Model-based segmentation and classification of trajectories

Citation for published version (APA):

Alewijnse, S. P. A., Buchin, K., Buchin, M., Sijben, S., & Westenberg, M. A. (2018). Model-based segmentation and classification of trajectories. *Algorithmica*, 80(8), 2422-2452. <https://doi.org/10.1007/s00453-017-0329-x>

DOI:

[10.1007/s00453-017-0329-x](https://doi.org/10.1007/s00453-017-0329-x)

Document status and date:

Published: 01/08/2018

Document Version:

Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Model-Based Segmentation and Classification of Trajectories*

Sander P.A. Alewijnse

Kevin Buchin[§]

Maike Buchin[¶]

Stef Sijben

Michel A. Westenberg[§]

March 9, 2019

Abstract

We present efficient algorithms for segmenting and classifying trajectories based on a movement model parameterised by a single parameter, like the Brownian bridge movement model. Segmentation is the problem of subdividing a trajectory into interior-disjoint parts such that each part is homogeneous in its movement characteristics. We formalise this using the likelihood of the model parameter, and propose a new algorithm for trajectory segmentation based on this. We consider the case where a discrete set of m parameter values is given and present an algorithm to compute an optimal segmentation with respect to an information criterion in $O(nm)$ time for a trajectory with n sampling points. We also present an algorithm that efficiently computes the optimal segmentation if we allow the parameter values to be drawn from a continuous domain.

Classification is the problem of assigning trajectories to classes of similar movement characteristics. The set of trajectories might for instance be the subtrajectories resulting from segmenting a trajectory, thus identifying movement phases. We give an algorithm to compute the optimal classification with respect to an information criterion in $O(m^2 + km \log m)$ time for m parameter values and k trajectories, assuming bitonic likelihood functions. We also show that classification is NP-hard if the parameter values are allowed to vary continuously and present an algorithm that solves the problem in polynomial time under mild assumptions on the input.

1 Introduction

Recent advances in tracking technology lead to increasing amounts of trajectory data being collected. For instance, animals are tracked to understand their movement behaviour, vehicles for analysing traffic situations, and sports players for analysing their play. The analysis of large amounts of these data requires efficient algorithms.

Here we study two fundamental analysis tasks on trajectory data: segmentation and classification. The trajectory of a moving entity is typically recorded as a sequence of time-stamped positions. Segmentation asks to break such a trajectory into interior-disjoint consecutive pieces in a meaningful way. This commonly means that properties of the movement should remain similar within such a piece and should be different between consecutive pieces. Trajectory classification asks to assign trajectories to classes of trajectories with similar properties. More formally, a *segmentation* of a trajectory is a partition of a trajectory into *subtrajectories*, i.e., contiguous subsequences, called *segments*. These segments are disjoint—except for their endpoints, which are also called *splitting points*—and cover the whole trajectory. A *classification* of a set of trajectories $\mathcal{T}$ is a partition of $\mathcal{T}$ into disjoint *classes*.

If the trajectories correspond to periods of homogeneous behaviour, the classification can detect when behavioural states recur. This scenario in particular occurs in the case that we want to assign the segments of a segmented trajectory to classes. These classes might correspond to behavioural states like *flying*,

*This is a post-peer-review, pre-copyedit version of an article published in *Algorithmica*. The final authenticated version is available online at: <https://doi.org/10.1007/s00453-017-0329-x>

[§]TU Eindhoven, The Netherlands, {k.a.buchin,m.a.westenberg}@tue.nl

[¶]Ruhr-Universität Bochum, Germany, Maike.Buchin@rub.de

Figure 1: Examples of classification of fisher tracks. Classes indicated by colour.

foraging, resting etc. This setting is illustrated in Fig. 1, which shows two movement tracks, collected from a fisher [15, 16], a small mammal, first segmented using one of the methods from this paper, and then assigned to two classes.

We note that in this paper we do not assume that labeled data is given, and therefore the algorithmic task is closer to clustering. Nonetheless, we use the more common terminology of trajectory classification, in particular since trajectory clustering is most commonly used in the context of detecting clusters of spatially-close trajectories [4]. In the following we first summarize related algorithmic work on *criteria-based segmentation*. We then describe *model-based segmentation and classification*, before presenting a formal problem statement and our results.

1.1 Criteria-based Segmentation

Previous work on trajectory segmentation in computational geometry has focused on *criteria-based segmentation*, where each segment fulfils given spatio-temporal criteria. An optimal segmentation is one with a minimal number of segments. For this setting several algorithms have been proposed. Buchin *et al.* [7] developed a framework for segmentation given a *decreasing monotone* criterion, that is a criterion which if it holds on a certain segment, also holds on every subsegment of that segment. In this framework a segmentation can usually be computed in $O(n \log n)$ time, where n is the number of sampling points. However, not all natural criteria are decreasing monotone, and for this case Aronov *et al.* [2] developed an algorithm that runs in $\Theta(n^2)$ time. They also study the segmentation problem if the splitting points are allowed to be between the recorded data points. They show that this problem is NP-hard and give polynomial-time algorithms for certain cases. Alewijnse *et al.* [1] proposed a framework that can efficiently handle both decreasing and increasing monotone (defined analogously to decreasing monotone) criteria in $O(n \log n)$ time. Criteria-based segmentation can also be used for classification, by using multiple criteria, one for each class. This setting has been successfully applied to data of migrating geese [8]. Sankararamanan *et al.* [18] proposed to segment a trajectory by detecting similar subtrajectories¹. In movement ecology, classification algorithms have been used to identify behavioural states from acceleration data [17, 19].

1.2 Model-based Segmentation and Classification

Criteria-based segmentation and classification partition data based on pre-specified criteria. The motivation for these tasks –and other movement analysis tasks– is usually to make inferences about the underlying movement process. In the light of this objective it is only natural to take a more statistical perspective on these analysis tasks: As we describe in more detail below, trajectory segmentation and classification can be seen as fitting a parameterised movement model to the data. We propose efficient algorithms to do so.

¹For this they introduce a new model for similarity. The term *model* is used in our paper in a different sense, namely as referring to statistical models.

Taking such an approach is essential when designing algorithms for applications –as in ecology– that use trajectory data in a statistical analysis. In ecology, movement models are used to infer a continuous motion from discrete samples of the movement path. Mostly random movement models, like the Brownian bridge movement model (BBMM) [9, 12] and variants of it [3, 14], Lévy walks [13] and behavioural change point analysis [11] are used. Recently, the BBMM has also been introduced to computational movement analysis [5].

These movement models have a parameter that can be tuned to fit the data. In the case of the BBMM, this parameter is called the *diffusion coefficient* and depends on the magnitude of changes in speed and direction of motion. Often, probability-based approaches such as maximum likelihood estimation are used to optimize the parameter value for a given data set.

In these movement models, a link l , i.e. the part of the trajectory between two consecutive observations, has an associated log-likelihood $L_l(x)$ as a function of the model parameter x , indicating how well the model fits the data for each possible value of x . The log-likelihood of a parameter value x for a (sub)trajectory (e.g. a segment) τ is given by summing the log-likelihoods of all the links in τ : $L_\tau(x) = \sum_{l \in \tau} L_l(x)$.

We use the term *partition* to refer to either a segmentation or classification, which additionally assigns a value $x(P)$ of the model parameter to each part P (i.e. a segment or class). A partition $\mathcal{P}$ has an associated log-likelihood, which is defined as

$$L_{\mathcal{P}} = \sum_{P \in \mathcal{P}} \sum_{e \in P} L_e(x(P)) . \quad (1)$$

That is, the log-likelihood of each part P (i.e., segment or class) is the sum over the log-likelihoods of its elements e (i.e., single links or trajectories), evaluated at the value of the model parameter $x(P)$ assigned to P . The log-likelihood of $\mathcal{P}$ is the sum of the log-likelihoods of its parts.

We could now define an optimal partition as one that maximises the log-likelihood, but then it would be optimal to put each link or trajectory into its own part, resulting in the largest possible number of degrees of freedom for the model. One solution is to fix the number of parts, but typically the number of parts is not known beforehand. To determine a good number of segments or classes an information criterion like the Bayesian information criterion can be used [14, 11].

To facilitate multi-scale analysis, we use a more general notion of an *information criterion* (IC) to define the optimal partition. An IC assigns a value to each partition based on its log-likelihood and the complexity of the model (that is, the number of parts). In particular we consider ICs of the form

$$IC(\mathcal{P}) = -2L_{\mathcal{P}} + |\mathcal{P}| \cdot p , \quad (2)$$

where $L_{\mathcal{P}}$ is the log-likelihood of the model instance and $|\mathcal{P}|$ is the number of parts of the partition. The number $p \geq 0$ is a penalty factor for adding complexity to the model that counteracts overfitting. The two most commonly used information criteria are the Bayesian information criterion (BIC) and the Akaike information criterion (AIC), which are special cases of Equation 2. The BIC has a penalty $p = \ln(n)$, where n is the number of measured locations for segmentation or the number of trajectories for classification. The AIC has a constant penalty $p = 2$. A stability diagram as presented by Alewijnse *et al.* [1] can be used to find a suitable penalty factor for a given data set.

The segmentation in Fig. 1 was computed using log-likelihood functions based on the BBMM and with the BIC penalty factor. Each trajectory produced 5 segments, and these 10 segments were given as input to our (discrete) classification algorithm, using the same log-likelihood functions and penalty factor. This resulted in two classes, indicating that the animal was alternating between two behavioural states.

1.3 Problem Statement and Results

We assume a *trajectory* τ is given by a sequence of n time-stamped positions $(\vec{p}_i, t_i)$ ($1 \leq i \leq n$), where $\vec{p}_i =: \tau(i)$ is the location of a moving entity at time t_i . We denote by $\tau[i, j]$ the subtrajectory of τ starting at time t_i and ending at time t_j . We use k to denote the number of segments in a segmentation, s_i ($0 \leq i \leq k$) for the index of the i th splitting point, and $\tau(s_i)$ for the i th splitting point. The resulting segmentation is $S_1 = \tau[1 =: s_0, s_1]$, $S_2 = \tau[s_1, s_2]$, $\dots$, $S_k = \tau[s_{k-1}, s_k := n]$. Furthermore we denote with $x(S_i)$ the model parameter for the segment S_i .

We define an optimal segmentation to be one that minimises the value of the IC.

Definition 1.1. An *optimal segmentation* of a trajectory τ is a segmentation $\{S_1, \dots, S_k\}$ and selection of model parameters for the segments $x(S_i)$ ($1 \leq i \leq k$) that achieves the minimum value for the information criterion among all segmentations and parameter values for τ .

When classifying a set of trajectories $\mathcal{T}$, we use ℓ to denote the number of classes and $C_1, \dots, C_\ell \subseteq \mathcal{T}$ to denote the classes. Again, each class C_i is assigned a value of the model parameter $x(C_i)$.

Definition 1.2. An *optimal classification* of a set of trajectories $\mathcal{T}$ is a classification $\{C_1, \dots, C_\ell\}$ and selection of model parameters for the classes $x(C_i)$ ($1 \leq i \leq \ell$) that achieves the minimum value for the information criterion among all classifications and parameter values for $\mathcal{T}$.

