Straight-path queries in trajectory data

Journal of Discrete Algorithms 36 (2016) 27–38

Contents lists available at ScienceDirect

Journal of Discrete Algorithms

www.elsevier.com/locate/jda

Straight-path queries in trajectory data ☆

Mark de Berg [1], Ali D. Mehrabi *,[1]

Department of Computer Science, TU Eindhoven, Netherlands

ARTICLE INFO

Article history:
Available online 3 September 2015

Keywords:
Trajectory data
Data structures
Path simplification problem

ABSTRACT

Inspired by sports analysis, we study data structures for storing a trajectory representing the movement of a player during a game, such that the following queries can be answered: Given two positions s and t, report all sub-trajectories in which the player moved in a more or less straight line from s to t. We consider two measures of straightness, namely *dilation* and *direction deviation*, we present efficient construction algorithms for our data structures, and we analyze their performance. We also present an $O(n^{1.5+\varepsilon})$ algorithm for the following simplification problem: given a trajectory P and a threshold τ, find a simplification of P with a minimum number of vertices such that each edge in the simplification replaces a sub-trajectory whose length is at most τ times the length of the edge. This significantly improves the fastest known algorithm for the problem.

1. Introduction

Background. Video analysis is nowadays an important tool for sports coaches. Traditionally, video analysis is done manually: someone watches a video of a match and annotates the video with various types of events—goals or points being scored, changes of ball possession, and so on. However, manual analysis is labor intensive and annotating all league matches of an entire season would be very time-consuming and expensive. Therefore there has been considerable interest in automating parts of the process. A basic step in an automated analysis is to extract the movements of the players and the ball from the video. Nowadays this can be done quite accurately, giving us for each player a *trajectory*: a sequence of the player's location at regular time steps. The sampling rate of these trajectories is high: the data set of soccer trajectories made available by Petterson et al. [9], for instance, reports the positions at a frequency of 20 Hz. The availability of high-quality trajectories enables the use of geometric algorithms and data structures. In this paper we study two problems in this area.

The first problem is an indexing problem, related to the following query a coach may wish to ask: show me all video fragments in which player X runs in a more or less straight line from a certain position s on the field to another position t. We thus need a data structure storing a collection of trajectories (corresponding to the movements of player X in all matches) such that we can efficiently answer *straight-path queries*: given a directed query segment st, report all subtrajectories starting near s and going in a more or less straight line to a point near t. (We will define the problem more formally below.)

The second problem we study is a simplification problem: given a trajectory P, compute a simplification P' with a minimum number of vertices under the condition that P' is sufficiently similar to P. Here we require (as is usually done)

☆ A preliminary version appeared in the 9th Workshop on Algorithms and Computation (WALCOM) 2015.

* Corresponding author.
E-mail address: amehrabi@win.tue.nl (A.D. Mehrabi).

[1] M. de Berg and A.D. Mehrabi were supported by the Netherlands Organization for Scientific Research (NWO) under grants 024.002.003 and 612.001.118, respectively.

http://dx.doi.org/10.1016/j.jda.2015.08.002
1570-8667/

that the vertices of P' form a subset of the vertices of P—it is not allowed to introduce vertices at new locations. Computing such a simplification is useful to reduce storage requirements, and also to smooth out irregularities in the data due to small errors in the reported locations.

Straight-path queries: related work. The focus of our work is on data structures that come with proven guarantees on the query time but also on the quality of the reported results. For the latter we need to define when a subtrajectory is sufficiently similar to the query segment st. We are aware of only one such result, obtained by De Berg et al. [3]. They show how to store a trajectory P of n vertices such that, given a query segment st and a threshold Δ, one can find all subtrajectories of P whose so-called *Fréchet distance* to st is at most Δ. However, their work has several drawbacks. First of all, in addition to all the correct subtrajectories their data structure may report additional subtrajectories whose Fréchet distance to st can be up to a factor $2+3\sqrt{2}$ times larger than Δ. Second, their data structure is a complicated multi-level structure which is difficult to implement and unlikely to be efficient in practice. Finally, they only show how to (approximately) count the subtrajectories—it is unclear how to actually report them in an efficient manner. Gudmundsson and Smid [7] recently studied a more general version of the problem, where the data structure stores a geometric tree instead of a path and the query is a path, but their solution works for c-packed paths and only reports a single subpath (and is rather involved).

There are also several non-algorithmic papers about automating the analysis of sports videos; see e.g. the survey by Yu and Farin [13]. The paper by Shim et al. [11] is the one most closely related to our work. Given a video database and a query trajectory, Shim et al. study the problem of finding all video fragments whose trajectory is similar to the query trajectory. They first use spatio-temporal representation schemes to model the trajectories and then apply a k-warping distance algorithm to measure the similarity between the query trajectory and the trajectories of the moving objects. In contrast to what we study in this paper, they do not focus on the running time of their algorithm nor the amount of storage it needs.

Straight-path queries: our approach and results. We take the following practical approach. We partition the soccer field into a grid of square cells (the cell size can be set by the user). To specify a query the coach indicates a starting cell C_s and a target cell C_t, and the data structure should report all subtrajectories where the player moved in a more or less straight line from C_s to C_t. We still have to define what it means when "a player moves from C_s to C_t in a more or less straight line". Let s be the point where the player's trajectory P exits C_s and let t be the point where it enters C_t. Then we want the subtrajectory from s to t—we denote this subtrajectory by $P[s,t]$—to be similar to the segment st. We study two different definitions for this similarity.

- The first option is to use the so-called *dilation* of $P[s,t]$, which is defined as $|P[s,t]|/|st|$, where $|\cdot|$ denotes the Euclidean length of a path or segment. We say that the player moves in a more or less straight line from C_s to C_t when the dilation of $P[s,t]$ is at most some (predetermined) threshold $\tau \geqslant 1$. In other words, $P[s,t]$ can be at most a factor τ longer than the segment st.
- The second option is to require that the player always moves in more or less the same direction along $P[s,t]$. We define the *direction deviation* of a trajectory to be the maximum angle between any two (directed) segments on the trajectory. We then say that the player moves in a more or less straight line from C_s to C_t when the direction deviation of $P[s,t]$ is at most some (predetermined) threshold $\alpha < \pi/2$. We call such a subtrajectory *α-straight*.

Our first data structure for straight-path queries is a look-up table that stores, for all pairs of cells C_s, C_t in the grid, the set $S(C_s, C_t)$ of straight C_s-to-C_t subtrajectories (according the chosen definition of straightness). This means that a query can be answered in $O(1+A)$ time by a look-up table, where A is the number of reported subtrajectories. Our contributions for this simple data structure are (i) efficient algorithms to compute all sets $S(C_s, C_t)$, (ii) a theoretical analysis of the worst-case size of the data structures, and (iii) an experimental evaluation of the size of the data structures in practice.

Because the worst-case size of our first data structure is large, we also present a data structure that uses much less storage. This data structure can be used when the straightness measure is the direction deviation. A drawback is that, in addition to the correct subtrajectories, the data structure may also report some additional α-straight subtrajectories that start near C_s. We analyze the maximum possible error—that is, how far from C_s the reported subtrajectories may start—both theoretically and experimentally.

