$ for integer $j \in [1, \lceil 1/\varepsilon\rceil]$ such that $P = \bigcup P_j$. Then, for any $Q = \{q_1, q_2, \ldots, q_{\lceil 1/\varepsilon\rceil}\}$ such that each $q_j \in P_j$, we have $\max_{x\in\mathbb{R}} |KDE_P(x) - KDE_Q(x)| \leq 2\varepsilon$. If each $q_j = p_{\lceil (j-1/2)\varepsilon n\rceil}$, then $\max_{x\in\mathbb{R}} |KDE_P(x) - KDE_Q(x)| \leq \varepsilon$.*

As such, to generate a valid ε-approximation, we can simply select all $p_{\lceil (j-\frac{1}{2})\varepsilon n\rceil}$ from P with $j \in [1, \lceil\frac{1}{\varepsilon}\rceil]$ to put into Q. Intuitively, the approach uses a uniform sampling approach over a sorted point set, that selects $\frac{1}{\varepsilon}$ points to put into Q and that guarantees an error bound of ε. The benefit of using this approach over, for example, the grid-based approach described previously, is that sort-selection achieves an error bound over *all* locations in the domain. Additionally, sort-selection selects points based on their indices, which we can utilize when we want to maintain the sort-selection coreset over time.

In the remainder of this chapter, we will describe how to augment our KDS described in Chapter 3 to maintain coreset Q as well as the extrema in the terrain induced by KDE_Q rather than KDE_P. As we will see, using this approach reduces the number of events required to maintain these extrema, while providing a guarantee on what extrema from T_P are actually maintained.

3.4.2 Kinetic sort-selection

In order to use the sort-selection coreset in the KDS, we must first be able to kinetically maintain it. The sort-selection coreset utilizes the fact that the input point set P is maintained in a sorted manner. However, maintaining the entire point set P as a sorted list requires $\Theta(n^2)$ external events [3], which we want to avoid if possible. Fortunately, we are only interested in the indices of points in the coreset Q, not of all points in P. Therefore, the pointset P need not be maintained in a sorted manner entirely; if coreset Q is initialized correctly, it can be maintained with relative ease as follows.

Observe that if two points in a sorted list P collide, their indices switch. Therefore, if coreset Q is initialized correctly, coreset Q only changes whenever a point $q \in Q$ collides with a point $p \in P$. Upon such a collision, since points in Q are selected based on their indices, we update the coreset by simply replacing q by p in Q. We call the certificates that check for collisions between Q and P *coreset certificates*, since they facilitate the maintenance of the coreset.

There are multiple ways in which these coreset certificates can be constructed. If we want to minimize the number of coreset certificates, we could maintain the neighbors of all points in Q. That way, we only require two coreset certificates for each point in Q to check for collisions with their neighbors. Maintaining the neighbors of points in Q, however, is non-trivial; ultimately it would require maintenance of the entire point set P in a sorted manner, which is not efficient. Alternatively, we can "inverse" this approach and maintain, for each point $p \in P$ what points in Q are the nearest to their left and right. This means $O(n)$ coreset certificates are necessary, but the maintenance becomes more manageable; instead of maintaining the entirety of P in a sorted manner, we now only need to track the nearest points from Q to the left and right of each point in P. Since these change only when a point from Q collides with a point from P, no additional events need to be triggered to maintain these coreset certificates: if they are initialized correctly, they can simply be updated whenever a coreset certificate fails.

Now that coreset Q can be kinetically maintained, coreset Q can simply be used in place of input point set P as input to the kinetic data structure described in Chapter 3. This means that, rather than maintaining the extrema of terrain T_P, the KDS instead maintains the extrema of terrain T_Q, which approximates T_P. The KDS again maintains a list of certificates, a priority queue representing the event queue, the augmented interval tree (now only containing points from Q and their boundaries), and the balanced BST containing the extrema of T_Q.

3.4.3 Analysis

We analyze the KDS using the quality measures described in Section 2.3.1, with the addition of the maintenance of the sort-selection coreset.

Compactness

The compactness of the KDS described in Chapter 3 is linear in the number of points in the input set. Since we now maintain $|Q| = \frac{1}{\varepsilon}$ points, we have $O(\frac{1}{\varepsilon})$ collision- and critical certificates. However, in order to maintain Q we also need coreset certificates. Recall that, to maintain Q, we need a coreset certificate for each point in P that fails whenever it collides with a point from Q.

Since, for each point in P, we track which points from Q lie closest to its left and right, we can do so using a number of certificates linear in the size of P, which is $O(n)$. Thus, the KDS remains compact when using the sort-selection coreset.

Locality

The locality of the KDS becomes dependent on the number of coreset certificates in which a single point in Q can appear. Similar to what was described in Section 3.3.4, each point $q \in Q$ appears in at most 6 collision certificates and 3 critical certificates. In addition, however, to maintain Q each point $q \in Q$ appears in a coreset certificate for each point from P in between q and the neighbors of q in Q. Observe that, since we use sort-selection, the indices of points in Q are $\lceil (j - 1/2)\varepsilon n \rceil$ for $j \in [1, \lceil \frac{1}{\varepsilon} \rceil]$. This means that there are $O(\varepsilon n)$ points in between each neighboring pair of points from Q. Thus, each point in Q appears in $O(\varepsilon n)$ coreset certificates. This means that, using the sort-selection coreset, the KDS is no longer local.

Efficiency

Finding the efficiency of the KDS using the sort-selection coreset is somewhat more complicated. To argue why, we must first discuss another classical open problem in combinatorial geometry. Let an *arrangement of lines* be the subdivision of the two-dimensional plane $\mathbb{R}^2$ formed by a collection of lines. The *k-level* of an arrangement of lines is the polygonal chain formed by the line segments that have exactly k line segments below them. For an example of an arrangement of lines and its 3-level, see Figure 3.15. Finding matching bounds on the complexity of a k-level is open. The best known bounds on the complexity of a single k-level in a line arrangement are $O(nk^{\frac{1}{3}})$ and $nk^{\Omega(\sqrt{\log k})}$, respectively [11].

Additionally, in the upper bound analysis below, we will make use of the following observation.

Observation 1. *Let c_1 and c_2 be two polygonal chains consisting of n and m vertices, respectively. If both c_1 and c_2 are monotone with respect to some line ℓ, then the number of intersections between c_1 and c_2 is at most $O(n + m)$.*

Proof sketch. Intuitively, this observation can be shown to be correct using a charging argument. We charge each intersection i between c_1 and c_2 to a unique vertex v of c_1 or c_2. Let i' and v' be projections of i and v on line ℓ. We charge i to vertex v such that $|i'v'|$ is minimized over all vertices to the right of the ray through i' orthogonal on ℓ. This way, each intersection must be charged to a unique vertex v, since there cannot be more than one intersection between the same two line segments of c_1 and c_2. □

Using the observations above, we are able to prove an upper bound on the number of events caused by the KDS using the sort-selection coreset:

Lemma 6. *Let the movement of points in input point set P be described by linear functions over time. Then, in the worst case, the KDS using the sort-selection coreset causes $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$ internal events.*

Proof. Internal events are caused either by a failing coreset certificate, or by a failing collision certificate between two points in Q. We will handle these two certificates separately.

We show that finding the worst-case number of violated coreset certificates in our KDS using the sort-selection coreset under linear motion is equal the finding the complexity of $\frac{1}{\varepsilon}$ k-levels. Since the complexity of a single k-level is $O(nk^{\frac{1}{3}})$ and k is in $O(n)$, this means that the worst-case number of events is $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon})$.

Consider a plot of the linear functions that describe the points in P over time interval $[-\infty, \infty]$, where the coordinates of the point and the time are plotted on the vertical and horizontal axes, respectively (see Figure 3.16). This plot consists of an arrangement of n lines, where each line describes the movement of a single point in P. Intersections between two lines in this arrangement

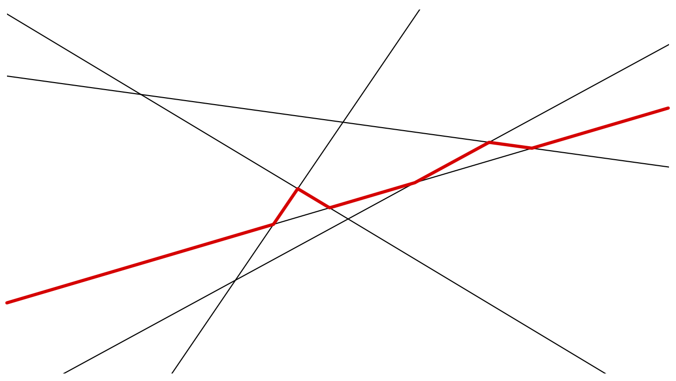


Figure 3.15: An example of an arrangement of lines. The 3-level of this arrangement is highlighted in red.

represents a collision between the two corresponding points. Observe that, by definition, the k-level of this line arrangement represents the path of the point in index $k+1$ in sorted list P. Thus, the number of vertices in a k-level represents the number of times the point at index $k+1$ changed. Since the points in Q are selected based on their index, the number of collisions involving points from Q is equal to the number of vertices in all $(\lceil(j-\frac{1}{2})\varepsilon n\rceil+1)$-levels for $j \in [1, \lceil\frac{1}{\varepsilon}\rceil]$.

The number of vertices in a single k-level is $O(nk^{\frac{1}{3}})$, and we have $k = (\lceil(j-\frac{1}{2})\varepsilon n\rceil+1) = O(n)$ for $j \in [1, \lceil\frac{1}{\varepsilon}\rceil]$. Therefore, we have $O(\frac{1}{\varepsilon})$ k-levels that correspond to indices that get put into Q, each involved in $O(nk^{\frac{1}{3}}) = O(n^{\frac{4}{3}})$ collisions. This results in a total of $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon})$ collisions, each of which causes a single coreset certificate to fail.

Note that collision certificates fail not only when the actual points of Q collide, but also when their boundaries collide. Therefore, we extend the line arrangement described above to include the trajectories of the kernel boundaries of each point. Observe that the trajectories of the kernel boundaries l_i and r_i of each point p_i can be found by simply displacing the line representing p_i by $-\sigma$ and σ, respectively.

As we saw previously, the trajectory of each point in Q is represented by a k-level in this line arrangement. Note that a k-level is t-monotone [30]. Therefore, the paths of points in Q are described by t-monotone, polygonal chains consisting of $O(n^{\frac{4}{3}})$ vertices [11]. Note also that the trajectories of the left- and right kernel boundaries l_q and r_q of each point $q \in Q$ can be represented by a copy of this t-monotone, polygonal chain, again displaced by $-\sigma$ and σ, respectively (see Figure 3.16, highlighted in red). Now, to find the number of failing collision certificates, we must find the number of intersections between the paths of points and boundaries in Q.

Each path of points and boundaries in Q is described by a t-monotone, polygonal chain of complexity $O(n^{\frac{4}{3}})$. According to Observation 1, two t-monotone, polygonal chains of complexity $O(n^{\frac{4}{3}})$ can intersect at most $O(n^{\frac{4}{3}}+n^{\frac{4}{3}}) = O(n^{\frac{4}{3}})$ times. Therefore, the total number of intersections between paths of points and boundaries in Q is equal to $O(n^{\frac{4}{3}}) \cdot (3|Q|)^2 = O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$. This means that, under linear motion, at most $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$ collision certificates are violated in the worst case. Thus, assuming linear motion, in the worst case $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon})$ coreset certificates and $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$ collision certificates are violated. This yields a total of $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon}) + O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2}) = O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$ internal events. □

External events can only take place whenever a collision certificate is violated. As stated above, under linear motion at most $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$ collision certificates are violated. Therefore, the number of external events is also at most $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$ in the worst case. Thus, the use of the sort-selection coreset reduces the total number of events from $\Theta(n^2)$ to $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$.