For the purpose of classification, we assume that the log-likelihood functions L_i are bitonic, i.e. that they have one maximum and are increasing before that and decreasing after. We also assume without loss of generality that the functions are given in increasing order by the parameter value at which they reach their maximum. That is, if we define $M_i := \arg \max_x (L_i(x))$, then $i < j \Rightarrow M_i \leq M_j$. If M_i is not uniquely defined for a function, it is the smallest value where the function reaches its maximum. We represent a classification $\mathcal{C}$ for $L_1, \dots, L_k$ by an array of length k where $\mathcal{C}[i]$ is the parameter value that the classification assigns to the class of L_i , or $\mathcal{C}[i] = \text{NIL}$ if $\mathcal{C}$ is a partial classification that does not classify L_i .

The algorithms we present do not use the trajectory data as such, but only the log-likelihood functions derived from them. In the following, we use log-likelihood functions as representatives for (sub)trajectories.

In this paper we develop efficient algorithms to compute optimal segmentations and classifications. To simplify the problems we first consider a discrete version of both problems. In this setting we assume that the parameter values $x(P)$ are drawn from a finite set of candidate values $X = \{x_1, \dots, x_m\}$ (in sorted order) and that the log-likelihood functions L_i are given by listing the values they take on $x_1, \dots, x_m$. Then we consider the case where the parameter values are drawn from a continuous domain $\mathcal{D}$, which is usually an interval on the real line.

In Section 2 we give efficient algorithms for computing an optimal segmentation in both the discrete and continuous case using dynamic programming. In particular, we use table compression to speed up the computation. In Section 3 we give an efficient algorithm for computing an optimal discrete classification using again dynamic programming. Then we show that classification is NP-hard if arbitrary log-likelihood functions are allowed and that the continuous classification problem is NP-hard even with bitonic log-likelihood functions. However, under mild assumptions on the input we give a geometric representation of the problem that allows to solve problem in polynomial time using geometric techniques. We focus on movement models with a single model parameter, such as the BBMM, and discuss the case of multi-parameter models at the end of the paper.

2 Segmentation

We present algorithms for trajectory segmentation based on a parameterised movement model, which compute an optimal segmentation with respect to a given information criterion IC. We solve the problem using dynamic programming, first for the case of a discrete set of candidate parameter values, then for a continuous domain.

2.1 Discrete model parameter

Let Opt_i denote the optimal segmentation of $\tau[1, i]$ with respect to IC. We compute all Opt_i in increasing order of i . In order to compute Opt_i we maintain a two-dimensional table O , where the columns represent trajectory vertices and the rows represent possible values of the model parameter. Each entry $O_{i,x}$ stores the optimal segmentation (minimising the IC) of subtrajectory $\tau[1, i]$ that ends with a segment S_k with parameter value $x(S_k) = x$.

Let $X = \{x_1, x_2, \dots, x_m\}$ denote the set of candidate parameter values. Given $O_{i,x}$ for all $x \in X$, it is straightforward to compute Opt_i : since every segment, in particular the last, has a parameter value in X , Opt_i is the entry $O_{i,x}$ that has minimal IC among all $x \in X$. The table O is computed using the greedy property stated in the following lemma.

Lemma 2.1. $O_{i,x}$ is equal to one of the following options:

Append: Opt_{i-1} appended with the one-link segment $\tau[i-1, i]$.

Extend: $O_{i-1,x}$ with the last segment extended by $\tau[i-1, i]$.

Proof. The last segment of $O_{i,x}$ consists of one link $\tau[i-1, i]$, or it consists of strictly more than one link.

- **Case 1: Last segment consists of one link.** The last segment is in this case a one-link segment $S_k = \tau[i-1, i]$ with $x(S_k) = x$. We are left with the part $\tau[1, i-1]$. Consider the information criterion defined in Equation 2. Each segment corresponds to its own independent term in the log-likelihood. Hence the part $\tau[1, i-1]$ should be segmented optimally independently of the rest of the trajectory. By definition, this results in segmentation Opt_{i-1} , which proves the append option.

- **Case 2: Last segment consists of strictly more than one link.** In this case $O_{i,x}$ is set to $O_{i-1,x}$, with the last segment extended by one link. We prove that this greedy choice is correct.

Let $\mathcal{S}_1$ denote the segmentation that is equal to $O_{i-1,x}$, with the last segment extended by one link. Let $\mathcal{S}_2$ denote the segmentation of subtrajectory $\tau[1, i-1]$ according to $O_{i,x}$ (with the last segment one link shorter). Since $O_{i,x}$ minimises the IC we have

$$\text{IC}(O_{i,x}) \leq \text{IC}(\mathcal{S}_1) .$$

Substitution of both values yields:

$$\text{IC}(\mathcal{S}_2) - 2L_i(x) \leq \text{IC}(O_{i-1,x}) - 2L_i(x) ,$$

where $L_i(x)$ is the log-likelihood of the link $\tau[i-1, i]$, evaluated at x . This implies that

$$\text{IC}(\mathcal{S}_2) \leq \text{IC}(O_{i-1,x}) .$$

By definition, segmentation $O_{i-1,x}$ minimises the information over all possible values of $\mathcal{S}_2$. Hence $\text{IC}(\mathcal{S}_2) = \text{IC}(O_{i-1,x})$. Greedily choosing $\mathcal{S}_2 = O_{i-1,x}$ is hence correct. This proves the extend option. $\square$

$\square$

Lemma 2.1 implies that we can compute $O_{i,x}$ in a dynamic programming fashion, looping over i and x . We compute each new entry $O_{i,x}$ using a comparison between two already computed table entries:

$$\text{IC}(O_{i,x}) = \min(\text{IC}(O_{i-1,x}), \text{IC}(Opt_{i-1}) + p) - 2L_i(x) .$$

If $\text{IC}(Opt_{i-1}) + p$ is smaller than $\text{IC}(O_{i-1,x})$, then $O_{i,x}$ is set according to the append option. Otherwise it is set according to the extend option.

In our algorithm we store segmentations by maintaining only the length l and parameter value x for the last segment ending at $\tau(i)$. The previous segments can be worked out by looking at Opt_{i-l} . Furthermore, we store for each segmentation the information according to IC. Using this storage format computing $O_{i,x}$ given $O_{i-1,x}$ and Opt_{i-1} takes only constant time. The actual segmentation can be retrieved from the tables in $O(n)$ time.

Theorem 2.2. The optimal segmentation of a trajectory τ with respect to an information criterion IC can be computed in $O(nm)$ time, and $O(n+m)$ space, where m is the number of candidate values for the model parameter and n the number of points on the trajectory.

Proof. Our algorithm computes the table $O_{i,x}$, which has size nm . Using Lemma 2.1, this takes constant time per entry, hence $O(nm)$ time in total. Computing the table with the overall optimal segmentations Opt_i takes $O(m)$ time per entry, hence also $O(nm)$ time in total. To compute an entry $O_{i,x}$, only $O_{i-1,x}$ is needed. Hence only one column (with m entries) of the table for O needs to be stored, in addition to the table for Opt , which has n entries. The algorithm thus requires only $O(n+m)$ space. $\square$ $\square$

2.2 Continuous model parameter

So far, we assumed the model parameter to come from a discrete set of candidate values. We now consider the case where the model parameter is allowed to take any value from a continuous domain (e.g. an interval on the real line). We observe that a segment is defined by a starting index $j \in \{1, \dots, n-1\}$ and an ending index $i \in \{j+1, \dots, n\}$. That is, there are only $O(n^2)$ different segments that can occur in the segmentation. Assume that the optimum model parameter $x(\tau[j, i])$ for a segment $\tau[j, i]$ can be computed in $F(i-j)$ time, for some nondecreasing function F .

As before, let Opt_i denote the optimal segmentation of $\tau[1, i]$ with regard to the IC. Then Opt_1 is empty, since $\tau[1, 1]$ contains only a single point. Opt_i for $i > 1$ consists of an optimal segmentation of $\tau[1, j]$ for some $j \in \{1, \dots, i-1\}$, combined with a single segment for $\tau[j, i]$.

Theorem 2.3. *The optimal segmentation of a trajectory τ with respect to an information criterion IC can be computed in $O(n^2 F(n))$ time, where n is the number of points on the trajectory and $F(k)$ is the amount of time needed to compute the optimum model parameter value for a fixed subtrajectory of length k . The algorithm uses $O(n)$ space, excluding space required for optimizing the parameter value.*

Proof. The IC for each candidate segmentation for Opt_i can be computed in $O(F(i))$ time, since the IC values of $\tau[1, j]$ ($1 \leq j \leq i-1$) are stored in the table for Opt and computing the optimum log-likelihood value for $\tau[j, i]$ takes $O(F(i))$ time. Since $i-1$ candidate segmentations are considered for Opt_i , with i ranging from 2 to n , we consider $O(n^2)$ candidate segmentations for subtrajectories of τ .

Thus, Opt_n , which is the optimal segmentation of τ can be computed in $O(n^2 F(n))$ time. We only need to store the Opt table of size $O(n)$, so the algorithm uses $O(n)$ space, or what is needed to optimise the parameter value for a candidate segment. $\square$ $\square$

For many practical log-likelihood functions like those used in the BBMM, $x(\tau[i, j])$ can be computed to a fixed precision in linear time and space, i.e. $F(n) = O(n)$, leading to an overall running time of $O(n^3)$ and space use of $O(n)$. The models often do not allow for analytic solutions, so numerical methods are required to optimise the parameter values. Generally, optimising the log-likelihood to fixed precision will produce the optimal segmentation, unless there are multiple classifications that have almost identical IC. This case can be detected and the functions optimised to higher precision, or one might accept solutions that are slightly suboptimal. The algorithm can also be adapted to report multiple solutions if it can't distinguish their IC.

If the log-likelihood functions allow for a closed-form expression of constant complexity, we can optimise each subtrajectory in amortised constant time and thus compute the optimal segmentation in $O(n^2)$ time. For example, using the BBMM where the observed positions have no location uncertainty, the log-likelihood of a (sub)trajectory is of the form

$$L(x) = c_1 + c_2 \log(x) + \frac{c_3}{x}, \quad (3)$$

where c_1 , c_2 and c_3 are constants that can be efficiently derived from the trajectory data. Similarly, if the location is observed at regular intervals and all positions have identical location uncertainty δ , the log-likelihood function has the form

$$L(x) = c_1 \log(c_2 \cdot x + c_3) + \frac{c_4}{x + c_5}, \quad (4)$$

with c_1 , c_2 , c_3 , c_4 and c_5 constants.

Data set	Trajectories	Total length	Max. length	m	Max. w
Fisher	8	32,904	13,099	100	12
				1,000	12
Gulls	98	2,485,399	111,192	100	11
	1	111,192		1,000	10
				5,000	11
Vervet monkeys	4	6,432	1,754	100	12
				1,000	12

Table 1: Results of experiments on table compression in practice.