The minimum-vertex path-simplification problem. In path-simplification problems the goal is to compute, for a given trajectory P, a trajectory Q with fewer vertices than P that is sufficiently similar to P. The Douglas–Peuker algorithm [5,10] is probably the best known and most widely used simplification algorithm. There are many different variants of the path-simplification problem, which differ in the similarity measure used, in what is being optimized (one can either minimize the number of vertices of P' under the condition that its similarity to P stays within a given bound, or one can optimize the similarity under the condition that the number of vertices does not exceed a given number), and whether or not the simplification can only use vertices from P. We study a variant called the *minimum-vertex path-simplification (MVPS) problem*, introduced by Gudmundsson et al. [6]. In the MVPS problem we want to find a minimum-size subset Q of vertices of P such that for any two consecutive vertices p_i and p_j in Q we have $|P[p_i, p_j]| \leqslant \tau |p_i p_j|$. (In other words, we are only allowed to use a shortcut $p_i p_j$ when the dilation of $P[p_i, p_j]$ is at most τ.) Gudmundsson et al. [6] reduce the MVPS problem to finding a shortest path from p_1 to p_n in an associated graph G_τ. Their algorithm has quadratic running time

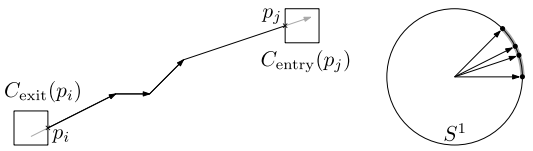

Fig. 1. A subtrajectory and the points on S^1 corresponding to its edges. The smallest circular interval containing the points is shown in grey.

as G_τ can have $\Theta(n^2)$ size. They also present an approximation algorithm that runs in $O(n \log n + n/\delta)$ time and computes a simplification whose size is within a factor of $(1+\delta)$ of the size of optimal solution, for any given $\delta > 0$. We present a dynamic-programming algorithm that computes an optimal solution in $O(n^{1.5+\varepsilon})$ expected time, for any $\varepsilon > 0$, thus significantly improving their quadratic algorithm.

2. The data structures

For simplicity of presentation we assume we are given a single trajectory P with n vertices, denoted by $v_0, \ldots, v_{n-1}$; it is trivial to extend the results to multiple trajectories. We further assume that the grid $\mathcal{G}$ we use to partition the soccer field is a square grid with $m \times m$ cells. Recall that $P[p, p']$ denotes the subtrajectory from p to p'. We say that $P[p, p']$ is a *C-to-C' subtrajectory* if p lies on the boundary of cell C and p' lies on the boundary of cell C' and $P[p, p']$ does not intersect C and C' except at p and p'. For two points $p, p' \in P$ we write $p \prec p'$ when p comes before p' in the order along P.

2.1. A look-up table for straight-path queries

As explained in the introduction, our first data structure is a look-up table that stores for every pair of grid cells C, C' the set $S(C, C')$ of all C-to-C' subtrajectories that are considered straight with respect to the given measure of straightness (dilation or direction deviation) and parameter (τ or α). More precisely, for each such subtrajectory $P[p, p']$ we store its starting point p and endpoint p'. The main questions are then: (i) how do we construct the sets $S(C, C')$ efficiently, and (ii) what is the maximum size of the data structure, that is, how large can $\sum_{C,C'} |S(C, C')|$ be.[2] Next we answer these questions for the two straightness measures that we use.

In the sequel we call a point where P crosses from one cell into the next a *transition point*. (To deal correctly with degenerate situations we define each cell to be closed on the bottom and to the left, and open on the top and to the right. Thus vertical edges belong to the cell lying to their right and horizontal edges belong to the cell above; vertices belong to the cell to their top-right.) A transition point is an *exit point* for the cell being exited, and an *entry point* for the cell being entered. We denote the sequence of transition points by $p_0, p_1, \ldots$, where the transition points are ordered along P. We denote the cell from which P exits at p_i by $C_{\text{exit}}(p_i)$, and the cell being entered by $C_{\text{entry}}(p_i)$.

2.1.1. Direction deviation

We first study direction deviation as straightness measure.

Computing all α-straight subtrajectories. We first describe how to compute the sets $S(C, C')$ when direction deviation is used as straightness measure. Let α be the given straightness parameter, where we assume $0 \leqslant \alpha < \pi/2$. Note that α-straightness is a monotone criterion: if a subtrajectory $P[p, q]$ is α-straight, then any subtrajectory $P[p', q']$ with $p \prec p' \prec q' \prec q$ is also α-straight. Thus we can follow the following strategy: we walk along P from start to finish, and at each transition point p_j we walk back along P to report all α-straight subtrajectories of the form $P[p_i, p_j]$, where p_i is a transition point with $i < j$. Because α-straightness is a monotone criterion, we can stop the backwards walk as soon as we encounter a transition point p_i for which $P[p_i, p_j]$ is not α-straight. A problem with this approach is that if P has many consecutive vertices inside the same cell then we spend a lot of time walking back through that cell, which can cause a high running time. We thus have to proceed more carefully.

We model directions as points on the unit circle S^1. A subtrajectory $P[p_i, p_j]$ is α-straight if and only if the smallest circular interval of S^1 that contains all points corresponding to the directions of the edges of the subtrajectory has length α—see Fig. 1. Our algorithm now works as follows. As we walk along P we compute for each consecutive pair of transition points p_j, p_{j+1} the smallest circular interval $I(p_j, p_{j+1})$ containing all directions of the subtrajectory $P[p_j, p_{j+1}]$, if this interval has length at most α; if the interval has length greater than α then $I(p_j, p_{j+1})$ is defined to be NIL. (The smallest interval is uniquely defined when it has length at most α, since $\alpha < \pi$.) Note that if $P[p_j, p_{j+1}]$ is a single segment then $I(p_j, p_{j+1})$ degenerates to a point on S^1. At each transition point p_i we walk backwards from transition point to transition point (thus skipping over vertices of P) as long as the subtrajectories are α-straight. To check this we maintain the smallest circular interval I^* that contains all intervals $I(p_j, p_{j+1})$ that we encountered in the backwards walk.

[2] By hashing techniques we can make sure we only store information for pairs C, C' such that $S(C, C') \neq \emptyset$, so the amount of storage is indeed $O(\sum_{C,C'} |S(C, C')|)$.

Algorithm 1 ***FindStraightSubtrajectories***(P, α).

1. Set $j := 0$ and create an empty list $\mathcal{L}$ for storing transition points.
2. Walk along P from v_0 to v_{n-1}, tracing the trajectory through the grid. Whenever P crosses from one cell into another, do the following:
   (i) Create a transition point p_j at the crossing point. If $j = 0$ then skip to Step (v), otherwise continue with Step (ii).
   (ii) Set $I(p_{j-1}, p_j) := \mathit{SmallestInterval}(P[p_{j-1}, p_j])$
   (iii) $I^* := I(p_{j-1}, p_j)$; let *ptr* point to the end of $\mathcal{L}$.
   while $I^* \neq$ NIL and *ptr* $\neq$ NIL
   do Let p_i be the transition point *ptr* points to.
   if $C_{\text{exit}}(p_i) = C_{\text{entry}}(p_j)$
   then *ptr* := NIL
   else Report $P[p_i, p_j]$ as an α-straight $C_{\text{exit}}(p_i)$-to-$C_{\text{entry}}(p_j)$ subtrajectory. If $I(p_i) =$ NIL then $I^* :=$ NIL, else $I^* := \mathit{Merge}(I^*, I(p_i))$.
   Move *ptr* backwards (to its predecessor).
   (iv) If there is a transition point p_i in $\mathcal{L}$ with $C_{\text{exit}}(p_i) = C_{\text{exit}}(p_j)$—we can test this in $O(1)$ time by maintaining some extra information—then we remove p_i from $\mathcal{L}$ and we set $I(p_{i'}) := \mathit{Merge}(I(p_i), I(p_{i'}))$, where $p_{i'}$ is the successor of p_i in $\mathcal{L}$.
   (v) Append p_j to $\mathcal{L}$ with $I(p_j) := I(p_{j-1}, p_j)$, and set $j := j + 1$.