Note that these upper bounds on the number of events (both internal and external events) are not tight. By nature of the fact that these bounds are based on the complexity of k-levels in line

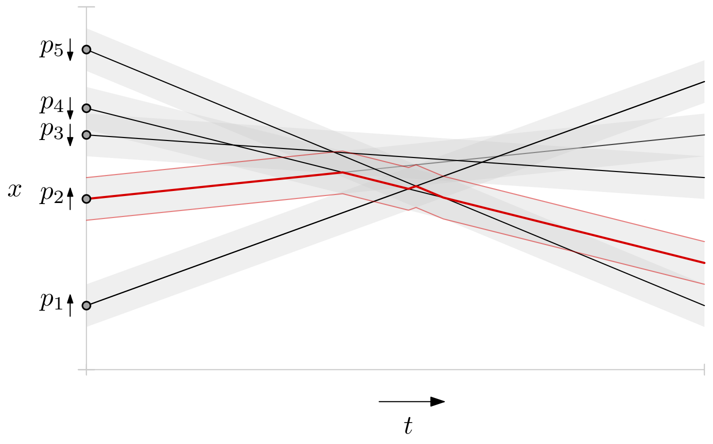


Figure 3.16: An example of an arrangement of lines constructed from a moving point set P as detailed in the proof of Lemma 6. The gray cylinders indicate the kernels of the points. The highlighted red lines indicate the point at index 2 in sorted list P, as well as the left- and right boundary of this point.

arrangement, finding matching bounds for the number of events remains open. For the number of external events, we can provide the following lower bound:

Lemma 7. *Let the movement of points in input point set P be described by linear functions over time. Then, in the worst-case, the KDS using the sort-selection coreset requires $\Omega(\frac{n}{\varepsilon})$ external events.*

Proof. Since we use sort-selection to construct Q, there are $\varepsilon n - 1$ vertices from P in between any two vertices $q_1, q_2 \in Q$ (if n is large enough). Observe that, during movement, a point $q \in Q$ is restricted to moving between its to neighbors in P; whenever q crosses one of its P neighbors, the crossed neighbor replaces q in Q, since they switch indices in P. In order to induce an external event, a point from Q must cross a point/boundary belonging to another vertex in Q. Thus, we construct our instance as follows: given some value of ε, at $t = 0$ points in P are placed such that $|p_i p_{i+1}| = \frac{\sigma}{\varepsilon n} + \hat{\varepsilon}$ for all $1 \leq i < n$ for some arbitrarily small $\hat{\varepsilon} > 0$. This ensures that each point in Q is slightly outside of the kernels of each of their neighbors in Q at $t = 0$. Then, each point p_i linearly moves to the left with speed $s(p_i)$ defined as $s(p_i) > \frac{s(p_{i-1})}{\varepsilon n \hat{\varepsilon}} \cdot \sigma$. This way, each point q in Q is involved in a BB collision with its left-neighbor in Q, then q is involved in a PP collision p_l in P (meaning p_l replaces q in Q), after which q is involved in a PP collision with its left-neighbor in Q. This means that, after these three collisions, q is again in Q and recursively triggers these three collisions for each point in Q. This process repeats until all points in P have crossed the right-most point in P, triggering $|P| \cdot |Q| = \frac{n}{\varepsilon}$ external events. Thus, the number of external events triggered by the KDS using the sort-selection coreset is, in the worst case, $\Omega(\frac{n}{\varepsilon})$. □

Responsiveness

The responsiveness of the KDS is no longer completely dependent on the repairs required after a collision- or critical certificate fails. As discussed in Section 3.3.4, the repairs required after a failed collision- or critical certificate take an amount of time polylogarithmic in the size of the maintained point set: $O(\log(\frac{1}{\varepsilon})$. When a coreset certificate between $q \in Q$ and $p \in P$ fails, however, the repairs may take more time. First, we have to update Q by replacing q by p. Then, we update the pointers from p and q that point to the nearest point from Q to their left and right. Since Q is sorted by nature of how it is constructed, we can do all this in $O(1)$. Afterwards, however, we have to update the coreset certificates themselves, as well as the event queue. All coreset certificates that depend on q must be updated, of which there can be $O(\varepsilon n)$. Updating each certificate itself takes $O(1)$ time, as it is simply replacing q by p in each certificate. Rescheduling a certificate in the event

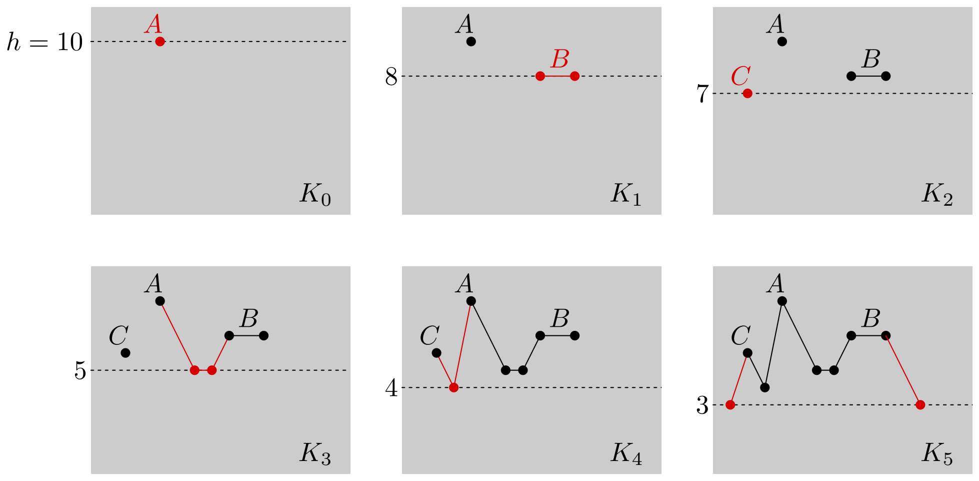


Figure 3.17: An example of a filtration of the simplicial complex K_5.

queue, however, requires a deletion and an insertion. Both of these operations require an amount of time that is polylogarithmic in the size of the event queue. Since, as we saw earlier, there can be $O(n)$ coreset certificates at a time, the size of the event queue can be $O(n)$. Therefore, the time to repair $O(\varepsilon n)$ coreset certificates can, in total, be $O(\varepsilon n \log n)$. This means that the KDS is no longer responsive when using the sort-selection coreset.

3.4.4 Persistence bounds

In this section, we aim to investigate how well the extrema of terrain T_Q approximate the extrema of terrain T_P. More specifically, we provide a guarantee that there exists an injection from the "most relevant" extrema in terrain T_P to the extrema of T_Q. If we are able to make such a guarantee, then we can be sure that the terrain T_Q actually maintains the "most relevant" extrema of terrain T_P. To this end, we desire a way to quantify the relevance of the extrema of T_P.

To do so, we require a number of tools from computational topology. Let a *d-simplex* be a generalization of a triangle to d dimensions. In our scope, it is enough to recognise a *0-simplex* as a point, and a *1-simplex* as a line segment. Using these definitions, we look at T_P as a *simplicial complex* [12]. In general, a simplicial d-complex is simply a set composed of d-simplices and simplices of lower dimensions. In our setting, since T_P is described by a one-dimensional function, T_P can be described by a simplicial 1-complex, which consists exclusively of points (corresponding to the bends in the terrain) and the line segments between these points. Let this simplicial complex describing T_P henceforth be referred to as K_P. Similarly, we refer to the simplicial complex describing T_Q as K_Q.

To quantify a measure of relevance for each of our extrema, we will create a *filtration* of the simplicial complex described by T_P. A filtration of a simplicial complex K is, in general, any sequence of simplicial complexes $K_0, K_1, \ldots, K_m$ such that $K_0 \subseteq K_1 \subseteq \cdots \subseteq K_m = K$. Intuitively, a filtration describes a step-wise assembly of simplicial complex K. Every step along the way also describes a simplicial complex K_i, which describes a subset of every simplicial complex K_j where $0 \leq i < j \leq m$. Depending on how we construct this filtration over K_P, studying it can reveal useful information about the topology of K_P, and thus about the shape of the terrain T_P.

In order to retrieve a useful filtration of K_P that can help us in studying maxima, we create a filtration as follows. We start with a simplicial complex K_0 that consists exclusively of the simplex/simplices that describe the *highest* maximum of T_P. If the highest maximum of T_P is a single-bend maximum, this will simply be a point. If the highest maximum is a plateau, K_0 will consist a line segment representing this plateau, as well as two points representing the incident h-bends. Next we search for the second highest *extremum* (so: maximum *or minimum*). K_1 consists of the simplices describing this second highest extremum, as well as each simplex that

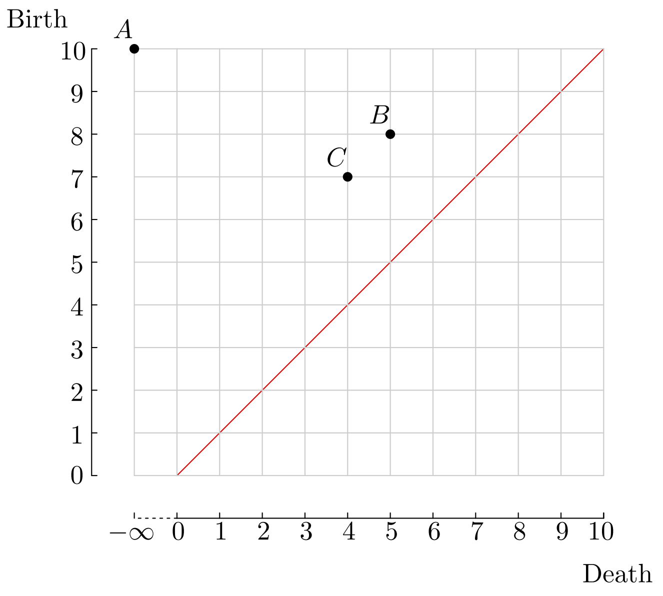


Figure 3.18: The persistence diagram of the three maxima from Figure 3.17. Note that, since A never dies, its moment of death is set to $-\infty$.

describes a bend or a line segment in T_P that is *completely above* this second highest extremum. Note that, since K_0 only consists of the highest maximum, $K_0 \subseteq K_1$. Now, to construct K_2, we take the third highest extremum and add each simplex that is completely above that. For K_3 we do so with the fourth highest extremum, and so forth. Each time we encounter two extrema of equal height, we add both of them to the simplicial complex. Once we have iterated over all extrema, we will have a filtration that describes how to construct K_P from the top down. See Figure 3.17 for an example.

A simplicial complex can consist of multiple connected components. Consider a filtration of K_P as described above, from the top down. In this filtration, each maximum creates a new connected component, and each minimum merges two existing connected components. For a connected component γ, we say γ is *born* at K_i if it is created by the maximum that is newly included in K_i (so, $\gamma \in K_i$ but $\gamma \notin K_{i-1}$ for $0 < i \leq m$). When two connected components merge at K_j, we consider the older component of the two to persist, meaning the younger component ceases to exist. We say connected component γ *dies* at K_j if it is merged in K_j with an older connected component. If γ is born at K_i and dies at K_j, then we call the difference in height between the extrema corresponding to K_i and K_j, respectively, the *persistence* of γ. For example, if γ is born due to a maximum at height h_{max} and dies due to a minimum at height h_{min}, then the persistence of γ is equal to $h_{max} - h_{min}$. If γ is born at K_i but never dies, we set its persistence to infinity. Since our goal is to quantify a degree of relevance to each extremum, we consider the persistence of an extremum equal to the persistence of the connected component that is born or dies at that extremum.