2.3 Fixed number of segments

It is straightforward to change the algorithms in such a way that they compute the segmentation with a fixed number k of segments and maximal log-likelihood. All tables get an extra dimension: the number of segments. Those tables can be computed using a greedy property that is similar to Lemma 2.1. To compute $O_{i,x,k}$ (with k the number of segments) we choose between either appending a one-link segment to $Opt_{i-1,k-1}$ or extending the last segment of $O_{i-1,x,k}$ by one link. Each table entry is hence computed in constant time, which results in $O(nmk)$ running time and $O((n+m)k)$ required space for discrete segmentation. For continuous segmentation the algorithm needs $O(n^2(F(n)+k))$ time and $O(kn)$ space. Note that these algorithms do not only give the optimal segmentation with k segments, but also the optimal segmentations with $k-1, k-2, \dots, 1$ segments. These segmentations can be used to compute stability diagrams [1] that aid a user in selecting a value for the penalty factor.

2.4 Table compression

When studying the DP tables (for BBMM) we observed a way to speed up the basic segmentation algorithm from Section 2. Let $\text{last}(\mathcal{S})$ denote the starting index of the last segment of segmentation $\mathcal{S}$. We noticed that the variation of $\text{last}(O_{i,x})$ for a fixed i over all x is very limited. Given an ordered set of parameter values $x_1 < x_2 < \dots < x_m$ there seems to be only a small number w of values (compared to m and n) x_j for which $\text{last}(O_{i,x_j}) \neq \text{last}(O_{i,x_{j+1}})$. This observation is supported by experimental evaluation. We segmented trajectories from several data sets from movement ecology, collected from fishers [15, 16], gulls [20] and vervet monkeys [6], with input sizes up to $n = 111,192$ and $m = 5,000$. The largest number of changes in the value of last in any single column of a table in these experiments is $w = 12$, so much smaller than m . In our experiments, w did not depend strongly on n and m : There were many trajectories of varying lengths in the data that had w close to 12 and w usually did not change when increasing m . The results of our experiments are summarised in Table 1.

We improve the original algorithm by storing the DP table in a compressed way. Multiple consecutive entries of $O_{i,x}$ with equal value for $\text{last}(O_{i,x})$ can be compressed into one entry $O_{i,[x_s,x_f]}$. Note that not all parameter values that are stored in the same compressed table entry have identical information. The information of a segmentation $O_{i,x}$ with $x \in [x_s, x_f]$ consists of two parts: a variable part for the last segment S_k , which depends on the exact value of $x(S_k) \in [x_s, x_f]$, and a constant part for the rest of the segmentation, which is $\text{IC}(Opt_l)$, with $l = \text{last}(O_{i,[x_s,x_f]})$. The constant part can be stored without causing problems to the compression.

The change in our DP table representation causes the need for several changes in the DP algorithm. First of all, we do not loop over a sampled set X of parameter values. Instead, we loop over intervals that together cover the whole parameter space. In the original algorithm we compared the information of the append and extend option. This was a simple comparison of two real numbers. In our compressed setting the comparison between append and extend requires comparing two functions, which we call the *append* and *extend function*. Computing the lower envelope takes time linear in the complexity of the envelope, if the likelihoods of the (sub)trajectories can be expressed with a closed-form expression of constant complexity, such as the ones of Equations 3 and 4. When the lower envelope is not completely equal to one of the two

compared functions, the interval is split in a number of parts: parts of the append function and parts of the extend function.

Extending the last segment or appending a new segment with a different parameter value takes constant time given a suitable representation of the information. After looping over all intervals we merge neighbouring intervals that have equal last value. This guarantees that computing each column of the compressed $O_{i,[x_s, x_f]}$ table takes $O(w)$ time, where w is the maximum number of indices where last changes (i.e. the maximum number of intervals) in one table column.

The segmentation Opt_i is equal to the $O_{i,x}$ with minimal IC. In the original algorithm we take the minimum over all $x \in X$. In a compressed table, we find the best $x \in [x_s, x_f]$ for every interval, and take the best of those segmentations. Computing the best parameter value in an interval takes constant time given a suitable representation of the extend function. The following theorem summarises our analysis.

Theorem 2.4. *Given a trajectory τ consisting of n points and using a movement model that allows for expressing the likelihoods of (sub)trajectories using a closed-form expression of constant complexity, the optimal segmentation with respect to an information criterion IC can be computed in $O(nw)$ time, where w is the maximal number of indices at which the start index $\text{last}(O_{i,x})$ changes along x for any single i .*

3 Classification

We now consider the problem of classification, i.e. given a set of trajectories $\mathcal{T} = \{\tau_1, \dots, \tau_k\}$ and a penalty factor p , the goal is to compute a partition of $\mathcal{T}$ and assign a parameter value to each class that minimises an information criterion.

We assume that the log-likelihood functions for all trajectories are bitonic and that the log-likelihood $L(\tau_i)$ of trajectory τ_i reaches its maximum at the parameter value M_i . We also assume w.l.o.g. that the trajectories are given in order of increasing M_i . If this is not the case, they can easily be sorted in a preprocessing step.

The algorithms in this section depend critically on the assumption of bitonic likelihood functions. Without this, one can construct examples that violate key properties such as the one given in Observation 1 and our algorithms no longer work. We show in Section 3.2 that both discrete and continuous classification are NP-hard if non-bitonic log-likelihood functions are allowed. We justify the assumption of bitonicity with a reference to the applications that these algorithms were designed for. Although it is possible to construct trajectories for which the likelihood function of a trajectory using the BBMM is not bitonic, these examples are highly unlikely to occur in practice: They require large variations in both sampling rate and speed of about a factor 100. Even in this case, both the parameter values and log-likelihoods of the local maxima are close to one another and thus the algorithm would still produce an almost optimal solution. Again, one can also check the bitonicity in preprocessing and warn the user if there are non-bitonic input functions.

3.1 Dynamic Programming for Discrete Parameter

First we present a dynamic programming algorithm for the case that a discrete set of parameter values is given. A natural approach would be to process the trajectories in the order they are given. However it is not necessarily the case that if $M_i < M_j$ that this order is also reflected in the classes they are associated with. Fig. 2 shows an example. However, we can use the following property to efficiently compute the optimal classification.

Observation 1. *Let $x(C_1) < \dots < x(C_\ell)$ be the parameter values assigned to the classes by an optimal classification. Then, a trajectory that reaches its maximum log-likelihood in the interval $[x(C_j), x(C_{j+1}))$ must be in either C_j or C_{j+1} , by the bitonicity of the log-likelihood function. In particular if we know that some $x(C_i)$ is selected then $x(C_j)$ with $j < i$ does not depend on any of the trajectories with maximum larger than or equal to $x(C_i)$.*

Using this observation, the optimal classification can be efficiently computed. For a problem instance $(L_1, \dots, L_k), (x_1, \dots, x_m)$, add dummy values $x_0 := -\infty$ and $x_{m+1} := \infty$. Let $\mathcal{L}_{i,j} := \{L_l \in \{L_1, \dots, L_k\} \mid x_i \leq M_l < x_j\}$, with $0 \leq i < j \leq m+1$, denote the set of functions that reach their maximum log-likelihood at a

Figure 2: An optimal classification may not respect the order in which trajectories obtain their maximum log-likelihood. For low penalty factors, there will be two classes $C_1 = \{L_1, L_3\}$ and $C_2 = \{L_2, L_4\}$ with $x(C_1) = x_1$ and $x(C_2) = x_2$.

```

DISCRETECLASSIFICATION( $(L_1, \dots, L_k), (x_1, \dots, x_m)$ )
1   $Opt_0$  = An array of length  $m$  with all elements set to NIL
2   $\mathcal{C}$  = An arbitrary (complete) classification of  $L_1, \dots, L_k$ 
3  for  $i = 1$  to  $m$ 
4      for  $j = 0$  to  $i - 1$ 
5           $O_j = Opt_j$ 
6          for  $L_l \in \mathcal{L}_{j,i}$ 
7               $O_j[l] = \arg \max_{x \in \{x_j, x_i\}} (L_l(x))$ 
8           $Opt_i = \arg \min_{\{O_j | 0 \leq j < i\}} IC_i(O_j)$ 
9           $\mathcal{C}_i = Opt_i$  with all NILs replaced by  $x_i$ 
10         if  $IC(\mathcal{C}_i) < IC(\mathcal{C})$ 
11              $\mathcal{C} = \mathcal{C}_i$ 
12 return  $\mathcal{C}$ 

```

Algorithm 1: Discrete classification.

parameter value between x_i and x_j . For $i \in \{1, \dots, m\}$, let $Opt_i = \{C_1, \dots, C_\ell\}$ be an optimal classification of $\mathcal{L}_{0,i}$, conditioned on $x(C_\ell) = x_i$, even if C_ℓ is empty. Let $\mathcal{C}_i$ denote the complete classification that is obtained from Opt_i by assigning all trajectories in $\mathcal{L}_{i,m+1}$ to C_ℓ . The following lemma suggests a means to efficiently compute Opt_i .

Lemma 3.1. *Opt_i consists of an optimal classification of $\mathcal{L}_{0,j}$ for some $j \in \{0, \dots, i-1\}$ extended with a class with parameter value x_i , by assigning each trajectory in $\mathcal{L}_{j,i}$ the parameter value x_j or x_i that has the highest log-likelihood.*

Proof. Let $x(C_1) < \dots < x(C_{\ell-1}) < x(C_\ell) = x_i$ be the parameter values for the classes in Opt_i . If $\ell = 1$, all trajectories are assigned to one class, which we view as an extension of the empty classification Opt_0 . Otherwise $x(C_{\ell-1}) = x_j$ for some $j < i$, since the parameter values come from $\{x_1, \dots, x_m\}$. By Observation 1, Opt_i consists of an optimal classification of $\mathcal{L}_{0,j}$ (that uses x_j), i.e. Opt_j , and the trajectories in $\mathcal{L}_{j,i}$ are independently assigned to either $C_{\ell-1}$ or C_ℓ . $\square$ $\square$

The pseudocode of the classification algorithm is given in Algorithm 1. IC is the function that computes the value of the information criterion for a given (partial) classification. IC_i computes the IC under the assumption that x_i is used, even if no trajectory has been assigned to x_i . We do this to take into account that we assign the remaining trajectories to x_i in the next step. Comparing values of the IC is meaningful only between partial classifications which assign values to exactly the same trajectories, as is the case for all O_j in a single iteration of the outer loop.

Theorem 3.2. *Algorithm 1 computes the optimal classification of k trajectories with respect to an information criterion in $O(km^2)$ time and $O(mk)$ space, where m is the number of candidate parameter values.*

Proof. For the correctness observe that the optimal classification $\mathcal{C}$ is one of the $\{\mathcal{C}_i | 1 \leq i \leq m\}$ computed by the algorithm: Let x_i be the largest value that is selected in $\mathcal{C}$. Then all trajectories in $\mathcal{L}_{i,m+1}$ are assigned

to x_i , and the other trajectories are assigned according to an optimal classification of $\mathcal{L}_{0,i}$ conditioned on using x_i , i.e. Opt_i . This is exactly the $\mathcal{C}_i$ computed in the algorithm.