We ignored one aspect so far: the fact that $P[p_i, p_j]$ is α-straight is not sufficient for the subtrajectory to be reported. We also need $P[p_i, p_j]$ to be a valid $C_{\text{exit}}(p_i)$-to-$C_{\text{entry}}(p_j)$ subtrajectory. This is violated if $P[p_i, p_j]$ intersects $C_{\text{exit}}(p_i)$ or $C_{\text{entry}}(p_j)$ at some point other than p_i or p_j. To make sure our algorithm is output sensitive, we have to avoid reporting $P[p_i, p_j]$ in this case. This can be done by removing a transition point p_i from our list of transition points as soon as we encounter another transition point $p_{i'}$ with $C_{\text{exit}}(p_{i'}) = C_{\text{exit}}(p_i)$. This ensures that when we report $P[p_i, p_j]$, then $P[p_i, p_j]$ does not intersect $C_{\text{exit}}(p_i)$ except at p_i. When we remove p_i, we have to "merge" the intervals $I(p_{i-1}, p_i)$ and $I(p_i, p_{i+1})$ into a new interval $I(p_{i-1}, p_{i+1})$. To facilitate this we maintain an ordered list $\mathcal{L}$ of all encountered transition points that have not been deleted yet, and with each transition point p_i in $\mathcal{L}$ we store an interval $I(p_i)$ which is the smallest circular interval containing all directions of the subtrajectory $P[p_{i'}, p_i]$ (or NIL if this interval has length more than α), where $p_{i'}$ is the predecessor of p_i in $\mathcal{L}$. To avoid reporting a subtrajectory $P[p_i, p_j]$ that intersects $C_{\text{entry}}(p_j)$ at some point other than p_j, we can simply stop our backwards walk when we encounter a transition point p_i with $C_{\text{exit}}(p_i) = C_{\text{entry}}(p_j)$.

Algorithm 1 describes our algorithm in more detail. Subroutine $\mathit{SmallestInterval}(P[p_i, p_j])$ outputs the smallest circular interval containing all edge directions of $P[p_{j-1}, p_j]$ or, when this interval has length more than α, it outputs NIL. Similarly, for two circular intervals I_1, I_2 the subroutine $\mathit{Merge}(I_1, I_2)$ outputs the smallest circular interval containing I_1 and I_2 or, when this interval has length more than α, it outputs NIL.

Since walking back from a transition point p_j takes time $O(1 + k_j)$, where k_j is the total number of reported subtrajectories, we get the following theorem.

Theorem 1. *Let P be a trajectory with n vertices in a domain that is an $m \times m$ grid, and let α be a constant with $0 \leqslant \alpha < \pi/2$. Then we can compute all sets $S(C, C')$ of α-straight subtrajectories of P in $O(n + k)$ time, where k is the total size of all sets.*

Analysis of the number of α-straight trajectories. Next we prove bounds on $\sum_{C,C'} |S(C, C')|$, the total number of α-straight trajectories. To simplify the bounds we make the assumption that the average length of the segments $v_i v_{i+1}$ of P is at most the edge length of the grid cells. With our grid cells having size of 1 m $\times$ 1 m, for instance, and a sampling rate of 20 Hz this is clearly a realistic assumption. From now on we assume without loss of generality that the cells in the grid $\mathcal{G}$ have unit size, and that the average length of the segments $v_i v_{i+1}$ in P is at most 1. We call such a trajectory a *short-edge trajectory*. The following theorem states the main result for this case. Observe that in practice α would be chosen fairly small, in which case $\cos(\alpha/2)$ will be close to 1. The bound in the next theorem then becomes $O(nm)$.

Theorem 2. *Let P be a short-edge trajectory with n vertices within an $m \times m$ unit grid, and let $0 \leqslant \alpha < \pi/2$. For a pair C, C' of grid cells, let $S(C, C')$ be the collection of all C-to-C' α-straight subtrajectories. Then $\sum_{C,C'} |S(C, C')| = O(\min\{n^2, nm^2, nm/\cos(\alpha/2)\})$, and this bound is tight in the worst case.*

To prove Theorem 2 we first bound the length of any α-straight subtrajectory.

Lemma 3. *The length of any α-straight trajectory P' in an $m \times m$ unit grid is at most $\sqrt{2}m/\cos(\alpha/2)$.*

Proof. Since the directions of all edges in P' differ by at most α, there is a direction $\vec{d}$ such that any edge in P' makes an angle at most $\alpha/2$ with $\vec{d}$. Let ℓ be a line with direction $\vec{d}$, and project all edges of P' orthogonally onto ℓ. Note that if the projection of some edge s_i of P' onto ℓ has length x_i, then $|s_i| \leqslant x_i/\cos(\alpha/2)$. Furthermore, since $\alpha < \pi/2$, the projections of these segments have disjoint interiors. Hence, $|P'| = \sum_i |s_i| \leqslant \sum_i x_i/\cos(\alpha/2) = |pp'|/\cos(\alpha/2)$, where p and p' are the projections of the start and endpoint of P' onto ℓ. Because P' lies inside an $m \times m$ grid, we have $|pp'| \leqslant \sqrt{2}m$. □

The following lemma provides us with an upper bound on the number of distinct cells visited by a trajectory of length L.

Lemma 4. *A trajectory of length L on a unit grid visits $O(L + 1)$ cells.*

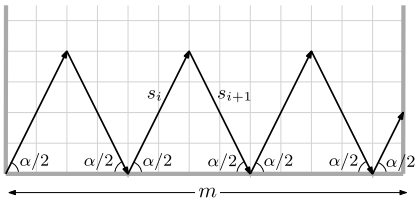

Fig. 2. A lower-bound witness for Lemma 5. Trajectory P is a sequence of segments s_i. In order to have a short-edge trajectory we assume that each segment s_i consists of several unit-length edges. The figure shows one round of P.

Proof. Consider the buffer region around the trajectory that consists of all points at distance at most $\sqrt{2}$ from the trajectory. Since the side length of the grid cells is 1, all cells visited by P lie in this buffer region. Hence, the area of the buffer, which is at most $2\sqrt{2}L + 2\pi = O(L+1)$, gives an upper bound on the number of cells visited by P. □

Using Lemma 3, we can now prove the upper bound from Theorem 2. Indeed, since the average length of the segments is at most 1, the total length of P is at most n. By Lemma 4 this implies that the total number of transition points of P is $O(n)$, which gives an $O(n^2)$ upper bound on the total number of subtrajectories between transition points (irrespective of whether they are α-straight or not). It also implies an upper bound of $O(nm^2)$, since there are only m^2 distinct cells C to start a C-to-$C_{\text{entry}}(p_j)$ subtrajectory for a fixed transition point p_j. On the other hand, for each transition point p_j, the length of the longest α-straight subtrajectory ending at p_j is at most $\sqrt{2}m/\cos(\alpha/2)$ by Lemma 3. Using Lemma 4 we can therefore bound the number of α-straight subtrajectory ending at p_j by $O(m/\cos(\alpha/2))$, which gives an $O(nm/\cos(\alpha/2))$ on the total number of α-straight subtrajectories. Combining this with the $O(n^2)$ and $O(nm^2)$ bounds, we obtain the claimed upper bound.

Lemma 5 gives a lower bound on the maximum number of α-straight subtrajectories that matches the upper bound just proved, thus completing the proof of Theorem 2.