To visualize the persistence of extrema, we can create a scatterplot of all connected components based on their birth- and death height. If a connected component does not die, we place it at $-\infty$. We call such a plot a *persistence diagram.* For an example of a persistence diagram of the maxima of the filtration in Figure 3.17, see Figure 3.18. Note that, in such persistence diagrams, the diagonal plays a special role; any point in the persistence diagram lies above the diagonal, since each connected component must be born *before* it can die.

The notion of persistence gives us a quantitative measure on the relevance of an extrema, based on how "pronounced" it is in terrain T_P. A low persistence extremum is not considered very relevant, and can often be attributed to noise. Conversely, extrema with a very high persistence are considered relevant features in the data. In the remainder of this section we show, given an

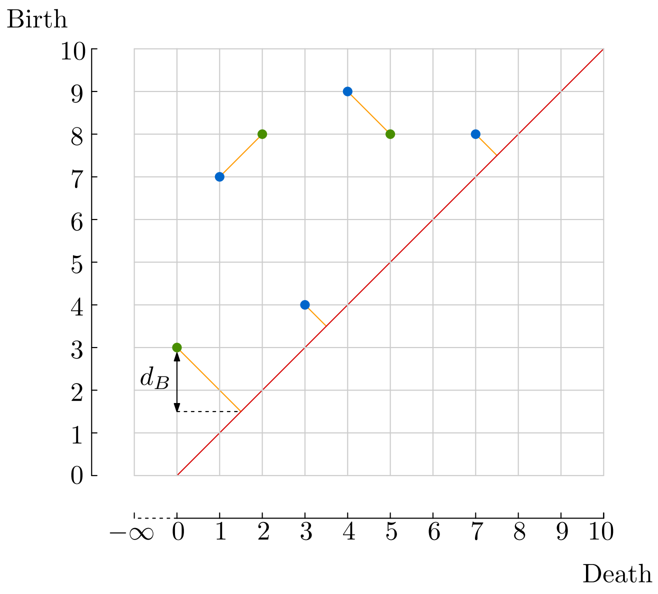


Figure 3.19: An example of how we compute the bottleneck distance between two persistence diagrams D (blue) and D_2 (green). The optimal matching Φ is indicated by the yellow lines.

ε-approximation Q of P, that there exists an injection from all extrema in T_P of persistence equal or higher than 2ε to the extrema in T_Q. This implies that the most relevant extrema from T_P (namely, those of persistence $\geq 2\varepsilon$) are maintained in T_Q.

To this end, we use a previous result that is concerned with the *stability* of persistence diagrams [8]. That is, this result quantifies how a persistence diagram can change due to small changes to the corresponding simplicial complex. Let the persistence diagrams of simplicial complexes K_P and K_Q be denoted $D(K_P)$ and $D(K_Q)$, respectively. In order to measure how similar K_P and K_Q are, we measure the similarity between their persistence diagrams $D(K_P)$ and $D(K_Q)$. We do so as follows. Given two points $p, q \in \mathbb{R}^2$ given by coordinates $p = (p_1, p_2)$ and $q = (q_1, q_2)$, let the *Chebyshev distance* [35] $||p - q||_\infty = \max(|q_1 - p_1|, |q_2 - p_2|)$. In order to compare $D(K_P)$ and $D(K_Q)$, we define the *bottleneck distance* between these persistence diagrams as

$$d_B(D(K_P), D(K_Q)) = \inf_{\phi} \sup_{x \in D(K_P)} ||x - \phi(x)||_\infty,$$

where ϕ is a matching between $D(K_P)$ and $D(K_Q)$. Here, in order to avoid cardinality mismatches, we allow any point in $D(K_P)$ or $D(K_Q)$ to additionally be matched to its projection on the diagonal. Let the optimal matching used to calculate the bottleneck distance be denoted Φ. Note that any point from $D(K_P)$ that is matched to its projection on the diagonal by optimal matching Φ corresponds to a connected component in K_P that is not actually matched to a component in $D(K_Q)$. See Figure 3.19 for an example.

At this point, we can utilize the previous work on the stability of persistence diagrams. In their work, Cohen-Steiner et al. [8] conclude the following. Let $f, g : \mathbb{R} \longrightarrow \mathbb{R}$ be functions over domain $\mathbb{R}$, both of which have a finite number of critical points. Given corresponding persistence diagrams $D(f)$ and $D(g)$, we have $d_B(D(f), D(g)) \leq \max_{x \in \mathbb{X}} |f(x) - g(x)|$. We use this result to bound the bottleneck distance between $D(K_P)$ and $D(K_Q)$. Recall that we construct K_P and K_Q from functions KDE_P and KDE_Q, respectively. Given the notion of the bottleneck distance, we can therefore use this result to conclude that

$$d_B(D(K_P), D(K_Q)) \leq \max_{x \in \mathbb{R}} |\text{KDE}_P(x) - \text{KDE}_Q(x)|$$

Since KDE_Q is an ε-approximation of KDE_P, this means that the bottleneck distance between the persistence diagrams of K_P and K_Q satisfies $d_B(D(K_P), D(K_Q)) \leq \varepsilon$.

Observe that the persistence of an extremum corresponding to connected component γ in K_P is reflected in the persistence diagram $D(K_P)$ as the distance of p_γ to the diagonal, where p_γ is the point in $D(K_P)$ corresponding to γ. More precisely, the persistence of γ is equal to $2 \cdot ||p_\gamma - p'_\gamma||_\infty$, where p'_γ is the projection of γ on the diagonal. Since the bottleneck distance between $D(K_P)$ and $D(K_Q)$ is $d_B(D(K_P), D(K_Q)) \leq \varepsilon$, we know that any point p_γ with $||p_\gamma - p'_\gamma||_\infty \geq \varepsilon$ cannot be matched to its projection on the diagonal by optimal matching Φ (otherwise, the bottleneck distance would be larger than ε). Because the persistence of γ is equal to $||p_\gamma - p'_\gamma||_\infty$, this means that Φ must match any point p_γ with persistence $\geq 2\varepsilon$ to a point in $D(K_Q)$. Thus, if we take optimal matching Φ and discard any matching of points in $D(K_P)$ and $D(K_Q)$ to the diagonal, we obtain an injection from the points in $D(K_P)$ with persistence $\geq 2\varepsilon$ to the points in $D(K_Q)$. Note that each point in $D(K_P)$ represents a connected component in K_P, and each connected components represents the maximum at which it is born *and* the minimum at which it dies. This means that we can find an injection of all maxima and minima in T_P of persistence $\geq 2\varepsilon$ to the maxima and minima in T_Q, respectively.

The existence of such an injection shows that the extrema in T_P are actually relatively well approximated by the extrema in T_Q. Since persistence provides a measure of relevance to the extrema of T_P, the fact that any extremum of persistence 2ε and higher are maintained in terrain T_Q implies that the most relevant extrema of T_P are still present in T_Q. Thus, while the use of the sort-selection coreset may cause some loss of information, the most relevant extrema are still maintained.

In summary, using the sort-selection coreset in the KDS in place of the original input point set has both benefits and drawbacks. The most significant advantage of using the coreset is that the number of processed events under linear motion is brought down from $O(n^2)$ to $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$. While this may not be a big improvement in small data sets, in very large data sets this can actually make a big difference. As we saw previously, we can achieve this improvement while losing little information about the most relevant extrema of T_P. There are two major drawbacks, however. First of all, the use of the coreset means that the KDS is no longer local: the number of certificates that depend on a single object is increased from $O(1)$ to $O(\varepsilon n)$. This means that changes in the trajectory of an entity may be relatively costly. Additionally, partially due to the loss of locality, the KDS is also no longer responsive: repairing the KDS may take $O(\varepsilon n \log n)$ time, which is quite a lot of compared to the $O(\log n)$ time it previously took. The main trade-off to consider here is therefore whether processing $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon^2})$ events that can take $O(\varepsilon n \log n)$ time each is more efficient than processing $O(n^2)$ events that can take $O(\log n)$ time each. Note that the loss in precision presented by the coreset can be regarded as a benefit. Often, extrema of very low persistence are not indicative of actual features of the data, but are merely an artifact of noise in the data. Therefore, not maintaining such irrelevant extrema provides the benefit of filtering out noise from the data.

In order to make the coreset more efficient, a number of extensions could be possible. For example, Lemma 5 also allows for a less strict set of vertices to be selected. Even though this would result in a 2ε-approximation, it may allows us to further reduce the number of events. Another example of an improvement could be to use other kinetic data structures, such as *kinetic heaps* [4], to maintain Q more efficiently. If we store a kinetic heap containing all points in P for each point in Q, it may be possible to maintain Q more efficiently than using the coreset certificates we proposed in this section. To check the feasibility of this approach, however, more investigation is required.

Chapter 4

Model-based group classification

This chapter investigates how we can use model-based movement analysis to classify behavioral movement phases. In Section 4.1 we discuss the terminology for movement data used throughout this chapter. In Section 4.2 we provide background on the movement model used in behavioral change point analysis model [15], as well as the segmentation and classification algorithms by Alewijnse et al. [2]. Additionally, we will describe in more detail how the segmentation approach is extended to group segmentation by Mols [25] using the movement model used in behavioral change point analysis. Then, in Section 4.3 we show how to apply the single-trajectory discrete classification algorithm by Alewijnse et al. to segmented groups of trajectories, and provide promising experimental results of a proof-of-concept implementation that demonstrates the utility of this approach.

4.1 Terminology

The terminology we use is similar to that used in previous work [2, 25]. We start by discussing the concept of the (sub-)trajectories themselves, then give a definition of a segmentation of a set of trajectories, and finally discuss the concept of classification.

We define a *trajectory* τ as a discrete sequence of timestamped, two-dimensional coordinates that represent a continuous motion through the sampled points. To visualize a trajectory, we linearly interpolate the movement between each pair of subsequent observations. A trajectory τ of length m can be represented by an ordered set containing the coordinates and the timestamps of each observation: $\tau = \{(z_1, t_1), (z_2, t_2), \ldots, (z_m, t_m)\}$. Here, $z_i = (x_i, y_i)$ represents the coordinates of the trajectory sampled at time t_i. Let $\tau(t_i) = z_i$ denote the location z_i of trajectory τ at time t_i. Additionally, let $\tau[t_i, t_j]$ with $1 \leq i < j \leq m$ indicate the *sub-trajectory* of τ between timestamps t_i and t_j. Formally, this means $\tau[t_i, t_j] = \{(z_i, t_i), (z_{i+1}, t_{i+1}), \ldots, (z_j, t_j)\}$. We also define the set of edges of τ as $\overline{\tau} = \{\tau[t_i, t_{i+1}] | 0 \leq i < m\}$. We denote the set of input trajectories as $\mathcal{T}$. In many applications, since we are interested in the movement of a group, all input trajectories in the group are sampled over a similar time interval. In this chapter we assume that, for an input of multiple trajectories, all trajectories in the input are sampled at the same timestamps, but not necessarily uniformly. This assumption is essential for the following definition of a segment.