The running time and space use are as follows. The outer loop has m iterations. The middle and inner loops have $O(m)$ and $O(k)$ iterations respectively. These middle and inner loops with all the operations in them take $O(mk)$ time. Computing the IC of $O(m)$ classifications, each of size k , takes $O(mk)$ time too, and all the other operations need less time than that, so the running time of the algorithm is $O(km^2)$. At any time, the algorithm stores $O(m)$ (partial) classifications, each of size k , leading to a space use of $O(mk)$. $\square$ $\square$

3.1.1 Improvement to Running Time

The running time of the discrete classification algorithm described in Section 3.1 can be improved by re-using values computed in previous iterations of the outer loop.

Observation 2. Let O_j^i be the value that O_j has at the end of the i th iteration of the outer loop. We observe that for all functions $L \in \mathcal{L}_{j,i}$, the value assigned to L in O_j^i is either x_j or x_i . Due to the bitonicity of L , for a fixed j there is a value s , $j \leq s \leq m+1$, such that x_i is assigned to L for all $i < s$ and x_j is assigned for all $i \geq s$.

A *switching value* is a pair (l, s_j) , where l is the index of a function in the input, i.e. an integer in $\{1, \dots, k\}$ and s_j is defined by

$$s_j := \min_{j < i \leq m} \{i \mid L_l(x_j) \geq L_l(x_i)\} ,$$

i.e. s_j is the smallest value of i such that L_l is assigned to x_j instead of x_i in O_j^i . For each of the k input functions there are m switching values, corresponding to all possible values of j . We store all switching values in m lists of length k each, such that each list S_j contains for each function the value of the loop counter i at which the selected value for this function in O_j switches from x_i to x_j in line 7 of Algorithm 1, sorted by the order these switches will occur. Using the bitonicity of L_l and binary search, each switching value can be computed in time $O(\log m)$ time, for a total of $O(km \log m)$. All lists are then sorted by increasing s_j using a linear time sorting algorithm such as counting sort in time $O(m(k+m))$.

We can exploit this to quickly update the value of O_j that was computed in the previous iteration of the outer loop. The differences between O_j^{i-1} and O_j^i can be summarised as:

1. For each $L_l \in \mathcal{L}_{i-1,i}$, a new value $O_j^i[l] = x_i$ is added. We observe that every function is added at most once for a fixed value of j and the functions are added in the order they appear in the input.
2. Some functions which were assigned to x_{i-1} in O_j^{i-1} are now assigned to x_j . These are exactly the functions L_l such that there is an entry (l, i) in S_j . Since S_j is sorted by the second member of each pair, we walk through S_j once for all updates of O_j , so all these updates for O_j take $O(k)$ time. Note that this case may also apply to functions that were added in the same iteration based on case 1.
3. The functions that were assigned to x_{i-1} in O_j^{i-1} and for which case 2 does not apply will now be assigned to x_i . Since there may be $O(k)$ functions in this list which are updated in every iteration, we need to efficiently perform these updates. We can have the classification update automatically by storing a special value indicating that this function is assigned to x_i regardless of the current value of i .

Using this approach, we can compute all the O_j^i ($j < i \leq m$) in $O(k+m)$ time for a fixed j , for a total of $O(m(k+m))$ time, given the lists of switching values.

However, we also need to evaluate the information criterion for each of these O_j^i , which takes $O(km^2)$ time if every value is computed from scratch. Thus, we also need to update the IC more efficiently in order to improve the running time of the algorithm. For cases 1 and 2, this is no problem, as each change in IC due to these cases can be computed in constant time and there are only $O(km)$ additions or updates.

Figure 3: For each function L_l with switching values $(l, s_j) \in S_j$ ($1 \leq j \leq m$), the s_j are bitonic in j . The dots on the graph indicate the candidate values for the classification. ((a)): If $x_j \leq M_l$, $s_j \leq s_{j-1}$, so the s_j are decreasing. ((b)): If $s_j \geq M_l$, either $s_j = j$ or x_{s_j} is at the end of an interval containing x_j where L_l is constant, so the s_j are increasing in j .

However, in case 3, there may be $\Theta(km^2)$ updates in total, thus leading to a bottleneck. We can summarise the change in IC for O_j^i due to functions that were assigned to x_{i-1} in O_j^{i-1} and are now assigned to x_i as

$$\begin{aligned} \Delta_{j,i} &:= \sum_{L_l \in I_{j,i}} L_l(x_i) - L_l(x_{i-1}) \\ &= \Delta_{j-1,i} + \sum_{L_l \in I_{j,i}^+} L_l(x_i) - L_l(x_{i-1}) - \sum_{L_l \in I_{j,i}^-} L_l(x_i) - L_l(x_{i-1}) , \quad \text{with} \\ I_{j,i} &:= \{L_l \in \mathcal{L}_{j,i} \mid (l, s_j) \in S_j \wedge s_j > i\} , \\ I_{j,i}^+ &:= I_{j,i} \setminus I_{j-1,i} , \\ I_{j,i}^- &:= I_{j-1,i} \setminus I_{j,i} . \end{aligned}$$

$I_{j,i}$ is the set of likelihood functions that are assigned to x_i in O_j^i . We observe that the change $\Delta_{j,i}$ in IC does not depend on j and can thus be computed from $\Delta_{j-1,i}$ by adding some terms to the sum and removing some others. If we can bound the size of $I_{j,i}^+$ and $I_{j,i}^-$ and also compute these efficiently, we can efficiently compute the value of $\text{IC}_i(O_j^i)$ from $\text{IC}_{i-1}(O_j^{i-1})$ and thus improve the running time of the algorithm.

Lemma 3.3. *Let $(l, s_j) \in S_j$ (for all $1 \leq j \leq m$) be the switching values of a function L_l . The s_j are a bitonic function of j (specifically first decreasing, then increasing).*

Proof. Consider two switching values (l, s_j) and (l, s_{j+1}) for an input function L_l . There are two cases:

1. $x_{j+1} \leq M_l$: Both x_j and x_{j+1} are in the increasing part of L_l . Since $x_j < x_{j+1}$ and L_l is increasing there, $L_l(x_j) \leq L_l(x_{j+1})$. Therefore we must have that $L_l(x_{s_j}) \leq L_l(x_{s_{j+1}})$.
By definition of s_j , it is the smallest i such that the log-likelihood at x_j is at least as large as that at x_i . Therefore, x_i must be in the decreasing part of L_l .
2. $x_j \geq M_l$: Both x_j and x_{j+1} are in the decreasing part of L_l , so the log-likelihood is at most $L_l(x_j)$ for all $i > j$. Therefore $x_j = j+1$ and $x_{j+1} = j+2$, so $x_j < x_{j+1}$ and the s_j are increasing for all j such that $x_j \geq M_l$. These two observations imply $s_j \geq s_{j+1}$ and thus the s_j are decreasing for all j such that $x_{j+1} \leq M_l$.

□

□

This property is illustrated in Fig. 3. As a result, for fixed i , a function is present in at most one $I_{j,i}^+$ and at most one $I_{j,i}^-$.

Lemma 3.4. $I_{j,i}^+$ is empty for all $1 \leq j < i \leq k$.

Proof. Assume $L_l \in I_{j,i}^+$. The definition of $I_{j,i}^+$ requires that the switching values of L_l are strictly increasing between $j-1$ and j . This only occurs if $L_l(x_j) < L_l(x_{j-1})$ and thus $M_l \leq x_{j-1}$. But then $L_l \notin \mathcal{L}_{j,i}$ and thus $L_l \notin I_{j,i}^+$, which is a contradiction. $\square$

Lemma 3.5. *The total size of all $I_{j,i}^-$ ($1 \leq j < i \leq m$) is at most km and they can all be computed in time $O(km \log m)$.*

Proof. For each value of i and each function L_l with $M_l < x_i$ (since otherwise $L_l \notin \mathcal{L}_{j,i}$ for any j), we want to find the unique value of j such that $L_l \in I_{j,i}^-$, i.e. the minimum j s.t. $L_l \notin I_{j,i}$. L_l is not in $I_{j,i}$ either because $L_l \notin \mathcal{L}_{j,i}$ (i.e. $M_l < x_j$) or j is the smallest value for which the entry (l, s) in S_j has $s \leq i$. For each function, we construct a list J_l of length m which is defined by $J_l[j] := s$ such that $(l, s) \in S_j$. That is, for each value of j , J_l stores the smallest i such that L_l is assigned to x_j in O_j^i . All the J_l can be constructed in $O(km)$ time by a scan over all the S_j : For an entry $(l, s) \in S_j$, we set $J_l[j] = s$. Note that the J_l are bitonic by Observation 3.3, specifically first decreasing, then increasing.

Now, for each L_l and $i \in \{1, \dots, m\}$ such that $x_i > M_l$, we look up the smallest j such that $J_l[j] \leq i$, since that indicates $L_l \in I_{j,i}^-$. This j always exists and is at most such that $x_j = M_l$. Using binary search for these lookups, we can thus construct all $I_{j,i}^-$ in $O(km \log m)$ time. $\square$

Given all $I_{j,i}^-$ we can compute all $\Delta_{j,i}$ in $O(km)$ time and use these values to update $\text{IC}_i(O_j^i)$ for the functions that are in case 3 described above. This leads to the following theorem:

Theorem 3.6. *With the optimizations described in this section, Algorithm 1 computes the optimal classification of k trajectories in $O(m^2 + km \log m)$ time and $O(mk)$ space, where m is the number of candidate parameter values.*

Proof. The time bound has already been established. In addition to the space use of the original algorithm, the tables S , Δ and J all have mk entries and thus do not increase the space bound. $\square$

3.2 NP-hardness with Non-bitonic Log-likelihood Functions

We show that both discrete and continuous classification are NP-hard by reduction from 3-SAT. Consider a Boolean formula F in conjunctive normal form with n variables and m clauses. We have a candidate parameter value x_t for each literal t , i.e. v or $\bar{v}$ for any variable v . For discrete classification these are exactly the candidate parameter values, for continuous classification we choose an interval that contains all x_t as the domain. The log-likelihood functions are constructed in such a way that an optimal classification only selects classes with $x(C) = x_t$ for some literal t .

Every variable v is represented by a log-likelihood function

$$L_v(x) := \begin{cases} m & \text{if } x = x_v \text{ or } x = x_{\bar{v}} , \\ 0 & \text{otherwise} . \end{cases}$$

Each clause c is also represented by a log-likelihood function

$$L_c(x) := \begin{cases} 1 & \text{if } x = x_t \text{ and the literal } t \text{ appears in } c , \\ 0 & \text{otherwise} . \end{cases}$$

Finally we choose the penalty factor $p = 2m - 1$.

An assignment of the variables is represented by a classification with one class C_v for each variable v . The parameter value of C_v is either x_v or $x_{\bar{v}}$, representing the truth value of v . If the assignment is satisfying, this results in a classification $\mathcal{C}$ with

$$\text{IC}(\mathcal{C}) = -2(mn + m) + (2m - 1)n , \quad (5)$$

since each L_v is assigned to C_v , where it has log-likelihood m , and each L_c can be assigned to at least one x_t for a literal t appearing in c , so it has log-likelihood 1. There are n classes and the penalty factor is

$2m - 1$. Any other classification (either representing a non-satisfying assignment or not representing an assignment) has a higher IC, as we now show.