Lemma 5. *For any constant $0 \leqslant \alpha < \pi/2$ and any positive integers n and m, there exists a short-edge α-straight trajectory with n vertices within an $m \times m$ unit grid such that $\sum_{C,C'} |S(C,C')| = \Omega(\min\{n^2, nm^2, nm/\cos(\alpha/2)\})$.*

Proof. For given n and m, Fig. 2 shows a trajectory with the claimed properties. It starts at the lower left corner of the field, then moves diagonally upwards at an angle $\alpha/2$ with the x-axis, then moves down when a grid line is hit, and so on. The zigzagging continues until the right boundary of the field is reached, at which point the path goes back in a similar fashion. Depending on how small or large the number of vertices in P is, P goes back and forth between the vertical boundary edges of the grid for several times. Imagine splitting P into subtrajectories at the points where it hits the vertical boundary edges of the grid; we call the resulting subtrajectories *rounds*. To prove the claimed lower bound we consider the following two cases. For $n < m/\cos(\alpha/2)$ the trajectory P consists of at most one round and it visits $\Omega(n)$ cells. Since in this case the subtrajectory between any pair of visited cells is α-straight, we get a lower bound of $\Omega(n^2)$ on the total number of α-straight subtrajectories. On the other hand, if $n \geqslant m/\cos(\alpha/2)$ then P consists of $n/(m/\cos(\alpha/2))$ rounds and each complete round visits

$$\Omega(\sum |s_i|) = \Omega(m/\cos(\alpha/2))$$

cells. Hence, a complete round consists of $\Omega(m^2/\cos^2(\alpha/2))$ α-straight subtrajectories and therefore the trajectory P consists of $\Omega(nm/\cos(\alpha/2))$ α-straight subtrajectories. Note that this construction is only valid for $m \geqslant 1/\cos(\alpha/2)$, otherwise the trajectory would leave the field through the top edge. For $m < 1/\cos(\alpha/2)$ we therefore get a bound of $\Theta(m^2 n)$. □

Remark. If the trajectory is not restricted to be short-edge then the upper bound of Theorem 2 becomes $O(\min\{n^2s^2, nm^2s, nms/\cos(\alpha/2)\})$, where s denotes the average length of the segments in the trajectory, and this bound is tight in the worst case. To see this, a similar argument as in the proof of Theorem 2 applies here. Indeed, the total number of transition points of P is $O(sn)$, the trivial upper bound on the total number of subtrajectories caused by P becomes $O(n^2s^2)$. In addition, since P can generate at most $O(sn)$ transition points, we can bound the total number of α-straight subtrajectories as $O(\min(nm^2s, nms/\cos(\alpha/2)))$ using Lemmas 4 and 3. To see that the upper bound is tight in the worst case, we can re-use the trajectory P in Fig. 2, the only difference is that we do not need to split each segment s_i into unit-length edges, which means we can have more rounds.

2.1.2. Dilation

We now turn our attention to dilation as a measure of straightness. We denote the dilation of $P[p,p']$ by $\text{dil}(P[p,p'])$.

Computing all subtrajectories with dilation at most τ. Compared to direction deviation as straightness measure, dilation is more difficult to handle, because it is not a monotone criterion: if $\text{dil}(P[p,p']) \leqslant \tau$ then a subtrajectory $P[p',q']$ with

Algorithm 2 ***FindSmallDilationSubtrajectories***(P, τ).

1. Set $j := 0$ and initialize an empty data structure $\mathcal{D}$.
2. Walk along P from v_0 to v_{n-1}, tracing the trajectory through the grid. Whenever P crosses from one cell into another, do the following:
   (i) Create a transition point p_j at the crossing point.
   (ii) If $j > 0$ then query the data structure $\mathcal{D}$ to find all transition points p_i such that $\psi_i \in \Gamma_\tau(p_j)$ and $i > j'$, where $p_{j'}$ is the most recent transition point with $C_{\text{entry}}(p_{j'}) = C_{\text{entry}}(p_j)$. (If there is no such transition point, then $j' = -1$.) For each such transition point p_i, report $P[p_i, p_j]$ as a $C_{\text{exit}}(p_i)$-to-$C_{\text{entry}}(p_j)$ subtrajectory with dilation at most τ.
   (iii) Insert $\psi_j := (x_j, y_j, d_j)$ into $\mathcal{D}$. If $\mathcal{D}$ already stored a transition point p_i with $C_{\text{exit}}(p_i) = C_{\text{exit}}(p_j)$ then we delete the corresponding point ψ_i from $\mathcal{D}$.
   (iv) Set $j := j + 1$.

$p \prec p' \prec q' \prec q$ may still have dilation larger than τ. Hence, when we want to find all subtrajectories $P[p_i, p_j]$ of dilation at most τ and ending at a given transition point p_j, and we walk back from p_j, then we cannot stop when we encounter a transition point p_i such that $\text{dil}(P[p_i, p_j]) > \tau$. A simple solution would be to always walk back all the way until the length of the subtrajectory is more than $\tau \cdot \sqrt{2}m$—since the distance between any two points on P is at most $\sqrt{2}m$ we do not have to walk back further—and at each transition point p_i check whether $\text{dil}(P[p_i, p_j]) \leqslant \tau$. This gives a quadratic solution for computing all subtrajectories with dilation at most τ. Next we improve upon this by presenting an output-sensitive algorithm.

The idea of our solution is that, as we walk along P, we store the transition points in a suitable data structure $\mathcal{D}$. This data structure allows us to perform a query with the current transition point p_j to find all transition points p_i such that $P[p_i, p_j]$ is a $C_{\text{exit}}(p_i)$-to-$C_{\text{entry}}(p_j)$ subtrajectory of dilation at most τ. To this end we associate to each transition point p_i a point $\psi_i := (x_i, y_i, d_i)$ in $\mathbb{R}^3$, where $d_i = |P[v_0, p_i]|$ and x_i and y_i are the x- and y-coordinate of p_i, respectively. Note that the values d_i can be computed in constant time as we walk on P, if we maintain the total length of the traversed part of P. Now when we arrive at transition point p_j, we are looking for all transition points p_i such that $\text{dil}(P[p_i, p_j]) \leqslant \tau$, that is, such that $(d_j - d_i)/\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \leqslant \tau$. Thus if we define the range $\Gamma_\tau(p_j)$ in $\mathbb{R}^3$ for p_j as

$$\Gamma_\tau(p_j) := \{(x, y, d) : \tau(x_j - x)^2 + \tau(y_j - y)^2 - (d_j - d)^2 \geqslant 0\}$$

then we are looking for all points p_i such that $\psi_i \in \Gamma_\tau(p_j)$.

As before, there is one other aspect to be taken into account: we are only allowed to report a subtrajectory $P[p_i, p_j]$ when it does not intersect $C_{\text{exit}}(p_i)$ except at p_i and it does not intersect $C_{\text{entry}}(p_j)$ except at p_j. The former is guaranteed by deleting a transition point p_i from $\mathcal{D}$ when we encounter a transition point $p_{i'}$ with $i' > i$ such that $C_{\text{exit}}(p_{i'}) = C_{\text{exit}}(p_i)$. The latter is guaranteed by refining our query: when we arrive at transition point p_j we find the most recent transition point $p_{j'}$ with $C_{\text{entry}}(p_{j'}) = C_{\text{entry}}(p_j)$—we can find this point (if it exists) in $O(1)$ time if we maintain a pointer from each grid cell to its most recent entry point—and then we only search for exit points p_i with $i > j'$. Algorithm 2 describes this in more detail.