A *segment* S is a set of sub-trajectories that have the same start- and end time. That is, a segment S consisting of h sub-trajectories starting at time t_i and ending at time t_j is defined as $S = \{\tau_1[i, j], \ldots, \tau_h[i, j]\}$ where $\tau_1, \ldots, \tau_h \in \mathcal{T}$. Each segment S has an associated segment parameter $x(S)$ that describes the movement characteristics of the (sub-)trajectories in the segment S. The utility of this parameter will be discussed in the next section. A complete *segmentation* $\mathscr{S}$ of an input set of trajectories $\mathcal{T}$ is then defined as a set of k segments $\mathscr{S} = \{S_1, \ldots, S_k\}$, such that for all trajectories $\tau \in \mathcal{T}$ each edge $e \in \overline{\tau}$ is in exactly one segment $S_i \in \mathscr{S}$.

Finally, we can define a *class* C as a set of segments from $\mathscr{S}$, such that $C \subseteq \mathscr{S}$. Each class also has an associated class parameter $x(C)$ that describes the movement characteristics of the (sub-)trajectories in the segments of class C. A complete *classification* $\mathscr{C}$ of a segmentation $\mathscr{S}$ is then defined as a partition $\mathscr{C} = \{C_1, \ldots, C_\ell\}$ of set $\mathscr{S}$. As such, each segment $S_i \in \mathscr{S}$ appears in exactly one class $C_i \in \mathscr{C}$.

Given a segmentation $\mathscr{S}$ of a set of trajectories $\mathcal{T}$, the goal of this part is to find an optimal classification $\mathscr{C}$ of $\mathscr{S}$. We do so by investigating whether we can apply the discrete classification algorithm by Alewijnse et al. directly to the segmentation of a group [2].

4.2 Background

The segmentation and classification approach by Alewijnse et al. [2] uses a model-based approach, which fits a parameterized movement model to the data. To be able to use segmentation and classification in statistical analysis of group movement, we need to be able to evaluate the quality of these segmentations and classifications. That way, we can be sure that the proposed models actually fit the data well before they are used to make inferences about the movements. In this section, since many definitions hold for both segmentations and classifications, we use the term *partition* when we refer to either a segmentation or a classification. Additionally, when we refer to a segmentation or a classification as a partition, we will use the term *part* to refer to either a segment in the segmentation or a class in the classification. The methods used to evaluate the quality of partitions are discussed in this section; first we discuss how we can utilize the likelihood functions of the fitted models to extract a quality measure, after which we will discuss how to use these quality measures to obtain meaningful partitions.

4.2.1 Log-likelihoods

A number of models can be used to describe the movement data, such as the Brownian bridge movement model (BBMM) and its variants [19], or behavioral change point analysis (BCPA) [15]. In these models, which are commonly used in ecology [2], a single movement parameter indicates the characteristics of the movement, yet the characteristics described by the input parameter differs per movement model. In this chapter, we will use BCPA to fit the model, since it is well-suited to find phases in movement data.

In these models, each edge $e \in \overline{\tau}$ has an associated log-likelihood function $LL_e(x)$ which describes how well the model describes the movement in e using parameter x. We will later discuss how we can compute these log-likelihood values. Using these log-likelihood values, we can describe the log-likelihood of (sub-)trajectories τ, segments S, and partitions $\mathscr{S}$ or $\mathscr{C}$ as the sum of the log-likelihoods of their parts as follows:

$$LL_\tau(x) = \sum_{e \in \overline{\tau}} LL_e(x)$$

$$LL_S(x) = \sum_{\tau \in S} LL_\tau(x)$$

$$LL_{\mathscr{S}} = \sum_{S \in \mathscr{S}} LL_S(x(S))$$

$$LL_{\mathscr{C}} = \sum_{C \in \mathscr{C}} \sum_{S \in C} LL_S(x(C))$$

To find an optimal partition of the input data, we now want to maximize the log-likelihood of the complete partition. However, if we exclusively use the log-likelihood as a quality measure, an optimal partition will result in each edge getting assigned their own segment/class with the most fitting parameter value.

Information Criterion

To limit the number of parts in a partition, an *Information Criterion* (IC) is used [2,15]. The goal of an IC is to assign each partition a quality value that is dependent mostly on the log-likelihood of the fitted movement parameters, but also on the complexity of the partition. This way, an IC can be used to implicitly limit the number of parts that can exist within an optimal partition. Here, we exclusively consider ICs of the form

$$IC(\mathscr{P}) = -2 \cdot LL_{\mathscr{P}}(x) + |\mathscr{P}| \cdot p$$

where $\mathscr{P}$ is the partition we want to compute the IC for, $|\mathscr{P}|$ denotes the number of parts in the partition, and $p \geq 0$ is a variable penalty value. Two common values for penalty value p are $p = ln(k)$ (where k is the number of segments) in the Bayesian IC (BIC) [32] and $p = 2$ in the Akaike IC (AIC) [1]. In order to ensure a reasonable number of classes, we will attempt to minimize the value of the IC instead of trying to maximize the absolute likelihood of the partition. Thus, we define an optimal partition to be the partition that minimizes the value of the IC.

4.2.2 Behavioral Change Point Analysis

Now that we know how to use the value of the log-likelihood function to obtain an optimal partition, we will discuss how these log-likelihood functions are actually obtained. We will limit this discussion to the modelswe use in the proof-of-concept implementation, which are the models described in Behavioral Change Point Analysis (BCPA). We use these models since they are specifically tailored to identify phases in movement behavior [15].

In the BCPA models, we require some type of descriptive parameter that is able to quantify certain characteristics of the movement. As examples, we use the persistence velocity and the turning velocity as defined in the original BCPA description by Gurarie et al. [15]. To this end, let the absolute positions z and the absolute orientation ϕ of the raw data be processed into the estimated speed V and turning angles Ψ of the data, obtained via

$$V(t_i) = ||z_i - zi_1||/(t_i - ti - 1),$$

$$\Psi(t_i) = \phi_i - \phi_{i-1}.$$

These speed and turning angle estimates are then transformed into two variables $V_p(t_i)$ (persistence velocity) and $V_t(t_i)$ (turning velocity) that describe the desired movement characteristics; $V_p(t_i)$ describes the tendency of the movement to persist in a given direction, and $V_t(t_i)$ describes the tendency of the movement to head in a direction perpendicular to the previous at any given time interval. These features are calculated with

$$V_p(t_i) = V(t_i)\cos(\Psi(t_i)),$$

$$V_t(t_i) = V(t_i)\sin(\Psi(t_i)).$$

The assumption is made that the movement characteristics V_p and V_t can be described by a Gaussian process with normal distribution $w_i =\sim \mathcal{N}(\mu, \sigma^2)$. BCPA also heavily relies on the assumption that there exists some correlation between subsequent data points that can be described by auto-correlation coefficient ρ^{τ_i} such that observation w_i can be described as

$$w_i = \mu + \rho^{T_i}(w_{i-1} - \mu) + \epsilon_i.$$

Here, $T_i = t_i - t_{i-1}$ and ϵ_i is an error term determined by the variance σ^2 of the data as well as auto-correlation coefficient ρ^{T_i}. The error term ϵ_i can be shown to have a mean of 0 and variance $\sigma^2(1 - \rho^{2T_i})$ [15].

We can approximate the mean and standard deviation of the modeled Gaussian process that describes V_p and V_t using the actual mean μ and the standard deviation σ of these variables. Since the model has a Gaussian error structure (with error term ϵ_i), we can compute the likelihood of any auto-correlation coefficient between V_p or V_t of w_i and w_{i-1} (for $1 \leq i \leq m$) according to the probability density function of conditional distribution $w_i|w_{i-1}$ (where w_i models the variable associated to trajectory sample z_i):

$$L_{z_i} = \frac{1}{\sqrt{2\pi\sigma^2(1-\rho^{2T_i})}} \cdot \exp\left[\frac{-(w_i - \rho^{T_i}(w_{i-1}-\mu)-\mu)^2}{2\sigma^2(1-\rho^{2T_i})}\right].$$

Note that this likelihood function deviates from the function given by Guraire at al. in the initial description of the approach [15]. However, Guraire et al. also provide an implementation of their algorithms in a later paper [16]. The provided implementation deviates from the original paper and features this function, which is the function that results from reproducing the described steps above.

Since the auto-correlation coefficient must be between 0 and 1, the likelihood of each observation can be maximized over $0 < \rho < 1$ to find the auto-correlation most likely to fit the data at each observation. We define the log-likelihood as $LL_{z_i}(\rho) = \log(L_{z_i}(\rho))$. Since each line segment in a set of (sub-)trajectories is independent, we can take the sum of the log-likelihoods of all line segments in the set to get a total log-likelihood $LL_S(\rho)$ for the auto-correlation of V_p or V_t of the entire set.

4.2.3 Group segmentation

The group segmentation work by Mols [25] describes three separate heuristic group segmentation methods. Of these three methods, we use the *two-step approach*, which first uses the grouping structure to group sub-trajectories prior to segmenting them. First we discuss how the (sub-)trajectories are grouped in a first pass using at method similar to the trajectory grouping structure [6], after which we discuss how these groups are segmented using model-based segmentation algorithm by Alewijne et al. [2].

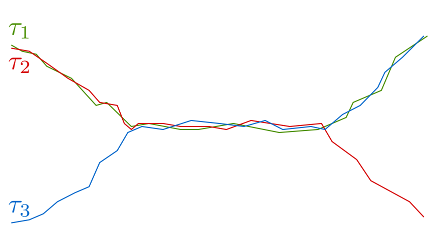
Grouping structure

The trajectory grouping structure is a model that attempts to capture, given an input set of trajectories, when and which subsets of entities travel together for a sufficiently long time [6]. The group segmentation work slightly alters this definition, using the original notion of "components" in the input data to find which entities travel together for some amount of time [25].

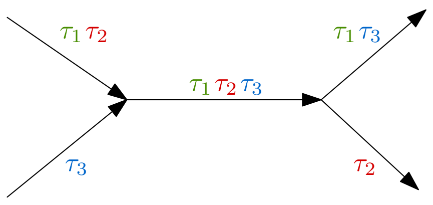
Given an input set of trajectories $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, let two trajectories τ_i and τ_j be *directly connected* at time t if the distance between $\tau_i(t)$ and $\tau_j(t)$ is smaller than some parameter ε. Two trajectories are *ε-connected* at time t if there exists a sequence of trajectories $\tau_i = \tau^0, \ldots, \tau^k = \tau_j$ such that for all $0 \leq i \leq k-1$, τ^i and τ^{i+1} are directly connected. A subset $R \subseteq \mathcal{T}$ of trajectories is *ε-connected* at time t if all trajectories in R are pairwise ε-connected at time t. We call such an ε-connected set R a *component* if and only if it is a maximal ε-connected set. It is each to see that at any time t the set of all components $\mathscr{D}(t)$ forms a partition of the input trajectory set $\mathcal{T}$.

In its original description, the trajectory grouping structure further describes the notion of groups using the components $\mathscr{D}(t)$, along with two additional input parameters: a temporal parameter δ and a size parameter m that describe how long a component must exist and how many trajectories it must contain for it to be considered a group. In the group segmentation work, these two parameters are set to 0 and 1, respectively, such that the set of components $\mathscr{D}(t)$ directly describe the grouping structure as they evolve through time [25]. To this end, in our application we consider a *group* G to be described by a set of trajectories $T_G \subseteq \mathcal{T}$ and a time interval $[t_s(G), t_e(G)]$, where $t_s(G)$ and $t_e(G)$ are the start and end times of group G, respectively. We aim to find a set of groups $\mathscr{G}$ such that, at all times t, the groups $G \in \mathscr{G}$ for which $t_s(G) \leq t \leq t_e(G)$ form a partition of the trajectories in $\mathcal{T}$.