First, if a classification does not have a class at either x_v or $x_{\bar{v}}$ for some variable v , we can add a class at one of those. This increases the log-likelihood of L_v by m and adds one class. Thus, if no additional clauses are satisfied, the IC decreases by $2m - p = 2m - (2m - 1) = 1$. If this assignment satisfies previously unsatisfied clauses, the IC decreases even more.

Second, if a classification has classes at x_v and $x_{\bar{v}}$, the log-likelihood of L_v does not increase compared to having just one of those classes. Selecting both can satisfy at most $m - 1$ additional clauses over selecting one literal of v and $\bar{v}$, whichever appears in more clauses. For each of these clauses the log-likelihood of L_c increases by 1. Selecting both adds an extra class, so the IC increases by at least $-2(m - 1) + p = -2m + 2 + 2m - 1 = 1$ compared to selecting either v or $\bar{v}$.

Finally, if $\mathcal{C}$ represents a non-satisfying assignment, then

$$\text{IC}(\mathcal{C}) = -2(mn + m') + (2m - 1)n ,$$

where $m' < m$ is the number of satisfied clauses. This is larger than the IC for a satisfying assignment by $2(m - m')$.

Thus, there is a classification $\mathcal{C}$ with the value of the IC in Equation 5 if and only if F is satisfiable, proving NP-hardness.

Theorem 3.7. DISCRETE CLASSIFICATION and CONTINUOUS CLASSIFICATION are NP-hard if non-bitonic log-likelihood functions are allowed.

From now on, we again always assume that all log-likelihood functions are bitonic. We have seen in Section 3.1 that in this setting the discrete classification problem can be solved in polynomial time. We show next that the continuous classification problem remains NP-hard.

3.3 NP-hardness for Continuous Parameter with Bitonic Log-likelihood Functions

We show that the decision version of continuous classification is NP-hard by reduction from SET COVER. Let $\mathcal{S} = \{S_1, \dots, S_n\}$ be a family of subsets of a universe $\mathcal{U} = \{1, \dots, m\}$ such that $\bigcup_{i=1}^n S_i = \mathcal{U}$, and assume that an integer $k < n$ is given. Given this instance of SET COVER, we transform it into a set of log-likelihood functions, a penalty factor p and a constant c such that a classification $\mathcal{C}$ with $\text{IC}(\mathcal{C}) \leq c$ exists if and only if $\mathcal{S}$ has a cover of size $\leq k$.

We enumerate the 2^n subsets of $\mathcal{S}$ in the following way:

$$\sigma_i := \{S_l \mid \text{The } l\text{th bit in the binary representation of } i \text{ is set}\} .$$

Thus, each subset σ_i of $\mathcal{S}$ is represented by exactly one integer $i \in \{0, \dots, 2^n - 1\}$.

We construct a set of log-likelihood functions over the domain $\mathcal{D} := [0, 2^n)$, where each interval $(i, i + 1)$ (for $i \in \{0, \dots, 2^n - 1\}$) represents the subset σ_i . For each $j \in \mathcal{U}$, we define a pair of log-likelihood functions L_j^- and L_j^+ as follows:

$$L_j^-(x) = \begin{cases} \lceil -x \rceil & \text{if } j \in \bigcup_{S_i \in \sigma_{\lfloor x \rfloor}} S_i \\ -x & \text{otherwise,} \end{cases} \quad L_j^+(x) = \begin{cases} \lceil x \rceil & \text{if } j \in \bigcup_{S_i \in \sigma_{\lfloor x \rfloor}} S_i \\ x & \text{otherwise.} \end{cases}$$

Observe that L_j^- is nonincreasing over the whole domain, while L_j^+ is nondecreasing. See Fig. 4a for an example of these functions. Now, for non-integer values of x , $L_j^-(x) + L_j^+(x) = 1$ if j is covered by the set $\sigma_{\lfloor x \rfloor}$ that is represented in the current interval of the domain, and they sum to 0 otherwise. Thus,

$$L(i + \delta) := \sum_{j=1}^m \left(L_j^-(i + \delta) + L_j^+(i + \delta) \right) = \left| \bigcup_{S_i \in \sigma_i} S_i \right|$$

for any $\delta \in (0, 1)$, and σ_i covers $\mathcal{U}$ iff $L(i + \delta) = m$. So, there is a classification of $\mathcal{S} = \bigcup_{j=1}^m \{L_j^-, L_j^+\}$ with one class and log-likelihood $\geq m$ iff $\mathcal{S}$ contains a cover.

Figure 4: Example of the reduction on the SET COVER instance $\mathcal{U} = \{1, 2, 3, 4\}$, $\mathcal{S} = \{S_1 := \{1, 3, 4\}, S_2 := \{1, 2\}, S_3 := \{1, 4\}\}$. The covers are $\sigma_3 = \{S_1, S_2\}$ and $\sigma_7 = \mathcal{S}$, since all functions in (a) are off the diagonals in the intervals $(3, 4)$ and $(7, 8)$. (b): The sum of K^- and K^+ encodes the size of each σ_i . The smallest cover is σ_3 and indeed the optimum classification $\mathcal{C} = \{C_1\}$ has $x(C_1) \in (3, 4)$.

Now, we introduce a new pair of functions K^- and K^+ , again nonincreasing and nondecreasing, that together represent the size of a subset $\sigma_i \subseteq \mathcal{S}$. Let $b(x)$ represent the number of set bits in the binary representation of $\lfloor x \rfloor$.

$$K^-(x) := -\lfloor x \rfloor - \frac{b(x)}{2(n+1)}, \quad K^+(x) := \lfloor x \rfloor - \frac{b(x)}{2(n+1)}.$$

Then, since $0 \leq b(x) \leq n$ for any $x \in \mathcal{D}$, we have

$$-1 < K^-(x) + K^+(x) = -\frac{b(x)}{n+1} \leq 0.$$

The number of set bits $b(x)$ equals the size of the set $\sigma_{\lfloor x \rfloor}$ represented through x . An example of this construction is shown in Fig. 4b. The log-likelihood of the classification $\mathcal{C}_x$ consisting of a single class with parameter value x is

$$L(\mathcal{C}_x) := K^-(x) + K^+(x) + \sum_{j=1}^m \left(L_j^-(x) + L_j^+(x) \right)$$

and $L(\mathcal{C}_x) \geq m - \frac{k}{n+1}$ if and only if $\sigma_{\lfloor x \rfloor}$ is a cover of $\mathcal{U}$ of size $\leq k$.

Lemma 3.8. *The set of functions $\{K^-, K^+, L_1^-, \dots, L_m^-, L_1^+, \dots, L_m^+\}$ with a penalty factor of $p = 4m \cdot 2^n$ admits a classification with $\text{IC} \leq c := 4m \cdot 2^n - 2 \left(m - \frac{k}{n+1} \right)$ if and only if $\mathcal{S}$ contains a cover of $\mathcal{U}$ of size at most k .*

Proof. By setting $p = 4m \cdot 2^n$, the classification $\mathcal{C}_x$ consisting of a single class C_1 with $x(C_1) = x$ has

$$\text{IC}(\mathcal{C}_x) := |\mathcal{C}_x| \cdot p - 2L(\mathcal{C}_x) \leq 4m \cdot 2^n - 2 \left(m - \frac{k}{n+1} \right) = c$$

if and only if $\sigma_{\lfloor x \rfloor}$ is a solution to the set cover instance, i.e. a cover of size $\leq k$.

If we consider a classification $\mathcal{C}^2$ consisting of two classes C_1 and C_2 , the optimal choice is $x(C_1) = 0$, $C_1 = \{K^-, L_1^-, \dots, L_m^-\}$, $x(C_2) = 2^n - \delta$ and $C_2 = \{K^+, L_1^+, \dots, L_m^+\}$, since then all functions are assigned to a point in the domain where they reach their maximum log-likelihood. The log-likelihood of this classi-

fication is

$$\begin{aligned} L(\mathcal{C}^2) &= K^-(0) + K^+(2^n) + \sum_{j=1}^m \left(L_j^-(0) + L_j^+(2^n) \right) \\ &= 2^n - \frac{1+2\varepsilon}{2(n+1)} + \sum_{j=1}^m (0 + 2^n) \leq (m+1) \cdot 2^n . \end{aligned}$$

The IC is

$$2 \cdot p - 2L(\mathcal{C}^2) \geq 8m \cdot 2^n - 2(m+1) \cdot 2^n \geq 4m \cdot 2^n > c ,$$

for $m \geq 1$. Thus, no classification with two classes achieves an IC of at most c . Since $\mathcal{C}^2$ assigns every function to the point in its domain where it has the highest log-likelihood, the log-likelihood of any classification cannot be improved beyond $L(\mathcal{C}^2)$, and thus adding more than two classes only increases the IC through the penalty factor. $\square$ $\square$

Lemma 3.9. *All the functions described above, as well as p and c , can be constructed in polynomial time.*

Proof. We need to construct $2m+2$ functions. K^- and K^+ are fixed functions for a particular value of n . $K^-(x)$ and $K^+(x)$ can be evaluated for any $x \in \mathcal{D}$ in $O(n)$ time by a fixed algorithm.

Next we consider a function L_j^- , which can be described by n bits characterising which subsets $S_l \in \mathcal{S}$ contain j . These n bits can be extracted from $\mathcal{S}$ in time polynomial in n and m for any reasonable way to store $\mathcal{S}$. This representation of L_j^- suffices to compute $L_j^-(x)$ for any $x \in \mathcal{D}$ in $O(n)$ time. The same n -bit string acts as the description of L_j^+ . $\square$ $\square$

Theorem 3.10. CONTINUOUS CLASSIFICATION is NP-hard, even when restricted to bitonic log-likelihood functions.

Observe that this reduction results in an instance of CONTINUOUS CLASSIFICATION that can be discretised and solved by DISCRETE CLASSIFICATION. However, this discrete instance has 2^n candidate parameter values and thus DISCRETE CLASSIFICATION needs superpolynomial time to solve this instance.

The reduction does not preserve the approximation ratio, since k only influences the fractional part of the IC, while the IC can be exponentially large in n and linear in m . Thus, the solution with $x = 2^n - \delta$ that selects all input sets is an arbitrarily good approximation of the optimal classification for sufficiently large m and n .

3.4 Polynomial-time Algorithm for Realistic Inputs

We now present an algorithm for CONTINUOUS CLASSIFICATION that runs in polynomial time under mild assumptions on the input. For reasonably homogeneous trajectories (and a standard movement model) the log-likelihood functions are quite smooth. In this case the number of classifications we need to consider reduces; essentially, this is because the reordering illustrated in Fig. 2 does not happen too often.

First we describe the assumptions in more detail. As before we assume that we can optimise the value of the model parameter and compute the log-likelihood for a fixed class of size n efficiently in $F(n)$ time. As argued before, it is reasonable to assume that $F(n) = O(n)$. The remaining assumptions that we need to make are related to an arrangement $\mathcal{A}$ of curves which are derived from the input log-likelihood functions (and which we define later in this section). The running time of our algorithm is polynomial in the complexity of $\mathcal{A}$, provided a few basic operations on these curves can be performed efficiently. These operations are

- computing the intersections between pairs of curves, and
- computing the intersections of a curve and a vertical, horizontal or diagonal line segment.