It remains to describe the data structure $\mathcal{D}$, which should answer the following type of queries:

Given a query point p_j and an index j', report the points p_i such that

$$\psi_i \in \Gamma_\tau(p_j) \text{ (in other words, with } \text{dil}(P[p_i, p_j]) \leqslant \tau) \text{ and } i > j'. \qquad (*)$$

First we focus on the condition $\psi_i \in \Gamma_\tau(p_j)$. The range $\Gamma_\tau(p_j)$ is a semi-algebraic set in $\mathbb{R}^3$. Hence, we can use the range-searching data structure of Agarwal et al. [1], which uses $O(n^{1+\varepsilon})$ storage and expected preprocessing (for any fixed $\varepsilon > 0$) and has query time $O(n^{2/3} + k)$, where k is the number of reported points. We can improve the query time if we allow more preprocessing, using standard techniques. For instance, we can obtain logarithmic query time using $O(n^3)$ preprocessing. To this end we map every point p_i to an algebraic surface $\Sigma_\tau(p_i)$ in $\mathbb{R}^3$, defined as

$$\Sigma_\tau(p_i) := \{(x, y, d) : \tau(x - x_i)^2 + \tau(y - y_i)^2 - (d - d_i)^2 = 0\}.$$

Now, whether or not $\text{dil}(P[p_i, p_j]) \leqslant \tau$ is determined by on which side of $\Sigma_\tau(p_i)$ the point p_j lies. Thus we can find all points p_i such that $\text{dil}(P[p_i, p_j]) \leqslant \tau$ by performing point location with p_j in the arrangement defined by the surfaces $\{\Sigma_\tau(p_i) : i < j\}$. The latter can be solved in $O(\log n)$ time after $O(n^{3+\varepsilon})$ preprocessing [4]. Unfortunately, we cannot afford cubic preprocessing. However, we can combine our first data structure with the cubic-storage solution in a standard manner [2, Exercise 16.16] to obtain a trade-off between storage and query time. In particular, for any s with $n \leqslant s \leqslant n^3$ we can construct a data structure using $O(s^{1+\varepsilon})$ expected preprocessing so that a query can be answered in time $O(n^{1+\varepsilon}/s^{1/3})$.

Recall that we need to extend the data structure such that when we do a query for entry point p_j we only report subtrajectory $P[p_i, p_j]$ when $i > j'$, where j' is defined as in Algorithm 2. This can be done by adding a so-called *range restriction* to the data structure [12]. We also need our data structure to be dynamic, that is, we need to be able to do insertions and deletions. This can be done by applying the logarithmic method [8] in combination with *weak deletions*. By applying these techniques we can obtain, for any fixed $\varepsilon > 0$, a data structure in which queries take $O(n^{1+\varepsilon}/s^{1/3})$ time and updates take $s^{1+\varepsilon}/n$ expected time. We now choose $s = n\sqrt{n}$ to balance the query time and insertion time. Putting everything together, we obtain the following result.

Theorem 6. *Let P be a trajectory with n vertices in a domain that is an $m \times m$ grid, and let $\tau \geqslant 1$ be a constant. Then, for any fixed $\varepsilon > 0$, we can compute all sets $S(C, C')$ of subtrajectories of P with dilation at most τ in expected time $O(n^{1.5+\varepsilon} + k)$, where k is the total size of all sets.*

Remark. The data structure described above is rather complicated and not very practical. In practice it is better to use a different data structure such as an octree to do the range searching.

Analysis of the number of subtrajectories with dilation at most τ. As before we assume the grid consists of unit-size cells, and we make the realistic assumption that we are dealing with short-edge trajectories. The following theorem states the number of subtrajectories with dilation at most τ in such trajectories.

Theorem 7. *Let P be a short-edge trajectory with n vertices within an $m \times m$ unit grid, and let $\tau \geqslant 1$. For a pair C, C' of grid cells, let $S(C, C')$ be the collection of all C-to-C' subtrajectories of dilation at most τ. Then $\sum_{C,C'} |S(C, C')| = O(\min\{n^2, nm^2, \tau nm\})$, and this bound is tight in the worst case.*

Proof. The proof is very similar to the proof for α-straight trajectories: For the upper bound we observe that the total number of transition points for a short-edge trajectory is $O(n)$. For each entry point p_j, the number of exit points p_i such that $P[p_i, p_j]$ is a $C_{\text{exit}}(p_i)$-to-$C_{\text{entry}}(p_j)$ subtrajectory is $O(\min(n, m^2))$, since on the one hand there are only $O(n)$ exit points and on the other hand each cell C can generate at most one C-to-$C_{\text{entry}}(p_j)$ subtrajectory for fixed p_j. Furthermore, the maximum length of any subtrajectory of dilation τ inside the grid is $\tau \cdot \sqrt{2}m$. Such a trajectory intersects $O(\tau m)$ cells, which gives an $O(\tau m)$ upper bound on the total number of $C_{\text{exit}}(p_i)$-to-$C_{\text{entry}}(p_j)$ subtrajectories for fixed p_j. For the lower bound we can use a similar construction as in Fig. 2. This time we choose the slope of the segments s_i such that the dilation of one zigzag is exactly τ (assuming $\tau \leqslant m$), which means we pick α such that $\cos(\alpha/2) = 1/\tau$. □

Remark. If the trajectory is not restricted to be short-edge then the upper bound of Theorem 7 becomes $O(\min\{n^2s^2, nm^2s, \tau nms\})$, where s denotes the average length of the segments in the trajectory, and this bound is tight in the worst case. To see this, a similar argument as in the proof of Theorem 7 applies here. First, observe that the total number of transition points for the trajectory is $O(ns)$ this time. Second, for each entry point p_j the number of exit points p_i such that $P[p_i, p_j]$ is a $C_{\text{exit}}(p_i)$-to-$C_{\text{entry}}(p_j)$ subtrajectory is $O(\min(ns, m^2))$. Third, similar to the argument used in the proof of Theorem 7, since any trajectory of dilation at most τ inside the grid intersects $O(\tau m)$ cells the total number of $C_{\text{exit}}(p_i)$-to-$C_{\text{entry}}(p_j)$ subtrajectory is $O(\tau m)$ for each fixed entry transition point p_j. To see that the upper bound is tight in the worst case, we can re-use the trajectory P in Fig. 2, the only differences are that we do not need to split each segment s_i into unit-length edges, and we need to pick α such that $\cos(\alpha/2) = 1/\tau$.

2.2. A more space-efficient alternative

Explicitly storing all sets $S(C, C')$ of straight subtrajectories leads to fast and accurate queries, but it is costly in terms of storage. Below we present a much more space-efficient alternative. This comes at the cost of slightly slower query times and the fact we may also report some subtrajectories that pass near to the starting cell C_s of the query (rather than starting exactly at C_s). The alternative solution works for direction deviation as straightness measure.

Let P be the given n-vertex trajectory inside an $m \times m$ grid $\mathcal{G}$, and let α be a given straightness threshold with $0 \leqslant \alpha < \pi/2$. Recall that direction deviation is a monotone criterion, so for any entry point p_j there is a point $p \prec p_j$ such that $P[p', p_j]$ is α-straight for all $p \preceq p' \prec p_j$ and $P[p', p_j]$ is not α-straight for any $p' \prec p$. We call $P[p, p_j]$ the *longest α-straight subtrajectory* for p_j. For a cell $C \in \mathcal{G}$, let $L(C)$ denote the set of all longest α-straight subtrajectories of P ending at some entry point on the boundary of C. For each cell C, we store the set $L(C)$ in a priority search tree[3] PST[C], as explained next.

Consider a cell C, an entry point p_j of C and the longest α-straight subtrajectory $P[p, p_j] \in L(C)$. We associate a 2-dimensional point $\chi(p_j)$ with this subtrajectory, as follows. Let $\phi(pp_j)$ be the counterclockwise angle that the directed segment pp_j makes with the positive x-axis. Then the point $\chi(p_j)$ is defined as $\chi(p_j) := (\phi(pp_j), |pp_j|)$. This gives us a set $X(C) := \{\chi(p_j) : P[p, p_j] \in L(C)\}$ of points in $\mathbb{R}^2$, which we store in PST[C]. Recall that a short-edge trajectory induces $O(n)$ transition points. Since for each transition point p_j we only store the longest subtrajectory $P[p, p_j]$ ending at p_j, we have the following lemma.

