DimWeaver: Dimensional Weaved Trajectory Compression

XENOPHON KITSIOS, Athens University of Economics and Business, Greece
PANAGIOTIS LIAKOS, Athens University of Economics and Business, Greece
KATIA PAPAKONSTANTINOPOULOU, Athens University of Economics and Business, Greece
NIKOS GIATRAKOS, Technical University of Crete, Greece
YANNIS KOTIDIS, Athens University of Economics and Business, Greece

High-resolution trajectory data form the basis for numerous applications involving marine and air traffic, drone and robotic navigation, and the study of human mobility in unconstrained environments. These applications often generate challenging 2D/3D movement data that span vast, freely navigable geographic areas. In this work, we introduce DimWeaver, a new trajectory simplification scheme which removes redundancies in trajectory representations by combining three novel techniques: i) *decoupling* of trajectories to constituent 1D components to exploit kinematic linearities per individual dimension, ii) *weaving* error allocation, allowing a constituent dimension to momentarily absorb more error, thus simplifying localized, complex movements more effectively, and iii) *grouping* 1D movements with similar velocity scalar values *within* and *across* dimensions, into the fewest possible groups, to enhance space-efficiency. Our experiments on eight datasets from diverse domains show that DimWeaver consistently outperforms existing baselines in terms of compression ratio. Moreover, for equivalent compression ratios, DimWeaver reconstructs the original trajectories more accurately, achieving significantly lower Root Mean Squared Error.

CCS Concepts: • **Information systems** → **Spatial-temporal systems**.

Additional Key Words and Phrases: Trajectory simplification

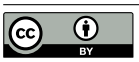
ACM Reference Format:
Xenophon Kitsios, Panagiotis Liakos, Katia Papakonstantinopoulou, Nikos Giatrakos, and Yannis Kotidis. 2026. DimWeaver: Dimensional Weaved Trajectory Compression. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 163 (June 2026), 26 pages. https://doi.org/10.1145/3802040

1 Introduction

Mobile devices, GPS sensors, and IoT deployments are omnipresent in a wide range of applications providing massive amounts of trajectory data, capturing movements of individuals, animals, drones, vessels and other objects over time [28, 32]. While high-resolution trajectories provide accurate representations of moving-object behavior, they frequently incorporate information redundancy, noise, or insignificant positional changes, resulting in greater storage, transmission, and processing costs, ultimately impeding the delivery of timely analytics and decision-making [44, 47, 51].

Trajectory simplification and compression very often serve as invaluable methods to address such challenges [27, 47]. Several works [9, 14, 16, 20, 26, 29–31, 34, 35, 39] recognize the importance of developing algorithms for efficient trajectory simplification and compression under application-defined accuracy constraints. At an abstract level, all these works effectively address the problem at hand, primarily focusing on detecting approximate kinematic linearities among the spatiotemporal

Authors' Contact Information: Xenophon Kitsios, Athens University of Economics and Business, Greece, xkitsios@aueb.gr; Panagiotis Liakos, Athens University of Economics and Business, Greece, panagiotisliakos@aueb.gr; Katia Papakonstantinopoulou, Athens University of Economics and Business, Greece, katia@aueb.gr; Nikos Giatrakos, Technical University of Crete, Greece, ngiatrakos@tuc.gr; Yannis Kotidis, Athens University of Economics and Business, Greece, kotidis@aueb.gr.



ACM 2836-6573/2026/6-ART163
https://doi.org/10.1145/3802040

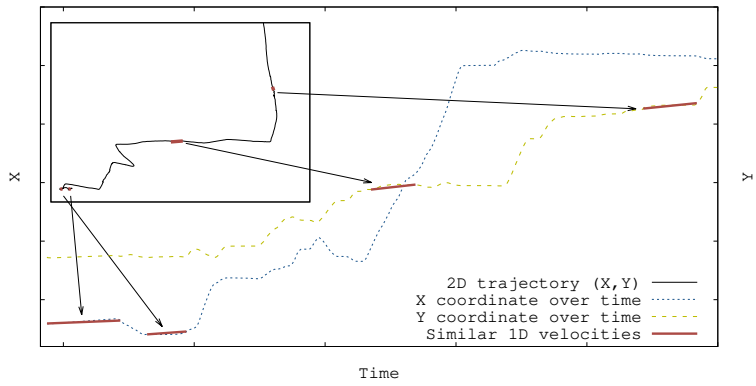


Fig. 1. Trajectory of a robot moving freely in an open area. Kinematic grouping identifies trajectory fragments—potentially distant in space, time, and across both dimensions—with similar 1D velocities and represents them using a single shared velocity value.