In the group segmentation work, this set of groups $\mathscr{G}$ is constructed by tracking the set of components $\mathscr{D}(t)$ through time. Each component D describes a group G consisting of the trajectories in D, where the start time $t_s(G)$ is the time at which component D first appears in $\mathscr{D}(t_s(G))$, and the end time $t_e(G)$ is the time at which the component D disappears from $\mathscr{D}(t_e(G))$. Note that if



(a) An input set of three input trajectories.



(b) The trajectories from (a) grouped into groups.

Figure 4.1: An example of how trajectories are grouped in the group segmentation work [25]. The directed edges in (b) represent the groups, labeled with the contained trajectories.

a component D appears in $\mathscr{D}(t)$ in more than one disjoint time intervals, it describes a separate group for each disjoint time interval it appears in. An example of a partition of $\mathcal{T}$ using groups in this manner can be found in Figure 4.1.

Model-based segmentation

The group segmentation work describes an approach to segment the groups found in the previous section using the model-based approach for single trajectories [2,25]. The approach segments each group $G \in \mathscr{G}$ separately and outputs the union of these segments as the segmentation $\mathscr{S}$. Since $\mathscr{G}$ describes a partition of $\mathcal{T}$, this results in a complete segmentation. Let $\mathcal{S}_G$ be a complete segmentation of group G. Formally, we then get a complete segmentation $\mathscr{S}$ of $\mathcal{T}$ using

$$\mathscr{S} = \bigcup_{G \in \mathscr{G}} \mathcal{S}_G$$

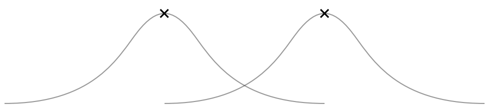
Note that the information criterion (IC) for a segmentation is simply computed by taking the sum of its parts. Therefore, to find optimal segmentation $\mathscr{S}$ all that is left to do is to compute an optimal segmentation $\mathcal{S}_G$ for each group G. For group G, the optimal segmentation is computed using the discrete segmentation algorithm [2], slightly adapted to work with groups. Given a group G consisting of trajectories $T_G \subseteq \mathcal{T}$ in time interval $[t_s(G), t_e(G)]$, a set of candidate parameters $\{x_1, \ldots, x_m\}$ and a penalty factor p, we aim to compute a segmentation $\mathcal{S}_G$ of G that assigns each segment $S \in S_G$ a parameter value $x(S)$ such that we minimize some IC.

Since we assume all trajectories to be sampled at the same timestamps, the discrete segmentation algorithm by Alewijnse et al. can be directly applied to the group G [2]. The only slight adjustment to the functioning of the algorithm is that, as opposed to using the log-likelihood of a single trajectory to find the value of the IC for each segment, we take the sum of log-likelihoods of all sub-trajectories in the segment. Once an optimal segmentation has been found for each group $G \in \mathscr{G}$, the union of these optimal segmentations describes the complete segmentation $\mathscr{S}$ of $\mathcal{T}$.

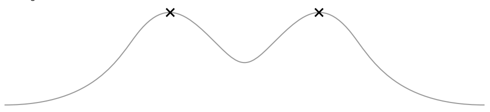
4.3 Classifying segments

We now consider the problem of classifying a set of group segments. That is, given a set of trajectories $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, a corresponding segmentation $\mathcal{S} = \{S_1, \ldots, S_k\}$ and a penalty factor p, we aim to compute a classification $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ of $\mathcal{S}$ (that is, a partition of $\mathcal{S}$) and assign each class C a parameter value $x(C)$ such that we minimize some information criterion.

The approach described here for group classification is similar by Alewijnse et al. to classify single trajectory segments [2]. The group classification algorithm works in the same manner as the discrete classification algorithm proposed by Alewijnse et al.; it uses a dynamic programming approach to classify a set of trajectory segments, represented by their log-likelihood functions as input, using a discrete set of candidate parameters. In order to investigate whether we can directly apply the approach to groups of trajectories as well, we input a set of log-likelihood functions representing segments as defined in the previous section.



(a) Two bitonic curves, with unique maxima indicated by $\times$.



(b) The sum of the two bitonic curves in (a), which has two maxima (indicated with $\times$), and is thus not bitonic.

Figure 4.2: The sum of the two bitonic curves in (a) is not bitonic, as it has two maxima.

The algorithm described by Alewijnse et al. requires the critical assumption that the input log-likelihood functions are *bitonic* [2]. That is, each log-likelihood function LL_i has a single maximum M_i, are increasing before the maxmimum and decreasing after the maximum. Also, the algorithm requires that the log-likelihood functions are provided in order of increasing M_i. In practice log-likelihood functions for realistic input trajectories are often bitonic, as trajectories with non-bitonic log-likelihood functions require large variations in sampling rate and speed. Even if the log-likelihood functions are not strictly bitonic, parameter values of local maxima are often quite close to each other, meaning that the algorithm will still produce results that are close to optimal.

This bitonicity assumption presents an issue when the algorithm is applied to groups of trajectories, since the log-likelihood function of a segment is defined as the sum of the log-likelihoods of its parts (thus; sub-trajectories). As argued by Alewijnse et al., in a realistic setting the log-likelihood functions of these sub-trajectories are likely to be bitonic [2]. However, the sum of two or more bitonic functions is not necessarily bitonic; see Figure 4.2. This means that a log-likelihood function LL_i of a segment does not necessarily have a single maximum M_i, and the input functions can therefore not be easily sorted by the value of their maxima.

In order to resolve this, observe that the group segmentation approach places sub-trajectories into segments according to an information criterion [25]. This means that the log-likelihood functions of the trajectories that make up a segment are very likely to have similar maxima that lie within some small range, meaning that it is very likely that the log-likelihood function of the segment reaches its maximum somewhere within that range. In order to approximate a suitable value to sort the input segments, we take the median of the maxima of the log-likelihood functions corresponding to the sub-trajectories in each segment. Since the maxima of the log-likelihood functions of the sub-trajectories in a segment are likely to lie very close together, this median value gives us a good approximation of a parameter value around which the maxima are centered. As such, the algorithm is expected to yield near-optimal results.

Algorithm 1 DISCRETECLASSIFICATION($(LL_1, \ldots, LL_k), (x_1, \ldots, x_m)$) [2]

```
1: $Opt_0$ = An array of length $m$ with all elements set to NIL
2: $\mathscr{C}$ = An arbitrary (complete) classification of $L_1, \ldots, L_k$
3: for i = 1 to m do
4:     for j = 0 to i=1 do
5:         $O_j = Opt_j$
6:         for $LL_j \in \mathscr{L}_{j,i}$ do
7:             $O_j[l] = \arg\max_{x \in \{x_j, x_i\}}(LL_l(x))$
8:         $Opt_i = \arg\min_{\{O_j | 0 \le j < i\}} IC_i(O_j)$
9:
```

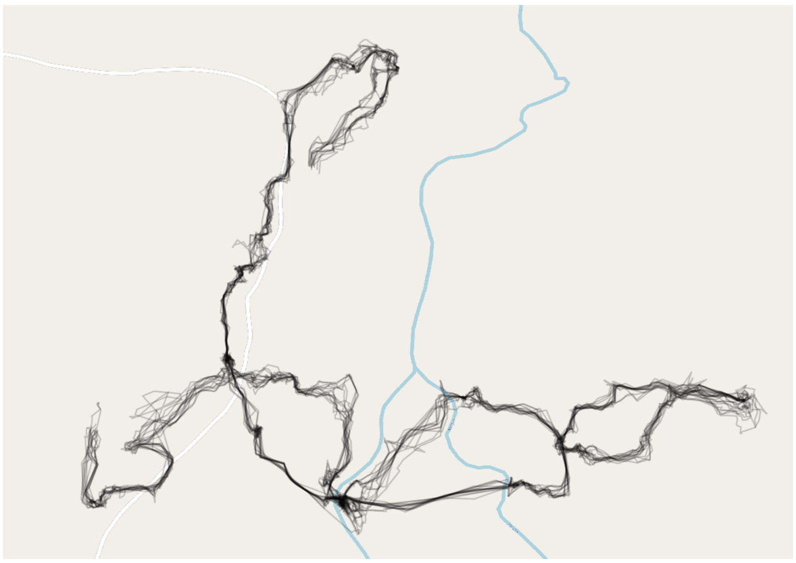


Figure 4.3: A full overview of the used subset from the baboon data set. The data describes a set of 15 baboons traveling through a wildlife reserve over a period of roughly 54 hours.

In order to classify a set of given input segments $\mathcal{S} = \{S_1, \ldots, S_k\}$ using a discrete set of parameters, we assume the problem instance is given by $(LL_1, \ldots, LL_k), (x_1, \ldots, x_m)$ where LL_i with $1 \leq i \leq k$ represents the log-likelihood function of segment S_i. Here, we assume that both $(LL_1, \ldots, LL_k)$ and $(x_1, \ldots, x_m)$ are sorted in ascending order. If not, they can simply be sorted in a preprocessing step. If the input is given in this format, then the classification algorithm by Alewijnse et al. [2], given in Algorithm 1, can be directly applied to group classification of segments.

4.3.1 Application to real-life data

In order to demonstrate the utility of this approach, a proof-of-concept implementation was used to obtain a classification for a segmented group of wildlife movement data. The data set that is used describes the trajectories of 25 baboons in a troop moving through a wildlife reserve in Laikipia, Kenya[1]. The locations of the baboons are sampled every second, over a time period of multiple days using a high-resolution global positioning system.

A subset of this data was recently used in experiments for group segmentation [25]. Since our approach extends the group segmentation approach, we will use the same subset of this data to demonstrate how the classification of these segments can help classify different movement phases. As such, 15 out of the 25 baboons in the original data set were selected by hand based on the consistency of location reports. These 15 trajectories were further sampled to obtain 15 sub-trajectories of 1000 observations each in such a way that the sub-trajectories describe a varied set of group movement phases over a period of approximately 54 hours. A full overview of these trajectories can be found in Figure 4.3.

The trajectories are first segmented using the two-step group segmentation method described in Section 4.2.3 using the auto-correlation based models from BCPA as described in Section 4.2.2, once using the persistence velocity V_p and once using turning velocity V_t as the describing parameters for the data. Afterwards, we classify the resulting segments of these output segmentations using the same models with the persistence- and turning velocity according to the approach described in Section 4.3.

In the visualizations of the classifications, we draw the movement trajectories of the input data

[1]Source: https://www.datarepository.movebank.org/handle/10255/move.405, obtained in May 2021.