We need the intersections with vertical and horizontal line segments to be able to restrict the arrangement to certain rectangles, and we need the (first and last) intersection with a diagonal line segment to compute a vertical decomposition (after rotation). Since these are operations on one or two log-likelihood functions,

Figure 5: A function $L^+ \in D^+$ by definition has $L^+(x_{h-1}^i) < L^+(x_h^i)$ and $L^+(x_{h-1}^{i+1}) \geq L^+(x_h^{i+1})$. The reverse holds for $L^- \in D^-$.

we assume that they can be performed in constant time. The complexity of $\mathcal{A}$ is polynomial in the input size if the pairwise number of intersections between curves is polynomially bounded. In our experiments (see later in this section) no pair of curves intersected more than once, and many did not intersect at all.

For a set of input functions $\mathcal{L} := \{L_1, \dots, L_k\}$ in increasing order of the parameter values $M_1, \dots, M_k$ at which they reach their maximum log-likelihood, let Opt_i denote the optimal classification with exactly i classes. Let C_j^i denote the j th class in Opt_i (sorted by parameter value).

The algorithm iteratively computes Opt_{i+1} from Opt_i , for $i \in \{1, \dots, k-1\}$. There is only one classification with one class (ignoring the parameter value), so $Opt_1 = \{C_1^1\} = \{\{L_1, \dots, L_k\}\}$. $x(C_1^1)$ can be computed in $O(F(k))$ time.

Let x_j^i be a shorthand notation for $x(C_j^i)$. Recall that a classification is stored in an array specifying for each function the parameter value of the class to which it is assigned. Recall that M_i is the smallest parameter value at which function L_i reaches its maximum log-likelihood.

If there are multiple classifications that achieve the maximum log-likelihood, we define the following tie breaking rule: The optimal classification is the one whose array representation is lexicographically smallest among the classifications with maximal log-likelihood. We show later that the algorithm described in this section computes this specific classification.

In constructing Opt_{i+1} , we use the following properties, where C_0^i and C_{i+1}^i are empty dummy classes:

Lemma 3.11. *Let $Opt_i = \{C_1^i, \dots, C_i^i\}$ and $Opt_{i+1} = \{C_1^{i+1}, \dots, C_{i+1}^{i+1}\}$ be the optimal classifications of a set of functions $\mathcal{L} := \{L_1, \dots, L_k\}$ with i and $i+1$ classes respectively. Let $x(C_0^i) := M_1$ and $x(C_{i+1}^i) := M_k$. Then, the j th class of Opt_{i+1} has a smaller parameter value than the j th class in Opt_i , but no smaller than the $j-1$ th class in Opt_i . That is,*

$$\forall j \in \{1, \dots, i+1\} : x_{j-1}^i \leq x_j^{i+1} \leq x_j^i .$$

Proof. Consider the second inequality in the lemma, $x_j^{i+1} \leq x_j^i$. Assume there exists a j such that $x_j^{i+1} > x_j^i$, and consider the smallest j for which this is true.

First, for the case $j = i+1$ we observe that $x_{i+1}^{i+1} \leq x_{i+1}^i = M_k$ is trivially true: Since all functions are decreasing for $x > M_k$, selecting $x_{i+1}^{i+1} > M_k$ can never result in a better log-likelihood than selecting $x_{i+1}^{i+1} = M_k$.

Thus, there must be an $h \in \{j+1, \dots, i+1\}$ such that $x_{h-1}^i < x_{h-1}^{i+1} \leq x_h^{i+1} \leq x_h^i$. Let

$$S^i := \bigcup_{m=1}^{h-1} C_m^i , \quad S^{i+1} := \bigcup_{m=1}^{h-1} C_m^{i+1}$$

and let $D^+ := S^{i+1} \setminus S^i$ and $D^- := S^i \setminus S^{i+1}$.

If $D^+ = D^- = \emptyset$, the union of the first $h-1$ classes of both classifications is identical, but Opt_i and Opt_{i+1} classify them differently, in particular the j th class. This implies that at least one of them is not optimal according to the tie breaking rules. Thus at least one of D^+ and D^- is nonempty.

Any function $L^+ \in D^+$ by definition has $L^+(x_{h-1}^i) < L^+(x_h^i)$ since it is not contained in S^i . It must also have $L^+(x_{h-1}^{i+1}) \geq L^+(x_h^{i+1})$ since it is in S^{i+1} . Because of the order of these four parameter values and the bitonicity of L^+ , it must fulfil

$$L^+(x_{h-1}^i) < L^+(x_h^i) \leq L^+(x_h^{i+1}) \leq L^+(x_{h-1}^{i+1}) .$$

Similarly, we have that for any $L^- \in D^-$

$$L^-(x_h^i) \leq L^-(x_{h-1}^i) \leq L^-(x_{h-1}^{i+1}) < L^-(x_h^{i+1}) .$$

This is illustrated in Fig. 5.

Let $L_{D^+}(x) := \sum_{L \in D^+} L(x)$ and $L_{D^-} := \sum_{L \in D^-} L(x)$. Since Opt_{i+1} is optimal, we get

$$\begin{aligned} \sum_{m=1}^{h-1} L_{C_m^{i+1}}(x_m^{i+1}) &\geq \sum_{m=1}^{h-1} L_{C_m^i}(x_m^i) + L_{D^+}(x_h^{i+1}) - L_{D^-}(x_{h-1}^i) , & \text{or} \\ \sum_{m=1}^{h-1} L_{C_m^{i+1}}(x_m^{i+1}) + L_{D^-}(x_{h-1}^i) &\geq \sum_{m=1}^{h-1} L_{C_m^i}(x_m^i) + L_{D^+}(x_h^{i+1}) . \end{aligned} \quad (6)$$

If these inequalities do not hold, we can construct a better classification with $i+1$ classes than Opt_{i+1} by combining the first $h-1$ classes of Opt_i with the remaining classes of Opt_{i+1} while ensuring that the log-likelihood functions in D^+ and D^- are assigned to exactly one class in the following way:

$$\mathcal{C}'_{i+1} := \{C_1^i, \dots, C_{h-2}^i, C_{h-1}^i \setminus D^-, C_h^{i+1} \cup D^+, C_{h+1}^{i+1}, \dots, C_{i+1}^{i+1}\} .$$

Similarly, by optimality of Opt_i , we get

$$\begin{aligned} \sum_{m=1}^{h-1} L_{C_m^i}(x_m^i) &\geq \sum_{m=1}^{h-1} L_{C_m^{i+1}}(x_m^{i+1}) + L_{D^-}(x_{h-1}^{i+1}) - L_{D^+}(x_h^i) , & \text{or} \\ \sum_{m=1}^{h-1} L_{C_m^i}(x_m^i) + L_{D^+}(x_h^i) &\geq \sum_{m=1}^{h-1} L_{C_m^{i+1}}(x_m^{i+1}) + L_{D^-}(x_{h-1}^{i+1}) , \end{aligned} \quad (7)$$

since otherwise the classification

$$\mathcal{C}'_i := \{C_1^{i+1}, \dots, C_{h-2}^{i+1}, C_{h-1}^{i+1} \cup D^-, C_h^i \setminus D^+, C_{h+1}^i, \dots, C_i^i\}$$

would be a better classification with i classes than Opt_i .

Combining (6) and (7) with $L_{D^+}(x_h^{i+1}) \geq L_{D^+}(x_h^i)$, we get $L_{D^-}(x_{h-1}^i) \geq L_{D^-}(x_{h-1}^{i+1})$. However, we know that $L_{D^-}(x_{h-1}^i) \leq L_{D^-}(x_{h-1}^{i+1})$, establishing that the inequalities in (6) and (7) are actually equalities. Thus $\mathcal{C}'_i$ has the same log-likelihood as Opt_i and $\mathcal{C}'_{i+1}$ has the same log-likelihood as Opt_{i+1} and neither $\mathcal{C}'_i$ nor $\mathcal{C}'_{i+1}$ breaks Observation 1. However, Opt_i and Opt_{i+1} cannot both have a lexicographically smaller representation than $\mathcal{C}'_i$ and $\mathcal{C}'_{i+1}$ respectively. Thus at least one of Opt_i and Opt_{i+1} is not optimal according to the tie breaking rule, which is a contradiction.

Thus any allowed value of j leads to a contradiction, showing that no j can violate the second inequality in the statement of the lemma. For the first inequality in the lemma, $x_{j-1}^i \leq x_j^{i+1}$, we observe that this concerns the class with the $(i-j+2\text{nd})$ -largest parameter value for both classifications and we can apply similar reasoning to show that this holds for all j , thus completing the proof. $\square$ $\square$

Lemma 3.12. *Functions shift to a class with a larger index as the number of classes increases, but no more than one class at a time. That is,*

$$\forall j \in \{1, \dots, k\} : L_j \in C_m^i \Rightarrow L_j \in C_m^{i+1} \cup C_{m+1}^{i+1} .$$

Proof. Consider a function $L_l \in C_m^i$. Then, $M_l \in (x_{m-1}^i, x_{m+1}^i)$. Furthermore, we have $L_l(x_m^i) > L_l(x_{m-1}^i)$ and $L_l(x_m^i) \geq L_l(x_{m+1}^i)$.

If $M_l \geq x_m^{i+1}$, then L_l cannot be assigned to a class C_p^{i+1} with $p \leq m-1$ by Observation 1.

If $M_l < x_m^{i+1}$, we have $L_l(x_m^{i+1}) \geq L_l(x_m^i) > L_l(x_{m-1}^i) \geq L_l(x_{m-1}^{i+1})$ for any $p \leq m-1$ by Lemma 3.11 (see also L^+ in Fig. 5) and thus $L_l \notin C_p^{i+1}$. Similarly, $L_l \notin C_p^{i+1}$ for any $p > m+1$ and thus $L_l \in C_m^{i+1}$ or $L_l \in C_{m+1}^{i+1}$. $\square$ $\square$

Figure 6: A pair of log-likelihood functions L_1, L_2 (a) and the arrangement $\mathcal{A}$ induced by the curves P_1, P_2 (b). Both functions have $L(p) = L(q)$, and thus the arrangement has a vertex at (p, q) . $\mathcal{A}$ has 5 faces, with the shaded faces corresponding to the same classification $C_1^2 = \{L_1\}$, $C_2^2 = \{L_2\}$.

When Opt_i is computed, we can use it to compute Opt_{i+1} . By Lemmas 3.11 and 3.12, a function can be in one of only two classes of Opt_{i+1} and the parameter value x_j^{i+1} for a class C_j^{i+1} must be in a particular interval $[x_{j-1}^i, x_j^i]$.