data using an error budget ϵ defined on an approximation metric such as the Synchronized Euclidean Distance (SED) [26, 34, 35], the Perpendicular Euclidean Distance (PED) [9, 14, 20, 29, 30] or the Direction-Aware Distance (DAD) [16, 31]. Thus, they drop intermediate 2D points that lie sufficiently close to a line segment connecting their surrounding points, under the assumption that such points do not significantly alter the overall shape or temporal dynamics of the trajectory.

Surprisingly, despite their special focus on pinpointing approximate kinematic linearities, all these approaches seem to neglect a fundamental observation: 2D or 3D kinematic linearity with respect to time t is equivalent to simultaneous 1D kinematic linearity in all coordinates; however, larger linear patterns may be observed when individual dimensions are examined separately. Therefore, prior art misses subtle (even approximate) linearities in the raw trajectory data that can be detected only upon *decoupling* the simplification process into simpler 1D constituents.

At first glance, decoupling may seem counter-intuitive for compression purposes. This is because line segments necessitate additional metadata and may require repeating timestamps across dimensions. Nonetheless, decoupling enables two additional forms of redundancy removal that prior work has overlooked: i) flexible error budget *weaving* (i.e., dynamic redistribution), and ii) the calculation of ranges of admissible 1D velocities that can be consolidated into a single scalar velocity.

The decoupled 1D trajectory components can be processed individually but on par with each other to identify approximate kinematic linearities. More specifically, decoupling enables flexible error budget allocation *weaving*, because ϵ can be dynamically distributed between the 1D compression subtasks, instead of allocating it evenly. This allows the dynamic allocation of greater error tolerance, on demand, to the dimension that temporarily needs it the most, boosting compression and still abiding by ϵ in 2D/3D.

Beyond flexible error allocation, decoupling also enables the removal of additional latent redundancies by revealing the potential for *kinematic grouping* strategies. Figure 1 illustrates the free motion of a robot in two dimensions, where speed and direction vary freely over time. By decomposing the 2D trajectory into its 1D components, we can identify intervals whose 1D velocities fall within the same tolerance-defined range, rather than requiring exact equality. This allows us to match intervals that occur at different times or along different axes. Our method exploits this observation by selecting a single representative velocity pattern –instead of the 4 distinct velocity values of Figure 1– capturing both consecutive and non-consecutive motion across

the highlighted segments from either the X or the Y dimension. This kinematic grouping enables trajectory fragments that may be far apart in time or space to be merged and encoded using a single shared value.

Exploiting these observations, we introduce DimWeaver, a novel error-bounded, SED-based [47] trajectory compression scheme, which incorporates the whole set of discussed redundancy removal means, currently overlooked by prior art. Our contributions are:

- We propose DimWeaver, a novel *one-pass trajectory compression method* that introduces the decoupling of the input trajectory into its individual 1D components and compresses them in a coordinated manner to preserve cross-component consistency.
- DimWeaver accepts an *application-defined error budget* and *weaves*, i.e., dynamically distributes, it between the 1D compression subtasks, allowing greater error tolerance, on demand, to the dimension that needs it the most, boosting compression efficiency.
- DimWeaver identifies 1D movements with *similar* velocity scalar values and groups them both within and across components, into the *fewest possible groups*. It, thus, employs a more powerful compression procedure that unravels all types of redundancies in the raw trajectory data.
- Experiments on eight diverse datasets show that DimWeaver surpasses existing techniques, delivering higher compression ratios and substantially lower reconstruction error.