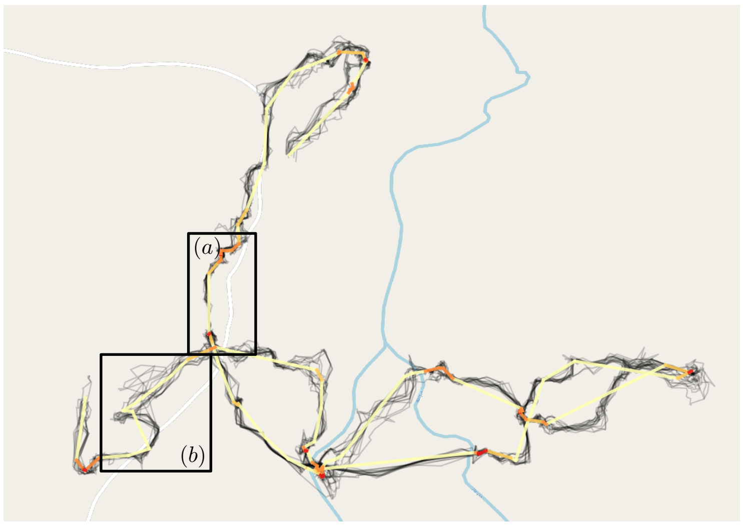


Figure 4.4: A full overview of the used subset from the baboon data set after classification with BCPA using parameter V_p. Highlighted sections can be found in Figure 4.5. Darker colors indicate higher auto-correlation values for the class the segment was assigned to.

as transparent black lines. Segments that describe a set of sub-trajectories are visualized as thick lines between the average start- and the average end positions of the sub-trajectories it describes. The color of these lines indicate the classes in which the segments are placed. The color palette we use ranges from light yellow, which indicates a very low auto-correlation coefficient, to dark red, which indicates a very high auto-correlation coefficient. Since the auto-correlation coefficient lies within range $[0, 1]$, additional colors are simply obtained by linearly interpolating between light yellow and dark red.

The results from the group segmentation and classification using the auto-correlation model with persistence velocity V_p can be seen in Figures 4.4 and 4.5, and the results using the auto-correlation model with turning velocity V_t can be seen in Figure 4.6. For the creation of the group segmentations, as well as for the group classification of these segmentations, the Bayesian information criterion with a penalty factor $p = 2$ was used. The distance parameter ε used to find the grouping structure prior to segmenting was set to $\varepsilon = 300$. The group classification algorithm used a discrete set of values, uniformly sampled between 0 and 1 with step size 0.01, as candidate parameter values. The group classification using the persistence velocity V_p produced a set of four classes, whereas the use of the turning velocity V_t produces five classes. For both of these sets of classes, the auto-correlation value assigned to the classes ranged between 0.01 and 0.95.

In order to analyze the output of the classification, it is worthwhile to more closely examine certain sections of the movement. To this end, consider the highlighted sections in Figure 4.4, of which close-ups can be found in Figure 4.5. The sub-trajectories displayed in Figure 4.5a describe the movement of the troop alongside a road over a time-span of approximately four hours. Intuitively, we can distinguish three phases during these four hours. First, the troop remains near a single location for 1.5 hours; then, the troop moves approximately 400 meters northwards in 0.5 hours; lastly, the troop spends approximately 2 hours moving further north slowly, during which individual baboons frequently stray somewhat far from the troop.

The group segmentation algorithm segmented this first phase in which the baboons remain near a single location as a single, short segment, which was classified as a segment in the class with the highest auto-correlation of 0.95. The fast, directed movement northward was classified in the class with the very lowest auto-correlation coefficient of 0.01. The last part of the movement that is slow-moving and exhibits more baboons straying from the group is segmented into multiple different segments, most of which are classified as classes with auto-correlation coefficients between 0.4 and 0.8.

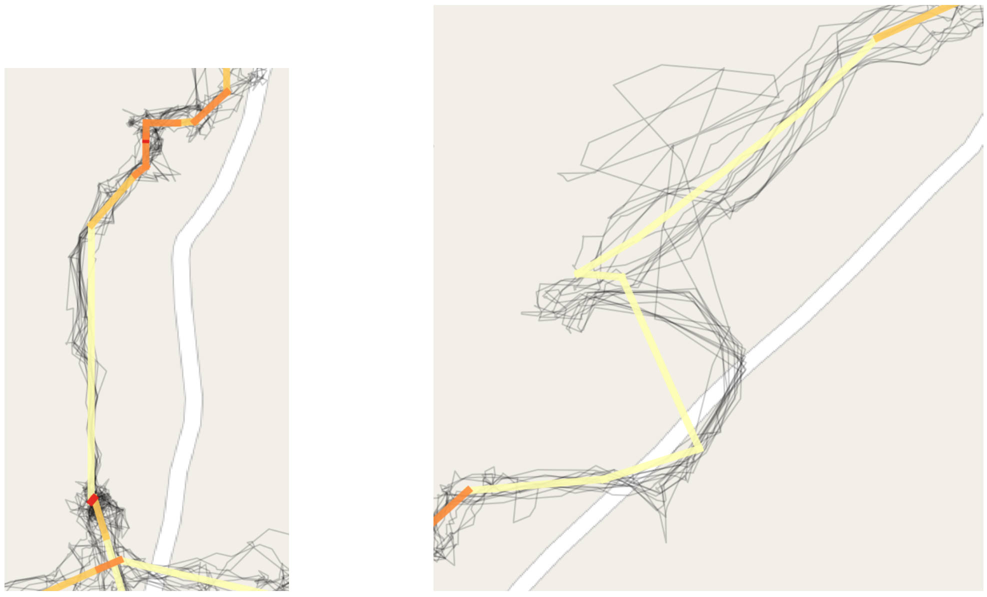


Figure 4.5: The two highlighted sections from Figure 4.4 after classification with BCPA using parameter V_p.

Most segments in the input data seem to be classified along a similar pattern. As is clear from Figure 4.4, segments that describe troop movement that remains near a single location for longer periods of time get classified with relatively high auto-correlation coefficients, whereas long stretches of directed movement get assigned a class with lower auto-correlation coefficients. It is important to note, however, that the assigned classes seem mostly dependent on the velocity of the movement rather than the directedness of the movement. Consider the sub-trajectories in Figure 4.5b. There, a significant portion of the sub-trajectories is not directly moving from one point to another, but taking rather large detours. However, these segments are classified in the same class as other, more directed movement segments, which is likely due to the velocity of the sub-trajectories in this segment being relatively high.

The group segmentation and classification using the turning velocity V_t can be seen in Figure 4.6. While the resulting classification using V_t has an additional class when compared to the classification that resulted from BCPA using V_p, the general classification pattern is similar: movement phases that exhibit compact, slower and less directed movement can mostly be differentiated from more long distance and directed movement phases, with the prior being assigned to classes with a higher auto-correlation and the latter being assigned to classes with lower auto-correlation coefficients.

As we can see from Figures 4.4 and 4.6, the proof-of-concept implementation of the group classification approach is generally able to distinguish fast and directed phases of movement in this data set from slower and compact movement. As such, these first experiments are quite promising. An important observation, however, is that the detection of the phases seems to be influenced significantly more by the velocity of the movement than by the turning angle. This is likely due to the fact that the values of the persistence velocity and the turning velocity are computed by multiplying the velocity of the movement with the sine and the cosine of the turning angle of the movement, respectively. While the value of the sine and the cosine of the turning angle can at most differ by a value of 2, the velocity of the movement in one segment can be orders of magnitude higher than the velocity in another segment. This means that the value of the persistence velocity and the turning velocity are more dependent on the velocity of the movement than they are dependent on the turning angle. This observation would justify why segments are

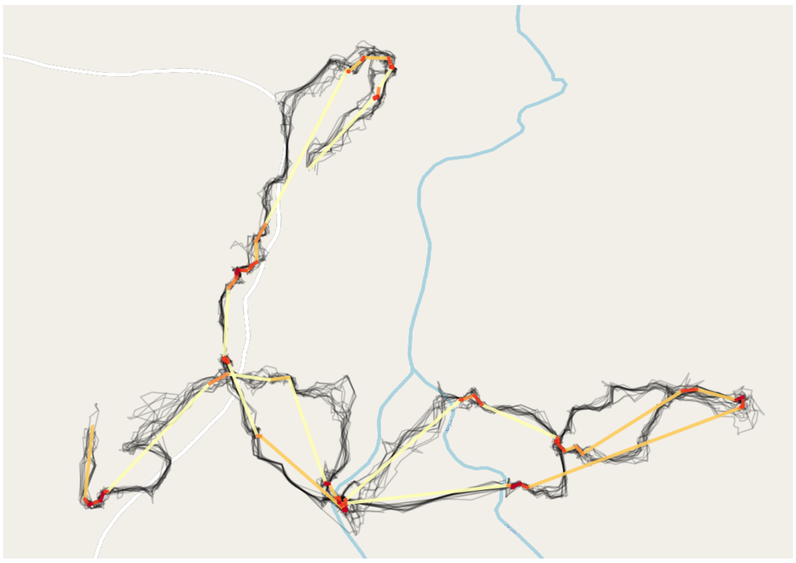


Figure 4.6: A full overview of the used subset from the baboon data set after classification with BCPA using parameter V_t. Darker colors indicate higher auto-correlation values for the class the segment was assigned to.

generally classified according to the average velocity of their sub-trajectories. Therefore, it seems that this bias toward the velocity is mainly an artifact of the persistence velocity and the turning velocity. In order to verify this, more experiments would be necessary in which different movement parameters are used in place of the persistence- and the turning velocity.

Chapter 5

Discussion

The analysis of movement data has long been an important tasks in a number of research areas. In such movement data, related sets of entities can form coherent groups in the data. In such groups, a number of entities remain close together for an extended period of time. Analyzing the shape of such groups allows us to distinguish them from each other, and allows us to make inferences about the underlying movement patterns. In this thesis, we characterize the shape of a group using its density, and we propose methods that allow for the identification of the dense and sparse areas of groups of one-dimensional entities. We achieve this by maintaining the extrema of a terrain T_P that represents an estimation of the probability density of a set of time-varying data points in one dimension.

In Chapter 2, we discussed the tools that would ultimately be useful during the construction of the data structure. We described how to use a concept from statistics called "kernel density estimation" to estimate the probability density distribution of a set of points, and describe how we will use this technique to construct a terrain T_P of which the extrema estimate the locations of dense and sparse areas in the data. We discussed a class of data structure called kinetic data structures, which allow proofs of correctness for attributes of geometric systems to be animated and maintained through time.

In Chapter 3 we utilized the tools described in the previous chapter to construct a kinetic data structure that is able to maintain the extrema of the time-varying terrain T_P through time. We investigated how points in P can be analyzed to find extrema in the terrain without constructing the kernel density estimation function in its entirety. After describing how to find these extrema in the static setting, we investigated when and how these extrema would change if the underlying point set was time-varying. We then described how to use these observations to build the kinetic data structure to maintain these extrema. We analysed the quality of the kinetic data structure using the common quality measures for kinetic data structures, and found it to be compact, local, responsive and weakly efficient.

While the KDS described above is theoretically efficient, in the worst case it may require a number of events that is quadratic in the input size. Therefore, we also investigated whether an approximation technique for kernel density estimation, called coresets, could be used to improve the efficiency of the kinetic data structure. We briefly described a small number of different options to generate coresets, and describe how to use the sort-selection approach to find a coreset with a fixed and guaranteed error bound. After discussing how to do so in a static setting, we described how to maintain the generated coreset in an efficient manner in the kinetic setting. We evaluated how this maintained coreset could be applied to our kinetic data structure, and what the impact of the coreset would be on the quality of the kinetic data structure. Using coresets, we were able to reduce the number of events processed by the KDS from $O(n^2)$ to $O(n^{\frac{4}{3}} \cdot \frac{1}{\varepsilon})$. In doing so, we are able to show that the most relevant extrema of the terrain are maintained after the application of the coreset. We measure the relevance of an extremum by its persistence: if the coreset describes an ε approximation of the input point set, we were able to show that any extremum with persistence equal to or higher than 2ε is maintained. This means that any low-persistence are not guaranteed to also appear in the estimated density function of the coreset. This is not necessarily an issue, however, since it may actually be desirable to filter out extrema of low relevance in some applications (to filter out noise, for example).