Lemma 8. *Let P be a short-edge trajectory with n vertices. Then the total amount of storage needed for all priority search trees* PST[C] *is $O(n)$.*

[3] A priority search tree [2, Section 10.2] stores a planar point set such that all points in a semi-infinite range $[x_1 : x_2] \times [y : \infty)$ can be reported efficiently.

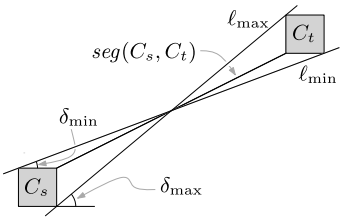

Fig. 3. Definition of $seg(C_s, C_t)$, $\delta_{\min}$ and $\delta_{\max}$.

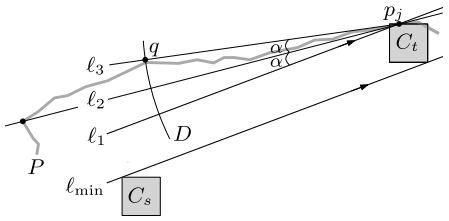

Fig. 4. Illustration for the proof of Lemma 9.

We now explain how we answer a straight-path query with starting cell C_s and target cell C_t. To simplify the presentation, we assume that C_t lies to north-east of C_s; the other cases can be handled in a symmetrical manner.

Let $seg(C_s, C_t)$ denote a shortest directed line segment connecting C_s to C_t. Let $\ell_{\min}$ denote the common tangent of C_s and C_t with minimum slope, and $\ell_{\max}$ denote the common tangent of C_s and C_t with maximum slope. Finally, let $\delta_{\min}$ and $\delta_{\max}$ denote the angles that $\ell_{\min}$ and $\ell_{\max}$ make with the positive x-axis; see Fig. 3. We then perform a semi-infinite range query on the priority search tree PST[C_t] with the semi-infinite range $R(C_s, C_t)$ defined as

$$R(C_s, C_t) := [\delta_{\min} - \alpha : \delta_{\max} + \alpha] \times [|s(C_s, C_t)| : \infty).$$

Thus, intuitively, we report a longest subtrajectory $P[p, p_j]$ if the direction of pp_j is similar to the direction of $seg(C_s, C_t)$ and pp_j is at least as long as the minimum distance between C_s and C_t. The following lemma states that the subtrajectories we report include all subtrajectories from C_s to C_t, and that any subtrajectory we report passes near C_s.

Lemma 9. *(i) Let $P[p_i, p_j]$ be an α-straight C_s-to-C_t subtrajectory. Then $\chi(p_j)$, the point stored for $P[p_i, p_j]$ in* PST[C_t]*, lies in the range $R(C_s, C_t)$. (ii) Let $P[p, p_j]$ be a subtrajectory in $L(C_t)$ such that $\chi(p_j) \in R(C_s, C_t)$. Then the distance from $P[p, p_j]$ to C_s is at most $O(1 + \sin(2\alpha) \cdot |seg(C_s, C_t)|)$.*

Proof. (i) Let $P[p, p_j]$ be the longest α-straight subtrajectory ending at p_j. Since $p \preceq p_i$ and $\alpha < \pi/2$, we have $|pp_j| \geqslant |p_i p_j|$. Since $|P[p_i, p_j]| \geqslant |seg(C_s, C_t)|$ by definition of $seg(C_s, C_t)$, this implies $|pp_j| \in [|seg(C_s, C_t)| : \infty)$.

To prove that $\phi(pp_j) \in [\delta_{\min} - \alpha : \delta_{\max} + \alpha]$, we observe that $P[p_i, p_j]$ must contain an edge whose angle with the positive x-axis is at most $\delta_{\max}$, since otherwise $P[p_i, p_j]$ would pass below C_s. Similarly, $P[p_i, p_j]$ must contain an edge whose angle with the positive x-axis is at least $\delta_{\min}$, otherwise $P[p_i, p_j]$ would pass above C_s. Because $P[p, p_j]$ is α-straight, this implies that all edges of $P[p, p_j]$ have angles in the range $[\delta_{\min} - \alpha : \delta_{\max} + \alpha]$. Hence, the angle of pp_j must be in the range as well.

(ii) We prove the bound for the case where $P[p, p_j]$ passes above C_s; the proof when it passes below C_t is similar.

We first analyze where the point p can lie. Let ℓ_1 be the line parallel to $\ell_{\min}$ and passing through p_j, and let ℓ_2 be the line through p_j making an angle α with ℓ_1; see Fig. 4. Observe that p must lie on or below ℓ_2, otherwise $\phi(pp_j) < \delta_{\min} - \alpha$. Now let ℓ_3 be the line making an angle α with ℓ_2. Then $P[p, p_j]$ cannot go above ℓ_3, otherwise $P[p, p_j]$ would not be α-straight. Draw the circle D centered at p_j and of radius $|seg(C_s, C_t)|$. Since $|pp_j| \geqslant |seg(C_s, C_t)|$, the point p must lie outside D. The worst case is now that $P[p, p_j]$ intersects D at the point q where D intersects ℓ_3, as illustrated in Fig. 4. An elementary computation now shows that the distance from q to C_s is $O(1 + \sin(2\alpha) \cdot |seg(C_s, C_t)|)$. □

Notice that the error in (ii)—that is, the distance from $P[p, p_j]$ to C_s—is a constant number of cells plus a fraction of $|seg(C_s, C_t)|$ that tends to zero as α tends to zero. Thus the error does not depend on $|pp_j|$, which is desirable since $|pp_j|$ can be large compared to $|seg(C_s, C_t)|$. It should be noted that the subtrajectory we report is guaranteed to pass near C_s, but does not necessarily start near C_s.

3. Distance-preserving path simplification

In this section we study a path-simplification problem and improve the fastest known algorithm for the problem, using the data structure presented in Section 2.1.2. Let $P = (p_1, p_2, \ldots, p_n)$ be a path with n vertices and $\tau \geqslant 1$ be a real

number. A path $Q = (p_{i_1}, p_{i_2}, \ldots, p_{i_k})$, with $1 = i_1 < i_2 < \cdots < i_k = n$, is called a *$\tau$-distance-preserving approximation* of P if $\mathrm{dil}(P[p_{i_t}, p_{i_{t+1}}]) \leqslant \tau$ for all $1 \leqslant t < k$. Gudmundsson et al. [6] introduced the *Minimum Vertex Path Simplification (MVPS) problem*: given a path P and a threshold $\tau \geqslant 1$, compute a τ-distance-preserving approximation of P having the minimum number of vertices. Their algorithm for the problem runs in $O(n^2)$ time. Our approach to improve this time bound uses dynamic programming. For any $1 \leqslant j \leqslant n$, let $M[j]$ denote the minimum number of vertices on any τ-distance-preserving approximation of $P[p_j, p_n]$. Define $S(j) := \left\{ p_i : i > j \text{ and } \mathrm{dil}(P[p_i, p_j]) \leqslant \tau \right\}$. Then $M[j]$ satisfies

$$M[j] = \begin{cases} 1 & \text{if } j = n \\ 1 + \min\left\{ M[i] \ : \ p_i \in S(j) \right\} & \text{if } j < n \end{cases}$$

Explicitly checking each point $p_i \in S(j)$ when computing $M[j]$ will lead to a quadratic algorithm. To speed up the algorithm we will augment the data structure from Section 2.1 so that, given a query index i, we can compute $\min\{M[i] : p_i \in S(j)\}$ quickly.