For a function $L_l \in C_j^i$, whether $L_l \in C_j^{i+1}$ or $L_l \in C_{j+1}^{i+1}$ is determined by the parameter values x_j^{i+1} and x_{j+1}^{i+1} selected for these classes. The parameter values for two consecutive classes can be represented by a point inside a rectangle $\mathcal{R}_j^i \subset \mathcal{D}^2$:

$$(x_j^{i+1}, x_{j+1}^{i+1}) \in [x_{j-1}^i, x_j^i] \times [x_j^i, x_{j+1}^i] =: \mathcal{R}_j^i.$$

Let $P_l := \{(p, q) \in \mathcal{D}^2 \mid L_l(p) = L_l(q) \wedge p < q\}$ be the set of points that represent pairs of parameter values at which L_l has the same value. If L_l is continuous and strictly bitonic (i.e. strictly increasing and decreasing), P_l is a bimonotone continuous curve that separates the region $\{(x, y) \in \mathcal{D}^2 \mid x < y\}$ into two parts. For any point (p, q) in the region below P_l , we have $L_l(p) < L_l(q)$. Thus, if the two candidate classes for L_l have parameter values x_j^{i+1} and x_{j+1}^{i+1} , and P_l lies below $(x_j^{i+1}, x_{j+1}^{i+1})$ or contains this point, then L_l is in C_j^{i+1} rather than C_{j+1}^{i+1} . Similarly, if P_l lies above $(x_j^{i+1}, x_{j+1}^{i+1})$, then L_l is in C_{j+1}^{i+1} . These curves are illustrated in Fig. 6.

If L_l is not strictly bitonic or discontinuous at a finite number of points, P_l is not a well-defined Jordan curve. In this case, we relax the definition of P_l to be the continuous, bimonotone curve separating the regions with $L_l(p) < L_l(q)$ from those with $L_l(p) \geq L_l(q)$ (note that the latter inequality is no longer strict). This curve always exists and suffices to find the optimal classification.

If the intersection of P_l with $\mathcal{R}_j^i$ for some $L_l \in C_j^i$ is empty, L_l is assigned to a fixed class in Opt_{i+1} regardless of the choice of parameter values. Otherwise, P_l divides $\mathcal{R}_j^i$ into two contiguous regions: one in which $L_l \in C_j^{i+1}$ and one in which $L_l \in C_{j+1}^{i+1}$. We first show how to compute Opt_2 from Opt_1 using these curves, then how to generalise this to arbitrary i .

3.4.1 Computing Opt_2 .

The set $\{P_1, P_2, \dots, P_k\}$ defines an arrangement of curves $\mathcal{A}$. This is illustrated in Fig. 6. Let $\mathcal{A}_j^i$ be the part of $\mathcal{A}$ inside $\mathcal{R}_j^i$. Each face of $\mathcal{A}_j^i$ defines a (not necessarily unique) partition of $C_1^1 = \{L_1, L_2, \dots, L_k\}$ into two classes, fixing the structure of the classification.

Lemma 3.13. *One of the faces f in $\mathcal{A}_1^1$ represents Opt_2 , in the sense that $C_1^2 = \{L_l \mid P_l \text{ lies below } f\}$ and $C_2^2 = \{L_l \mid P_l \text{ lies above } f\}$.*

Proof. By Lemma 3.11 and the definition of $\mathcal{R}_1^1$, we know that $p := (x_1^2, x_2^2) \in \mathcal{R}_1^1$. Let f be the face of $\mathcal{A}_1^1$ that contains p . If Opt_2 is not the classification represented by f there is a function L_l such that $L_l \in C_1^2$, but p lies below P_l or vice versa. However, if p is below P_l , $L_l(x_1^2) < L_l(x_2^2)$ and the classification is improved by assigning L_l to C_2^2 , contradicting the optimality of Opt_2 . Thus, the classification represented by f is precisely Opt_2 . $\square$

Figure 7: Illustration of the algorithm for a set of curves. The coloured path through u and v represents the coloured faces $f_u \subseteq \mathcal{R}_1^2$ and $f_v \subseteq \mathcal{R}_2^2$, yielding a candidate for Opt_3 with $C_1 = \{L_1, L_4\}$, $C_2 = \{L_2, L_3, L_5, L_6\}$ and $C_3 = \{L_7, L_8\}$.

For each face, the parameter values of the two classes are optimised separately and the resulting classification with the highest log-likelihood is selected as Opt_2 .

3.4.2 Computing Opt_i ($i > 2$).

In computing Opt_3 from Opt_2 (or generally Opt_{i+1} from Opt_i for $i \geq 2$), we have to deal with the fact that both $\mathcal{A}_1^1$ and $\mathcal{A}_2^2$ produce candidates for subsets of C_3^3 . We could consider each pair of faces $(f_1, f_2) \in \mathcal{A}_1^1 \times \mathcal{A}_2^2$ and optimise parameter value for the three classes represented by f_1 and f_2 , but that does not generalise to arbitrary i , since we would have to consider every i -tuple of faces in $\mathcal{A}_1^1 \times \mathcal{A}_2^2 \times \dots \times \mathcal{A}_i^i$ and the number of such i -tuples is exponential in i .

Instead, the optimal classification can be computed by computing a longest path on a polynomial-size directed acyclic graph G_i . The vertices of G_i are organised in $i + 2$ levels, with a source vertex r at level 0 and a target vertex t at level $i + 1$. For each face in $\mathcal{A}_j^j$, G_i has a vertex at level j representing it. A vertex v at level j , or face in $\mathcal{A}_j^j$, corresponds to a partition of C_j^j into two parts λ_v and ρ_v , which are candidates to become part of C_j^{i+1} and C_{j+1}^{i+1} respectively. For each $j \in \{0, \dots, i\}$, G_i contains an edge from each vertex at level j to each vertex at level $j + 1$. See Fig. 7 for an illustration of this construction.

Consider two vertices u at level $j - 1$ and v at level j . u represents a partition of C_{j-1}^{i+1} into sets λ_u and ρ_u of candidate functions for C_{j-1}^{i+1} and C_j^{i+1} respectively. The partition is defined by the face in $\mathcal{A}_{j-1}^{i+1}$ represented by u . Similarly, λ_v and ρ_v are candidate sets to be part of C_j^{i+1} and C_{j+1}^{i+1} . Thus, the edge (u, v) fixes a set of functions $\rho_u \cup \lambda_v$ that is a candidate for C_j^{i+1} . Here, $\rho_r = \lambda_t = \emptyset$, i.e. r and t do not contribute to C_1^{i+1} and C_{i+1}^{i+1} respectively. For the candidate class represented by an edge, the parameter value is optimised and its log-likelihood is set as the edge weight.

An r - t path in G_i visits exactly one vertex at each level, and this path represents a particular classification, with each edge (u, v) defining a class $\rho_u \cup \lambda_v$. The log-likelihood of the classification is the sum of the log-likelihoods of the individual classes –represented by the weights of the edges in the path–, i.e. the path’s length.

Lemma 3.14. *Opt_{i+1} corresponds to the longest r - t path in G_i .*

Proof. We have already shown above that the length of an r - t path equals the log-likelihood of the corresponding classification and thus Opt_{i+1} cannot correspond to a path in G_i that is not the longest. What remains to be shown is that there are no classifications with a larger log-likelihood than the one represented by the longest path in G_i .

By Lemma 3.11 and the definition of $\mathcal{R}_j^i$, we know that $p_j := (x_j^{i+1}, x_{j+1}^{i+1}) \in \mathcal{R}_j^i$ for all $j \in \{1, \dots, i\}$. So, all p_j lie inside a face f_j represented by a vertex v_j at level j of G_i . Consider the path $(r, v_1, v_2, \dots, v_i, t)$ through these vertices. If Opt_{i+1} is not the classification represented by this path, then there is a function $L_l \in \lambda_{v_j}$ (resp. ρ_{v_j}) that is assigned to C_{j+1}^{i+1} (C_j^{i+1}), and the classification is improved by assigning L_l to C_j^{i+1} (C_{j+1}^{i+1}).

Thus, Opt_{i+1} is represented by a path in G_i , and since the length of a path equals the log-likelihood of its corresponding classification, it must be the longest path in G_i . $\square$ $\square$

The use of $\mathcal{A}$ and G_i is illustrated in Fig. 7. For the indicated value of x_1^1 , the grey rectangle is $\mathcal{R}_1^1$. Suppose that the parameter values for Opt_2 are found to be (x_1^2, x_2^2) at the big red dot. Then $C_1^2 = \{L_1, L_2, L_4, L_5\}$ and $C_2^2 = \{L_3, L_6, L_7, L_8\}$, i.e. the curves lying below and above this point. The red rectangles are $\mathcal{R}_1^2$ and $\mathcal{R}_2^2$ and the graph G_2 has six vertices at level 1 corresponding to the six faces of $\mathcal{A}_1^2$ and five vertices at level 2. A path through G_2 corresponds to selecting one face in each of the rectangles, thus fixing the classification.

3.4.3 Analysis.

$\mathcal{A}$ has $O(k\lambda_{s+2}(k))$ faces, where s is the maximal number of intersections between a pair of curves in $\mathcal{A}$ and $\lambda_s(n)$ is the maximum length of an (n, s) Davenport-Schinzel sequence [10]. Thus, a trivial bound on the complexity of G_i is obtained by observing that each level of G_i has at most $O(k\lambda_{s+2}(k))$ vertices and thus the number of vertices in G_i is $O(k^2\lambda_{s+2}(k))$. A better bound on the number of vertices of G_i can be obtained by observing that all faces of $\mathcal{A}$ are both x -monotone and y -monotone, since the edges bounding them are also bimonotone.

Lemma 3.15. G_i has $O(k\lambda_{s+2}(k))$ vertices and $O((k\lambda_{s+2}(k))^2)$ edges.

Proof. The rectangles $\mathcal{R}_j^i$ and $\mathcal{R}_{j+1}^i$ overlap only in a single point (x_j^i, x_{j+1}^i) . A face that does not contain such a point overlaps at most one rectangle $\mathcal{R}_j^i$ and thus contributes at most one vertex to G_i . Since there are $i-1$ of these points where two rectangles touch, at most $i-1$ additional points are contributed by faces that overlap more than one rectangle. Thus, the number of vertices in G_i is at most $2 + |\mathcal{A}| + i - 1 = O(k\lambda_{s+2}(k) + i) = O(k\lambda_{s+2}(k))$.

The bound on the number of edges follows immediately. $\square$ $\square$

The bound on the number of edges is tight, in the sense that if G_i contains $|V|$ vertices, it may have $\Theta(|V|^2)$ edges, namely if almost all faces intersect one of two consecutive rectangles.

If there are multiple optimal classifications with different combinatorial structures, they will each be found on their respective paths through G_i . Checking whether it is lexicographically smaller than the best found so far can be done in $O(k)$ time. If a set of functions reaches its maximum log-likelihood at multiple parameter values, we assume that the optimisation procedure can select the smallest of those. Thus, the algorithm can break ties between optimal classifications without affecting the running time, so the assumption that the algorithm computes this particular classification is justified.

Edelsbrunner *et al.* [10] give an algorithm for computing curve arrangements. It requires that the following operations can be performed efficiently on the curves:

1. Find the intersection points of two curves.
2. Given a vertical line segment, find its highest and lowest intersections with a curve.
3. Determine the relative order of two points on a curve using some parameterisation.
4. Given a curve, find its points of vertical tangency.