Using the approximation to improve the efficiency of the KDS, however, has some drawbacks. While the efficiency of the KDS is improved, the locality and the responsiveness are negatively impacted. While using the coreset, a single point may appear in at most $O(\varepsilon n)$ certificates, up from $O(1)$ in the KDS without the coreset. As a result, the responsiveness of the KDS is raised to $O(\varepsilon n \log n)$ from $O(\log n)$. Using a coreset to both increase the efficiency and reduce the number of irrelevant information that is stored is a promising method to improve the KDS. There are currently major drawback to this approach. Therefore, a promising direction for future work could be to try to reduce these drawbacks. This could, for example, be achieved by attempting different methods for generating the coreset. It would also be interesting to see whether the KDS could be made more efficient and responsive by, for example, maintaining the 2ε-approximation version of the sort-selection coreset. While this makes the coreset slightly less representative of the original point set, perhaps relaxing the constraints on the points in the coreset allows us to maintain it more efficiently. Also, as briefly discussed in Section 3.4, a promising direction for maintaining the coreset more efficiently could be to use auxiliary kinetic data structures to track each point in Q separately. This seems theoretically feasible, but more investigation is necessary to analyze what the implications for the quality measures would be.

As briefly discussed in Chapter 2, tracking topological properties beyond the extrema of terrain T_P also remains an interesting research direction. Although the methods described in Chapter 3 would not suffice to track features like the persistence pairings of the extrema, perhaps the described KDS can be extended to allow for the maintenance of these features as well. A major obstacle here is that tracking and comparing the heights of vertices in the terrain is very expensive, which is why our approach uses the slope of the terrain rather than the height of its vertices. Another way to extend the work described in Chapter 3 could be to investigate how these approaches extend to different types of kernels. This thesis works with the linear kernel because it allows us to maintain the extrema strictly using the slope of the terrain, which must then be an integer multiple of a constant. Using, for example, the Gaussian kernel could complicate matters somewhat, but it is not unlikely that the general structure of the kinetic data structure described in Chapter 3 could also be used in a setting using different kernels.

Furthermore, a natural next step is to investigate how the techniques described in Chapter 3 generalize to higher dimensions. In particular the two dimensional setting is interesting, as it more closely resembles the likely applications. This direction of research is likely to be more involved than the prior two discussed above, as many of the observations made about the one dimensional case either no longer hold or are significantly more complex to find and proof in higher dimensions. For example, in two dimensions linear kernel boundaries consist of a two dimensional circle centered around the data point rather than two separate points. This makes our definition of minima in one dimension difficult to generalize. The general structure of the kinetic data structure and the times at which events take place, however, may yet generalize to higher dimensions. In any case, the observations made in this thesis provide insight into the one-dimensional case, which provides a foundation from which to investigate changes in density in settings with more dimensions.

In addition to the work on group density, in Chapter 4 we investigated how different movement phases in the movement of a group can be distinguished. Specifically, we demonstrated that the single-trajectory classification algorithm by Alewijnse et al. can also be applied to groups of trajectories. Given a set of trajectories, we find groups in the data and split them into segments using the two-step segmentation approach by Mols [25]. Given these group segments, the single-trajectory classification algorithm by Alewijnse et al. is directly applied to these group segments to obtain a group classification. The classification algorithm attempts to fit a parameterized model to the movement data. The model we use is based on a Gaussian process. For the model parameter, we use the auto-correlation of the persistence velocity and the turning velocity. The persistence velocity describes the tendency of the movement to persist in a given direction, whereas the turning velocity describes the tendency of the movement to change its movement direction. We demonstrated the utility of the approach by using a proof-of-concept implementation to classify different phases in a group movement data set of a troop of baboons moving through a wildlife reserve. The results of the algorithm using the persistence velocity and the turning velocity were quite similar. The results of the initial experiments look promising, and provide intuitive classifications of the group of trajectories which are able to distinguish fast and directed phases of movement from slower and more compact movement.

We observed that, in these experimental results, the detection of the movement phases seems to be influenced significantly more by the velocity of the movement than by the movement direction. We hypothesize that this is mainly an artifact of the persistence velocity and the turning velocity rather than an issue with the approach itself. To verify this, it would indeed be useful to experiment with using slightly different movement parameters in the fitted model. For example, a way to produce a classification that is more sensitive to changes in the turning angle of the sub-trajectories could be to normalize the velocity of the sub-trajectories prior to using them to compute the persistence- and turning velocity. Alternatively, a completely different set of descriptive parameters could be used. It would be interesting, for example, to use a quantified estimate of the density around each entity as descriptive parameter in the group classification model. Such an estimate could be quantified by the sum of the distance between the k nearest neighbors of each entity.

Bibliography

[1] Hirotugu Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[2] Sander P.A. Alewijnse, Kevin Buchin, Maike Buchin, Stef Sijben, and Michel A. Westenberg. Model-based segmentation and classification of trajectories. *Algorithmica*, 80(8):2422–2452, 2018.

[3] Mohammad Ali Abam and Mark de Berg. Kinetic sorting and kinetic convex hulls. *Computational Geometry*, 37(1):16–26, 2007.

[4] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *J. Algorithms*, 31(1):1–28, 1999.

[5] Jiang Bian, Dayong Tian, Yuanyan Tang, and Dacheng Tao. Trajectory data classification: A review. *ACM Transactions on Intelligent System Technology*, 10(4), 2019.

[6] Kevin Buchin, Maike Buchin, Marc van Kreveld, Bettina Speckmann, and Frank Staals. Trajectory grouping structure. *Journal of Computational Geometry*, 6(1):75–98, 2015.

[7] 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.

[8] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry - DCG*, 37:263–271, 2005.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT press, 3rd edition, 2009.

[10] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer, 3rd edition, 1997.

[11] Tamal K. Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19(3):373–382, 1998.

[12] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction.* American Mathematical Society, 2010.

[13] Jasper A.J. Eikelboom. *Sentinel animals: Enriching artificial intelligence with wildlife ecology to guard rhinos.* PhD thesis, Wageningen University, 2021.

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

[15] Eliezer Gurarie, Russel D. Andrews, and Kristin L. Laidre. A novel method for identifying behavioural changes in animal movement data. *Ecology letters*, 12(5):395–408, 2009.

[16] Eliezer Gurarie, Christen Fleming, William Fagan, Kristin Laidre, Jesús Hernández-Pliego, and Otso Ovaskainen. Correlated velocity models as a fundamental unit of animal movement: Synthesis and applications. *Movement Ecology*, 5(13), 2017.

[17] Attila Gyulassy, Peer-Timo Bremer, Bernd Hamann, and Valerio Pascucci. A practical approach to morse-smale complex computation: Scalability and generality. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1619–1626, 2008.

[18] Attila Gyulassy, Vijay Natarajan, Valerio Pascucci, and Bernd Hamann. Efficient computation of Morse-Smale complexes for three-dimensional scalar functions. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1440–1447, 2007.

[19] Jon S. Horne, Edward O. Garton, Stephen M. Krone, and Jesse S. Lewis. Analyzing animal movements using brownian bridges. *Ecology*, 88(9):2354–2363, 2007.

[20] Yan Huang, Cai Chen, and Pinliang Dong. Modeling herds and their evolvements from trajectory data. *Geographic Information Science*, pages 90–105, 2008.

[21] San-Yih Hwang, Ying-Han Liu, Jeng-Kuen Chiu, and Ee-Peng Lim. Mining mobile group patterns: A trajectory-based approach. *Advances in Knowledge Discovery and Data Mining*, pages 713–718, 2005.

[22] Amílcar S. Júnior, Bruno N. Moreno, Valéria C. Times, Stan Matwin, and Lucídio A.F. Cabral. GRASP-UTS: an algorithm for unsupervised trajectory segmentation. *International Journal of Geographical Information Science*, 29(1):46–68, 2015.

[23] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. *Advances in Spatial and Temporal Databases*, pages 364–381, 2005.

[24] Luis A. Leiva and Enrique Vidal. Warped k-means: An algorithm to cluster sequentially-distributed data. *Information Sciences*, 237:196–210, 2013.

[25] Jorik Mols. Model-based segmentation of collective movement. Master's thesis, TU Eindhoven, 2021.

[26] Anders Nilsson. Predator behaviour and prey density: evaluating density-dependent intraspecific interactions on predator functional responses. *Journal of Animal Ecology*, 70(1):14–19, 2001.

[27] Tim Ophelders, Willem Sonke, Bettina Speckmann, and Kevin Verbeek. A KDS for discrete Morse-Smale complexes. *Computational Geometry: Young Researchers Forum*, pages 3:1–3:2, 2018.

[28] Emanuel Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076, 1962.

[29] Jeff M. Phillips. ε-samples for kernels. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1622–1632. SIAM, 2013.

[30] Radovs Radoicić and Géza Tóth. Monotone paths in line arrangement. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, pages 312—314. Association for Computing Machinery, 2001.

[31] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, 1987.

[32] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461 – 464, 1978.

[33] Robert Sedgewick and Kevin Wayne. *Algorithms.* Addison-Wesley Professional, 4th edition, 2011.

[34] Nithin Shivashankar, M. Senthilnathan, and Vijay Natarajan. Parallel computation of 2D Morse-Smale complexes. *IEEE Transactions on Visualization and Computer Graphics*, 18(10):1757–1770, 2011.

[35] Kevin Verbeek and Willem Sonke. 2IMG10 Topological Data Analysis, 2020. Lecture 9: Stability.

[36] Yan Zheng, Jeffrey Jestes, Jeff M. Phillips, and Feifei Li. Quality and efficiency for kernel density estimates in large data. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, page 433–444, 2013.

[37] Yan Zheng and Jeff M. Phillips. Coresets for kernel regression. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 645–654, 2017.

Appendix A

Changes to colliding bends

For completeness, we provide arguments for each possible BB and BP collision, based on the types of colliding bends.

A.1 BB collisions

To prove that, in a BB collisions, the colliding boundaries inherit each others type, we show that:

1. If bl_j is a starting or an ending h-bend at t_0^- then br_i is a starting or an ending h-bend at t_0^+, respectively;
2. If br_i is a starting or an ending h-bend at t_0^-, then bl_j is a starting or an ending h-bend at t_0^+, respectively;
3. If br_i or bl_j is a regular bend at t_0^-, then bl_j or br_i will respectively be a regular bent at t_0^+.

Since br/bl bends can only be starting h-bends, ending h-bends, or regular bends, this list is exhaustive. The arguments for each of these cases is provided separately below.

1. Let bl_j be a starting h-bend at t_0^-. Then, by Lemma 3, we have

$$\begin{aligned}
& & |R_\ell(l_j(t_0^-))| &= |R_r(l_j(t_0^-))| + 1 \\
\Rightarrow & \quad \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(r_i(t_0^-))| &= |R_r(r_i(t_0^-))| + 1 \\
\Rightarrow & \quad \text{(By Eq. 3.14 \& 3.15)} & |R_\ell(r_i(t_0^+))| &= |R_r(r_i(t_0^+))|
\end{aligned}$$

By Lemma 3, this means that br_i is a starting h-bend at t_0^+.

Now, let bl_j be an ending h-bend at t_0^-. Then, by Lemma 3, we have

$$\begin{aligned}
& & |R_\ell(l_j(t_0^-))| &= |R_r(l_j(t_0^-))| \\
\Rightarrow & \quad \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(r_i(t_0^-))| &= |R_r(r_i(t_0^-))| \\
\Rightarrow & \quad \text{(By Eq. 3.14 \& 3.15)} & |R_\ell(r_i(t_0^+))| + 1 &= |R_r(r_i(t_0^+))|
\end{aligned}$$

By Lemma 3, this means that br_i is an ending h-bend at t_0^+.