The data structure. Observe that the condition $p_i \in S(j)$ is essentially the same as the condition $(*)$ on page 32. Hence, we can report all points in $S(j)$ using the data structure described on page 32. However, we only want to report the point p_i with the minimum $M[i]$ value. To this end we have to augment the data structure with extra information.

The data structure is a two-level tree[4] whose first level is a binary search tree $\mathcal{T}$ storing all the vertices on the path based on their indices. This level enables us to restrict the attention to points p_i with $i > j$. With each node v of the first-level tree $\mathcal{T}$, we have an associated tree $\mathcal{T}'_v$ that allows us to select all points p_i with $\mathrm{dil}(P[p_i, p_j]) \leqslant \tau$. This selection comes in the form of a number of nodes w whose canonical subsets together form a disjoint partition of $S(j)$. Thus we augment the data structure as follows: with each node w in any of the associated trees, we store the value $M_w := \min\{M[i] : p_i \in S_w\}$, where S_w is the canonical subset associated to w. (When $M[i]$ is not known yet, it is defined as $+\infty$.) When we now perform a query with a point p_j, we search our data structure to select a set of nodes w whose canonical subsets form $S(j)$, and we take the report of the smallest M_w-value among the selected nodes.

The algorithm. Now the algorithm simply works as follows. We construct the data structure described above, where each M_w-value is initialized to $+\infty$, and we initialize $M[n] := 1$. We then compute each value $M[j]$, for $j = n-1, \ldots, 1$, by performing a query with p_j as described above. After having computed $M[j]$ we set $M_w := \min(M_w, M[j])$ for each node w whose canonical subset contains p_j.

We have seen before that the data structure can be constructed in $O(n^{1.5+\varepsilon})$ time and that each query takes $O(n^{0.5+\varepsilon})$ time. Since any point p_j is stored in $O(n^{0.5+\varepsilon})$ canonical subsets, updating the M_w-values at each step can be done in $O(n^{0.5+\varepsilon})$ as well. Hence, we obtain the following result.

Theorem 10. *The MVPS problem can be solved in $O(n^{1.5+\varepsilon})$ expected time.*

Note that $M[0]$ is the minimum number of vertices for the MVPS problem. In order to be able to report the set of vertices associated with the value of $M[0]$—that is, the desired minimum-size simplification—we store at each $M[j]$ the vertex $p_i \in S(j)$ giving the minimum M-value. We then only need to follow the sequence of vertices stored in array M and starting at $M[0]$.

4. Experimental results

In the previous sections we analyzed the worst-case storage of our look-up table and the worst-case error of our priority-search-tree. In this section we perform an initial experimental evaluation of our structures. Our focus is on the two aspects mentioned above (storage and error), and not on the preprocessing times. Hence, we implemented simplified versions of the algorithms described in the previous section and did not optimize the algorithms. We have implemented the algorithms in C++ using GNU g++, version 4.8.1. Our test machine has an Intel i7-3770 processor with 16 GB main memory and running the Windows 7 Enterprise 2009 operating system.

Data sets. We have taken our test trajectories from the publicly available soccer data set by Petterson et al. [9]. This data set describes the movement of the home players in three professional soccer games captured in 2013 at the Alfheim Stadium in Tromsø, Norway. The data sets are obtained using body sensors and the ZXY Sport Tracking System, which reports the players' position (as well as some other measurements) at 20 Hz with an approximate error of one meter; see the paper by Petterson et al. [9] for details.

To obtain our trajectory we arbitrarily selected one player (player #9) from the game `Tromsø IL-Strømsgodset`, and we focused on the movement of this player in the first half of the game. The grid we used in most experiments has cells of size 1 m $\times$ 1 m. As mentioned, the players' positions are reported at 20 Hz. We also generated trajectories at 10 Hz,

[4] For the current application we do not need insertions and deletions, so we do not need to apply the logarithmic method.

Table 1
Average length of the trajectory segments for different sampling rates.

	20 Hz	10 Hz	5 Hz	2 Hz	1 Hz
# segments	53 059	26 546	13 295	5330	2679
Average length	0.10835	0.21647	0.43155	1.06740	2.07262

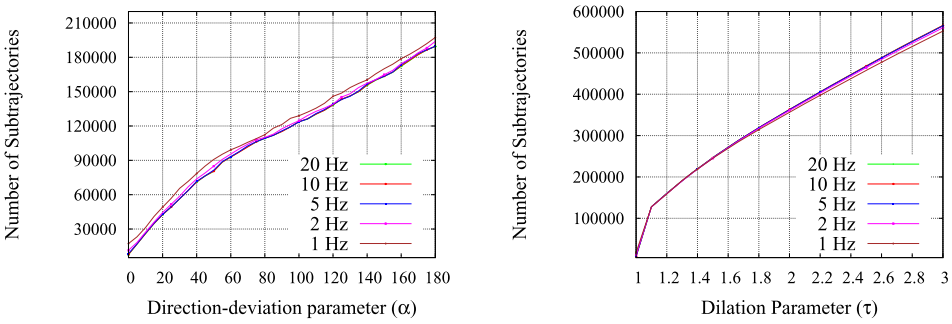

Fig. 5. The sizes of the data structures after running Algorithm 1 (left figure), and Algorithm 2 (right figure), on trajectories P_1, P_2, P_5, P_{10}, and P_{20}.

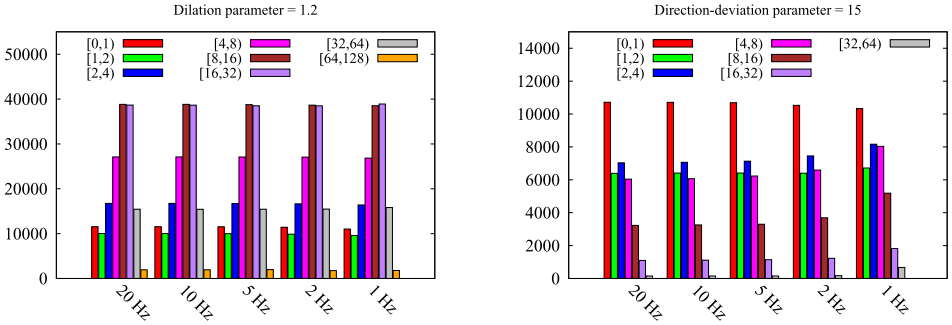

Fig. 6. The number of subtrajectories inside different ranges of distance between cells. The y-axis indicates the number of subtrajectories.

5 Hz, 2 Hz, and 1 Hz by sub-sampling the original trajectory. The reason was to smooth out noise in the reported locations although, as we will see, the different sampling rates give very similar results in the experiments. Table 1 shows the average length of the segments of the trajectories, which shows that the short-edge assumption is realistic. To ease in reference, we let P_k denote the trajectory obtained at k Hz sampling rate.

Size of the look-up table. We first investigate the size of our look-up table. More precisely, we compute $\sum_{C,C'} |S(C, C')|$, the number of almost straight C-to-C' subtrajectories over all pairs of cells C, C'. We do this for dilation with parameter τ in the range $[1, 3]$ with step size 0.1, and for direction deviation with parameter α in the range $[0, 180]$ with step size 5 degrees. Fig. 5 shows the results of the experiment.

As Fig. 5 shows our approach is quite feasible since the number of almost straight C-to-C' subtrajectories is much less than what our theoretical bounds state. Of course, the total number of almost straight C-to-C' subtrajectories increases with increasing τ and with increasing α. Interestingly, the numbers are almost independent of the sampling rate. This is surprising for direction deviation as straightness measure, since especially for low values of α we expected that the paths sampled at 1 Hz would have much longer straight subtrajectories than for 20 Hz. On the other hand, the 20 Hz path visits more cells, and this apparently almost cancels out the other effect. For dilation the fact that more cells are visited even slightly dominates that paths become straighter, since for large τ the total number of paths is (very slightly) larger with increasing sampling rate.