Data set	Trajectories	Number of intersections
Fisher (segmented by BIC)	929	87,113
Gulls	98	0
Vervet monkeys (daily trajectories)	462	1,709

Table 2: Results of experiments on number of intersections in $\mathcal{A}$ in practice.

Using a rotated coordinate system with the x -axis parallel to the line $y = -x$ and the y -axis parallel to $y = x$, all curves are strictly x -monotone without any points of “vertical” (parallel to the rotated y -axis) tangency. Thus 3. and 4. become trivial and every curve has only one intersection with any “vertical” line segment. If these operations can be done efficiently, we can also decide efficiently whether a particular point lies above or below a specific curve. The curves P_i and the arrangement $\mathcal{A}$ can be computed in $O(C \cdot k\lambda_{s+2}(k))$ time [10], where C is a constant depending on the input functions which represents the cost of finding intersections between two curves or between a curve and a vertical line segment. This will generally be a constant for a fixed movement model, since they are operations on 1 or 2 log-likelihood functions.

Then, to compute Opt_{i+1} from Opt_i we need to compute $\mathcal{A}_1^i, \dots, \mathcal{A}_i^i$, i.e. the intersection of $\mathcal{A}$ with the rectangles $\mathcal{R}_1^i, \dots, \mathcal{R}_i^i$. In particular, we need to know which faces of $\mathcal{A}$ intersect each rectangle to construct G_i . This step can be performed in $O(C \cdot k\lambda_{s+2}(k))$ time using a depth-first search on the dual graph of $\mathcal{A}$.

Computing the length of an edge in G_i is assumed to take $O(F(n))$ time, where n is the number of trajectories in the class represented by the edge. Then, computing all edge lengths takes $O((k\lambda_{s+2}(k))^2 \cdot F(k))$ time. Since G_i is a DAG, the longest path can be computed in $O(|V| + |E|) = O((k\lambda_{s+2}(k))^2)$ time.

Thus, given Opt_i and $\mathcal{A}$, computing Opt_{i+1} takes $O((k\lambda_{s+2}(k))^2 \cdot F(k))$ time. To find the optimal classification, we have to compute all classifications up to Opt_k and select the one among those with minimum IC for the given penalty factor.

Theorem 3.16. *An optimal classification of k trajectories can be computed in $O(k^3\lambda_{s+2}(k)^2 \cdot F(k) + C \cdot k\lambda_{s+2}(k))$ time, where $F(k)$ is the time required to optimise the log-likelihood for a set of k trajectories, C is a constant that depends on the input functions and s is the maximum number of pairwise intersections between the curves P_i .*

For the algorithm to run in polynomial time, we need to be able to efficiently compute the arrangement $\mathcal{A}$, and to optimise the log-likelihood for a set of trajectories (expressed as $F(n)$). Furthermore, the complexity of the algorithm depends on the complexity of $\mathcal{A}$ and the graphs G_i , which are polynomial if s is polynomially bounded. The time required to compute $\mathcal{A}$ also depends on the cost of computing intersections, which differs by movement model.

The maximum value of s depends on the movement model and can theoretically be unbounded. We conjecture that for the Brownian bridge movement model, $s = \Theta(n)$ in theory, although it is much better than that in practice. We computed the curves using the Brownian bridge movement model for several data sets, collected from fishers [15, 16], gulls [20] and vervet monkeys [6], and counted the number of intersections. The results are summarized in Table 2. We did not observe a pair of curves intersecting more than once in these experiments. The total number of intersections is much smaller than $\binom{n}{2}$, indicating many pairs do not intersect at all. The exact fraction of intersecting pairs of curves highly depends on the particular data set.

4 Discussion

We presented efficient algorithms for optimally segmenting and classifying trajectories using a parameterised movement model with respect to an information criterion. The result of the algorithms depends on the exact information criterion used, which is parameterised by the penalty factor p .

We discussed the algorithms for movement models with only a single parameter, and now briefly consider how to deal with models which have multiple parameters. At a break point, any subset of the variables

Figure 8: A fisher trajectory with the optimal segmentation indicated by colour.

might change while the others remain unchanged. This is reflected in the model complexity for the IC, where only the changed variables count towards the penalty factor. For discrete segmentation, consider a model with d variables $x_1, \dots, x_d$, which are drawn from sets of candidate parameter values $X_1, \dots, X_d$ respectively. For every cell $O_{i,x_1,\dots,x_d}$ of the dynamic programming table, 2^d candidates have to be considered, for the 2^d subsets of variables that may change there. Let $m = \prod_{i=1}^d |X_d|$ be the total number of possible combinations of parameter values. Then the optimal discrete segmentation can be computed in $O(2^d nm)$ time. For continuous segmentation, the problem is that there will be dependencies between different segments when only some of the variables change, and thus it will be hard to optimise all variables. In the other cases it is not obvious how to generalise the algorithms we presented to models with multiple parameters. With classification, Observation 1 is crucial, and it does not generalise to higher dimensions.

The quality of segmentations and classifications can be defined by other measures than the information criterion we used for our methods. The correctness of our algorithms relies on specific properties of these information criteria, so it is not trivial to generalise these methods to other quality measures, although some variations could possibly be integrated. However, we note that the framework we used offers quite a bit of freedom in selecting appropriate log-likelihood functions and penalty factors, thus allowing these methods to be used in a wide range of applications.

Our algorithms are strongly motivated by applications in movement ecology, and thus an extensive evaluation of these methods in an application context is highly important, but beyond the scope of this paper. We therefore just present an example segmentation using the Brownian bridge movement model. The trajectory shown in Fig. 8 is of (the animal) fisher [15, 16] and is segmented optimally with respect to BIC. We also tested our methods on trajectories obtained by a sequence of subtrajectories with different random generators. Generally, the segmentation algorithm detects the original subtrajectories up to possibly one data point on the boundary of the segments.

References

- [1] Alewijnse, S.P.A., Buchin, K., Buchin, M., Kölsch, A., Kruckenberg, H., Westenberg, M.A.: A framework for trajectory segmentation by stable criteria. In: Proc. 22nd Internat. Conf. Advances in Geographic Information Systems (ACM GIS), pp. 351–360. ACM (2014)
- [2] Aronov, B., Driemel, A., Kreveld, M.V., Löffler, M., Staals, F.: Segmentation of trajectories on nonmonotone criteria. *ACM Transactions on Algorithms (TALG)* **12**(2), 26 (2016)
- [3] Benhamou, S.: Dynamic approach to space and habitat use based on biased random bridges. *PloS one* **6**(1), e14,592 (2011)
- [4] Buchin, K., Buchin, M., Gudmundsson, J., Löffler, M., Luo, J.: Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry and Applications*, special

issue on 19th International Symposium on Algorithms and Computation (ISAAC) **21**(3), 253–282 (2011). DOI 10.1142/S0218195911003652

- [5] Buchin, K., Sijben, S., Arseneau, T.J., Willems, E.P.: Detecting movement patterns using Brownian bridges. In: Proc. 20th Internat. Conf. Advances in Geographic Information Systems (ACM GIS), pp. 119–128 (2012)
- [6] Buchin, K., Sijben, S., van Loon, E.E., Sapir, N., Mercier, S., Marie Arseneau, T.J., Willems, E.P.: Deriving movement properties and the effect of the environment from the brownian bridge movement model in monkeys and birds. *Movement Ecology* **3**(1), 18 (2015). DOI 10.1186/s40462-015-0043-8
- [7] Buchin, M., Driemel, A., van Kreveld, M., Sacristán, V.: An algorithmic framework for segmenting trajectories based on spatio-temporal criteria. In: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 202–211. ACM (2010)
- [8] Buchin, M., Kruckenberg, H., Kölzsch, A.: Segmenting Trajectories by Movement States, pp. 15–25. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). DOI 10.1007/978-3-642-32316-4_2
- [9] Bullard, F.: Estimating the home range of an animal: A Brownian bridge approach. Master’s thesis, The University of North Carolina (1999)
- [10] Edelsbrunner, H., Guibas, L., Pach, J., Pollack, R., Seidel, R., Sharir, M.: Arrangements of curves in the plane – topology, combinatorics, and algorithms. *Theoretical Computer Science* **92**(2), 319 – 336 (1992). DOI [http://dx.doi.org/10.1016/0304-3975\(92\)90319-B](http://dx.doi.org/10.1016/0304-3975(92)90319-B)
- [11] Gurarie, E., Andrews, R.D., Laidre, K.L.: A novel method for identifying behavioural changes in animal movement data. *Ecology Letters* **12**(5), 395–408 (2009). DOI 10.1111/j.1461-0248.2009.01293.x
- [12] Horne, J., Garton, E., Krone, S., Lewis, J.: Analyzing animal movements using Brownian bridges. *Ecology* **88**(9), 2354–2363 (2007)
- [13] de Jager, M., Weissing, F.J., Herman, P.M.J., Nolet, B.A., van de Koppel, J.: Lévy walks evolve through interaction between movement and environmental complexity. *Science* **332**, 1551–1553 (2011)
- [14] Kranstauber, B., Kays, R., LaPoint, S., Wikelski, M., Safi, K.: A dynamic Brownian bridge movement model to estimate utilization distributions for heterogeneous animal movement. *Journal of Animal Ecology* **81**(4), 738–746 (2012)
- [15] LaPoint, S., Gallery, P., Wikelski, M., Kays, R.: Animal behavior, cost-based corridor models, and real corridors. *Landscape Ecology* **28**(8), 1615–1630 (2013). DOI 10.1007/s10980-013-9910-0
- [16] LaPoint, S., Gallery, P., Wikelski, M., Kays, R.: Data from: Animal behavior, cost-based corridor models, and real corridors. Movebank Data Repository (2013). DOI 10.5441/001/1.2tp2j43g
- [17] Nathan, R., Spiegel, O., Fortmann-Roe, S., Harel, R., Wikelski, M., Getz, W.M.: Using tri-axial acceleration data to identify behavioral modes of free-ranging animals: general concepts and tools illustrated for griffon vultures. *The Journal of Experimental Biology* **215**(6), 986–996 (2012)
- [18] Sankararaman, S., Agarwal, P.K., Mølhave, T., Pan, J., Boedihardjo, A.: Model-driven matching and segmentation of trajectories. In: Proc. 21st Internat. Conf. Advances in Geographic Information Systems (ACM GIS), pp. 234–243. ACM (2013)
- [19] Shamoun-Baranes, J., Bom, R., van Loon, E.E., Ens, B.J., Oosterbeek, K., Bouten, W.: From sensor data to animal behaviour: An oystercatcher example. *PLoS ONE* **7**(5), e37,997 (2012). DOI 10.1371/journal.pone.0037997

- [20] Stienen, E.W., Desmet, P., Aelterman, B., Courtens, W., Feys, S., Vanermen, N., Verstraete, H., de Walle, M.V., Deneudt, K., Hernandez, F., Houthoofd, R., Vanhoorne, B., Bouten, W., Buijs, R.J., Kavelaars, M.M., Miller, W., Herman, D., Matheve, H., Sotillo, A., Lens, L.: Gps tracking data of lesser black-backed gulls and herring gulls breeding at the southern north sea coast. *ZooKeys* **555**, 115–124 (2016). DOI 10.3897/zookeys.555.6173