2. Let br_i be a starting h-bend at t_0^-. Then, by Lemma 3, we have

$$\begin{aligned}
& & |R_\ell(r_i(t_0^-))| &= |R_r(r_i(t_0^-))| \\
\Rightarrow & \quad \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(l_j(t_0^-))| &= |R_r(l_j(t_0^-))| \\
\Rightarrow & \quad \text{(By Eq. 3.17 \& 3.16)} & |R_\ell(l_j(t_0^+))| &= |R_r(l_j(t_0^+))| + 1
\end{aligned}$$

By Lemma 3, this means that bl_j is a starting h-bend at t_0^+.

Now, let br_i be an ending h-bend at t_0^-. Then, by Lemma 3, we have

$$\begin{aligned} & & |R_\ell(r_i(t_0^-))| + 1 &= |R_r(r_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(l_j(t_0^-))| + 1 &= |R_r(l_j(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.17 \& 3.16)} & |R_\ell(l_j(t_0^+))| &= |R_r(l_j(t_0^+))| \end{aligned}$$

By Lemma 3, this means that bl_j is an ending h-bend at t_0^+.

3. Let br_i be a regular bend at t_0^-. Then, by Lemma 3, we have either

$$\begin{aligned} & & |R_\ell(r_i(t_0^-))| &> |R_r(r_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(l_j(t_0^-))| &> |R_r(l_j(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.17 \& 3.16)} & |R_\ell(l_j(t_0^+))| &> |R_r(l_j(t_0^+))| + 1 \end{aligned}$$

or

$$\begin{aligned} & & |R_\ell(r_i(t_0^-))| + 1 &< |R_r(r_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(l_j(t_0^-))| + 1 &< |R_r(l_j(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.17 \& 3.16)} & |R_\ell(l_j(t_0^+))| &< |R_r(l_j(t_0^+))| \end{aligned}$$

Thus, by Lemma 3, bl_j is a regular bend at t_0^+ in both of these situations.

Now, let bl_j be a regular bend at t_0^-. Then, by Lemma 3, we have either

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| &> |R_r(l_j(t_0^-))| + 1 \\ \Rightarrow & \quad \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(r_i(t_0^-))| &> |R_r(r_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.14 \& 3.15)} & |R_\ell(r_i(t_0^+))| &> |R_r(r_i(t_0^+))| \end{aligned}$$

or

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| &< |R_r(l_j(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.12 \& 3.13)} & |R_\ell(r_i(t_0^-))| &< |R_r(r_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.14 \& 3.15)} & |R_\ell(r_i(t_0^+))| + 1 &< |R_r(r_i(t_0^+))| \end{aligned}$$

Thus, by Lemma 3, br_i is a regular bend at t_0^+ in both of these situations.

A.2 BP collisions

To show the possible events that can happen when a BP collision occurs, we distinguish six different cases, each of which we prove separately. These six cases are:

1. If $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-))$, then bp_i is a single-bend maximum at t_0^- and becomes a starting h-bend at t_0^+. Also, bl_j is an ending h-bend at t_0^- and becomes a regular bend at t_0^+.
2. If $R_\ell(p_i(t_0^-)) + 1 = R_r(p_i(t_0^-))$, then bp_i is a starting h-bend at t_0^- and becomes a regular bend at t_0^+. Also, bl_j is a regular bend at t_0^- and becomes a starting h-bend at t_0^+.
3. If $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-)) + 1$, then bp_i is an ending h-bend at t_0^- and becomes a single-bend maximum at t_0^+. Also, bl_j is a starting h-bend at t_0^- and becomes a regular bend at t_0^+.

4. If $R_\ell(p_i(t_0^-)) + 2 = R_r(p_i(t_0^-))$, then bp_i is a starting h-bend at t_0^- and becomes a regular bend at t_0^+. Also, bl_j is a regular bend at t_0^- and remains regular at t_0^+.
5. If $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-)) + 2$, then bp_i is a regular bend at t_0^- and remains regular at t_0^+. Also, bl_j is a regular bend at t_0^- and becomes an ending h-bend at t_0^+.
6. Otherwise, both bp_i and bl_j are regular at t_0^- and remain regular at t_0^+.

Since these cases cover all possible sizes of $R_\ell(p_i(t_0^-))$ and $R_r(p_i(t_0^-))$, this list is exhaustive. The arguments for each of these cases is provided below.

1. Assume $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-))$. By Equations 3.18 and 3.19, this also means $R_\ell(l_j(t_0^-)) = R_r(l_j(t_0^-))$. Thus, by Lemma 1 and 3, at t_0^- bp_i is a single-bend maximum and bl_j is an ending h-bend. This implies

$$\begin{aligned} & & |R_\ell(p_i(t_0^-))| = |R_r(p_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.20 \& 3.21)} & |R_\ell(p_i(t_0^+))| + 1 = |R_r(p_i(t_0^+))| \end{aligned}$$

and

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| = |R_r(l_j(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.22 \& 3.23)} & |R_\ell(l_j(t_0^+))| = |R_r(l_j(t_0^+))| + 2 \end{aligned}$$

This means, by Lemma 3.3 and 3, that at t_0^+ bp_i will be a starting h-bend and bl_j will be a regular bend.

2. Assume $R_\ell(p_i(t_0^-)) + 1 = R_r(p_i(t_0^-))$. By Equations 3.18 and 3.19, this also means $R_\ell(l_j(t_0^-)) + 1 = R_r(l_j(t_0^-))$. Thus, by Lemma 2 and 3, at t_0^- bp_i is a starting h-bend and bl_j is a regular bend. This implies

$$\begin{aligned} & & |R_\ell(p_i(t_0^-))| + 1 = |R_r(p_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.20 \& 3.21)} & |R_\ell(p_i(t_0^+))| + 2 = |R_r(p_i(t_0^+))| \end{aligned}$$

and

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| + 1 = |R_r(l_j(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.22 \& 3.23)} & |R_\ell(l_j(t_0^+))| = |R_r(l_j(t_0^+))| + 1 \end{aligned}$$

This means, by Lemma 2 and 3, that at t_0^+ bp_i will be a regular bend and bl_j will be a starting h-bend.

3. Assume $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-)) + 1$. By Equations 3.18 and 3.19, this also means $R_\ell(l_j(t_0^-)) = R_r(l_j(t_0^-)) + 1$. Thus, by Lemma 2 and 3, at t_0^- bp_i is an ending h-bend and bl_j is a starting h-bend. This implies

$$\begin{aligned} & & |R_\ell(p_i(t_0^-))| = |R_r(p_i(t_0^-))| + 1 \\ \Rightarrow & \quad \text{(By Eq. 3.20 \& 3.21)} & |R_\ell(p_i(t_0^+))| = |R_r(p_i(t_0^+))| \end{aligned}$$

and

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| = |R_r(l_j(t_0^-))| + 1 \\ \Rightarrow & \quad \text{(By Eq. 3.22 \& 3.23)} & |R_\ell(l_j(t_0^+))| = |R_r(l_j(t_0^+))| + 3 \end{aligned}$$

This means, by Lemma 2 and 3, that at t_0^+ bp_i will be a single-bend maximum and bl_j will be a regular bend.

4. Assume $R_\ell(p_i(t_0^-))+2 = R_r(p_i(t_0^-))$. By Equations 3.18 and 3.19, this also means $R_\ell(l_j(t_0^-))+2 = R_r(l_j(t_0^-))$. Thus, by Lemma 2 and 3, at t_0^- bp_i and bl_j are both regular bends. This also implies

$$\begin{aligned} & & |R_\ell(p_i(t_0^-))|+2 = |R_r(p_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.20 \& 3.21)} & |R_\ell(p_i(t_0^+))|+3 = |R_r(p_i(t_0^+))| \end{aligned}$$

and

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))|+2 = |R_r(l_j(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.22 \& 3.23)} & |R_\ell(l_j(t_0^+))| = |R_r(l_j(t_0^+))| \end{aligned}$$

This means, by Lemma 2 and 3, that at t_0^+ bp_i will be a regular bend and bl_j will be an ending h-bend.

5. Assume $R_\ell(p_i(t_0^-)) = R_r(p_i(t_0^-))+2$. By Equations 3.18 and 3.19, this also means $R_\ell(l_j(t_0^-)) = R_r(l_j(t_0^-))+2$. Thus, by Lemma 2 and 3, at t_0^- bp_i and bl_j are both regular bends. This also implies

$$\begin{aligned} & & |R_\ell(p_i(t_0^-))| = |R_r(p_i(t_0^-))|+2 \\ \Rightarrow & \quad \text{(By Eq. 3.20 \& 3.21)} & |R_\ell(p_i(t_0^+))| = |R_r(p_i(t_0^+))|+1 \end{aligned}$$

and

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| = |R_r(l_j(t_0^-))|+2 \\ \Rightarrow & \quad \text{(By Eq. 3.22 \& 3.23)} & |R_\ell(l_j(t_0^+))| = |R_r(l_j(t_0^+))|+4 \end{aligned}$$

This means, by Lemma 2 and 3, that at t_0^+ bp_i will be an ending h-bend and bl_j will be a regular bend.

6. In any other case, we must have either $R_\ell(p_i(t_0^-))+2 < R_r(p_i(t_0^-))$ or $R_\ell(p_i(t_0^-)) > R_r(p_i(t_0^-))+2$. We handle both of these situations separately.

Assume $R_\ell(p_i(t_0^-))+2 < R_r(p_i(t_0^-))$. By Equations 3.18 and 3.19, this also means $R_\ell(l_j(t_0^-))+2 < R_r(l_j(t_0^-))$. Thus, by Lemma 2 and 3, at t_0^- bp_i and bl_j are both regular bends. This also implies

$$\begin{aligned} & & |R_\ell(p_i(t_0^-))|+2 < |R_r(p_i(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.20 \& 3.21)} & |R_\ell(p_i(t_0^+))|+3 < |R_r(p_i(t_0^+))| \end{aligned}$$

and

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))|+2 < |R_r(l_j(t_0^-))| \\ \Rightarrow & \quad \text{(By Eq. 3.22 \& 3.23)} & |R_\ell(l_j(t_0^+))| < |R_r(l_j(t_0^+))| \end{aligned}$$

This means, by Lemma 2 and 3, that at t_0^+ bp_i and bl_j will both still be regular bends.

Now assume $R_\ell(p_i(t_0^-)) > R_r(p_i(t_0^-))+2$. By Equations 3.18 and 3.19, this also means $R_\ell(l_j(t_0^-)) > R_r(l_j(t_0^-))+2$. Thus, by Lemma 2 and 3, at t_0^- bp_i and bl_j are both regular bends. This also implies

$$\begin{aligned} & & |R_\ell(p_i(t_0^-))| > |R_r(p_i(t_0^-))| + 2 \\ \Rightarrow & \quad \text{(By Eq. 3.20 \& 3.21)} & |R_\ell(p_i(t_0^+))| > |R_r(p_i(t_0^+))| + 1 \end{aligned}$$

and

$$\begin{aligned} & & |R_\ell(l_j(t_0^-))| > |R_r(l_j(t_0^-))| + 2 \\ \Rightarrow & \quad \text{(By Eq. 3.22 \& 3.23)} & |R_\ell(l_j(t_0^+))| > |R_r(l_j(t_0^+))| + 4 \end{aligned}$$

This means, by Lemma 2 and 3, that at t_0^+ bp_i and bl_j will both still be regular bends.