In addition to analyzing the total size of our data structures, we have also computed a breakdown of the number of trajectories according to the distance between C and C'. In particular, we have partitioned the set of distances into buckets of form $[2^{i-1}, 2^i)$, and counted how many subtrajectories fall into each of the buckets. Fig. 6 illustrates the results of the experiment.

First, Fig. 6(left) indicates that the distribution of the number of subtrajectories with dilation at most $\tau = 1.2$ is independent of sampling rate. Being independent of sampling rate is as expected since increasing the sampling rate only smoothes out the subtrajectory $P[p, p']$, for any p and p', by replacing each segment of $P[p, p']$ by a set of shorter segments and therefore this should not affect the length of $P[p, p']$ much. On the other hand, Fig. 6(right) reveals an interesting fact for α-straight subtrajectories: since decreasing the sampling rate increases the length of the segments of the trajectory, the

Table 2
Results of the space-efficient alternative described in Section 2.2 for different queries on trajectory P_{10}. Error is the maximum distance, in meters, over all distances of the reported subtrajectories to cell C_s.

α	15	15	30	30	60	60
# answers	1	2	2	2	3	0
# reported subtrajectories	1	3	4	2	5	1
$\texttt{dist}(C_s, C_t)$	4.12	3.61	5.83	6.40	15.62	8.54
Error	0	0.44	1.74	0	2.09	1.98

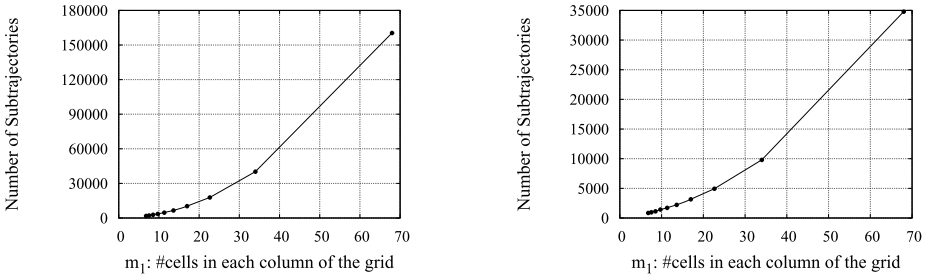

Fig. 7. The number of subtrajectories for different values of side length for grid cells. The number of grid cells is $m_1 \times m_2$, which is $m_1 \times \lceil 1.544 m_1 \rceil$ cells. Left figure shows the number of subtrajectories in P_{10} with dilation at most $\tau = 1.2$. Right figure shows the number of α-straight subtrajectories in P_{10} with $\alpha = 15$.

length of α-straight subtrajectories increases. Hence, decreasing the sampling rate creates more α-straight subtrajectories for higher ranges of distances between cells.

Accuracy of the priority search tree. In order to test the efficiency of our space-efficient alternative described in Section 2.2, we performed several queries on trajectory P_{10} (results for other sampling rates were similar). For each query pair (C_s, C_t), we measured the accuracy of reported answers by computing the maximum over all distances of reported subtrajectories to cell C_s. Table 2 summarizes the experiment. For the queries shown in the table (and, in fact, for all queries we did) the number of answers is a small constant, even for $\alpha = 60$, and the number of reported subtrajectories is not much larger. The maximum error is just a few cells.

Dependency on the side length of grid cells. Finally we investigate the dependency of sizes of our data structures on m. We consider P_{10} and different values for m and then measured the sizes of corresponding data structures. Consult Fig. 7 for results of the experiment. The number of subtrajectories seems to grow superlinearly with m_1. We expected a linear behavior, since α is a constant in the experiment ($\alpha = 15$) and $m_1 < n$, so the worst-case growth rate according to Theorems 2 and 4 is linear. Note, however, that the upper bounds in Theorems 2 and 4 are for short-edge trajectories, and changing the cell size (which is what we did to vary m_1) also changes the ratio of the edge lengths and the cell size. The lower-bound for trajectories that are not short-edge is a factor m higher (see the remark on page 31), and shows a quadratic dependence on m, which seems more in line with the experiments.

5. Concluding remarks

In this paper we studied a practical problem arising when developing a support system for the analysis of soccer matches: we want to store a given trajectory in a data structure such that, for a query start position s and target position t, we can quickly retrieve all subtrajectories of the given trajectory that are similar to the line segment st. In order to obtain a practical solution we partitioned the playing field into a regular grid, and map the points s and t to the centers of the containing grid cells.

We studied the resulting data-structuring problem for two similarity measures, dilation and direction deviation. We presented two simple data structures for this problem, gave efficient algorithms to construct them, and analyzed their worst-case performance theoretically. We also investigated their practical performance on trajectories from a publicly available data set with trajectories of soccer players. Our experimental study shows that our structures perform very well in practice.

It would be interesting to extend our approach to more complicated queries, for example to the case where the query is a polyline consisting of two or three (rather than one) segment.

Acknowledgements

We thank the anonymous WALCOM reviewers for their useful feedback and suggestions on preliminary version of the paper. The second author also wishes to warmly thank Dirk Gerrits for his help in initial steps of our implementation work and several useful discussions afterward.

References

[1] P.K. Agarwal, J. Matošek, M. Sharir, On range searching with semialgebraic sets II, SIAM J. Comput. 42 (2013) 2039–2062.
[2] M. de Berg, O. Cheong, M. v. Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, 3rd edition, Springer-Verlag, 2008.
[3] M. de Berg, A.F. Cook, J. Gudmundsson, Fast Fréchet queries, Comput. Geom. Theory Appl. 46 (6) (2013) 747–755.
[4] B. Chazelle, H. Edelsbrunner, L.J. Guibas, M. Sharir, A singly exponential stratification scheme for real semi-algebraic varieties and its applications, Theor. Comput. Sci. 84 (1) (1991) 77–105.
[5] D.H. Douglas, T.K. Peuker, Algorithms for the reduction of the number of points required to represent a line or its caricature, Can. Cartogr. 10 (2) (1973) 112–122.
[6] J. Gudmundsson, G. Narasimhan, M. Smid, Distance-preserving approximations of polygonal paths, Comput. Geom. Theory Appl. 36 (3) (2007) 183–196.
[7] J. Gudmundsson, M.H.M. Smid, Fréchet queries in geometric trees, in: Proc. Europ. Symp. Alg., 2013, pp. 565–576.
[8] M.H. Overmars, The Design of Dynamic Data Structures, LNCS, vol. 156, Springer, 1983.
[9] S.A. Pettersen, D. Johansen, H. Johansen, V. Berg-Johansen, V.R. Gaddam, A. Mortensen, R. Langseth, C. Griwodz, H.K. Stensland, P. Halvorsen, Soccer video and player position dataset, in: Proc. 5th Int. Conf. Multi. Syst., 2014, pp. 18–23.
[10] U. Ramer, An iterative procedure for the polygonal approximation of plane curves, Vis. Graph. Image Process. 1 (1972) 244–256.
[11] C. Shim, J. Chang, Y. Kim, Trajectory-based video retrieval for multimedia information systems, in: Proc. 3rd Int. Conf. on Advances in Info. Syst., vol. 3261, 2005, pp. 372–382.
[12] D.E. Willard, G.S. Luecker, Adding range restriction capability to dynamic data structures, J. ACM 32 (1985) 597–617.
[13] X. Yu, D. Farin, Current and emerging topics in sports video processing, in: Proc. Intl. Conf. Mult. & Expo., 2005, pp. 526–529.