2 Preliminaries

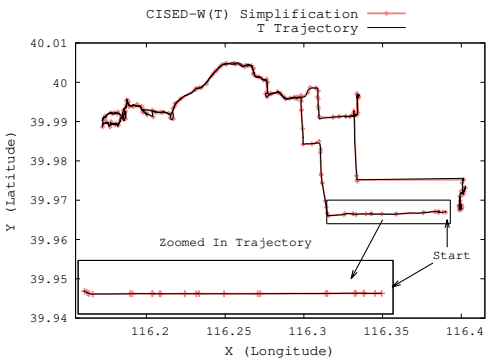
2.1 Trajectory Data

Many trajectory-processing techniques assume movement constrained to a predefined 2D network, as in vehicle tracking on road graphs [28]. This assumption enables strong optimizations—such as map-matching GPS coordinates to network elements and using network-based rather than Euclidean distances for movement prediction [6, 7, 17, 21, 23]. In contrast, we study the orthogonal [27] problem of unconstrained trajectories, where people, animals, robots, marine vessels, aircrafts and drones move freely in 2D or 3D space without any underlying network. This enables our techniques to support applications characterized by unpredictable movement or trajectories that traverse extensive geographic areas. As a result, road-mapping–based techniques—and the large body of methods built around them—are not applicable in our setting and therefore not directly comparable. Throughout this work, we use the term trajectories to refer specifically to this unconstrained scenario.

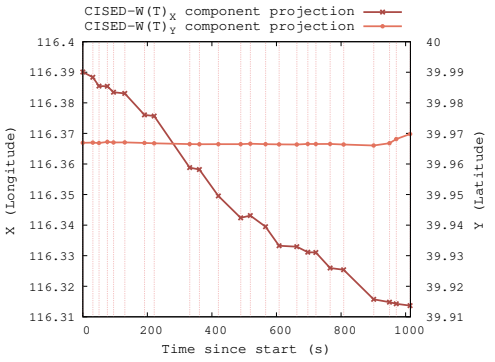
A trajectory is defined as a sequence of n spatiotemporal points, denoted as $T = [p_1, p_2, \ldots, p_n]$, where each point is typically represented as $p_i = (x_i, y_i, t_i)$, with (x_i, y_i) indicating the spatial location and t_i the corresponding timestamp. Although most existing studies focus on trajectories in 2D space, the spatial component is not inherently constrained to 2D and can be generalized to 3D. To ensure the temporal validity of a trajectory, the timestamps must be strictly ordered, i.e., $t_1 < t_2 < \cdots < t_n$.

Trajectory data are typically generated by devices compatible with a Global Navigation Satellite System (GNSS). For instance, a smartwatch may record a user's movements every 2–3 seconds while cycling. In these scenarios, the y coordinate corresponds to latitude, x to longitude, and t denotes the sampled time, often represented as a Unix timestamp.

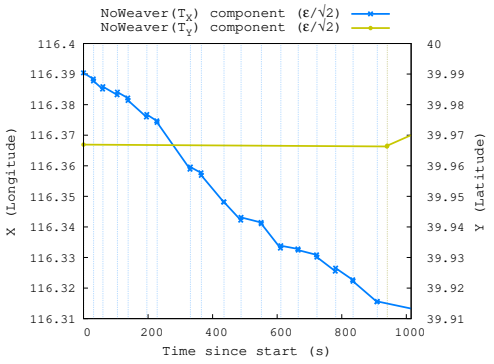
In other contexts, however, trajectory points may be represented using alternative coordinate systems. For example, in a factory environment, spatial coordinates might refer to positions within a local reference frame —such as distances in meters from a predefined origin— rather than global geographic coordinates. Similarly, timestamps may reflect relative or system-specific time units, depending on the data acquisition setup and application requirements.



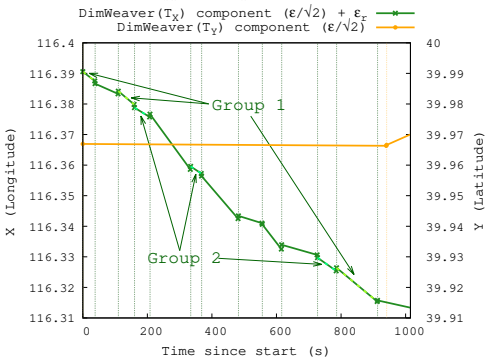
(a) Original trajectory T and its simplification T' using algorithm CISED-W



(b) Projection of the simplified trajectory T' onto its 1D components T'_X and T'_Y (zoomed-in area)



(c) Result of decoupled simplification using NoWeaver, a stripped-down baseline that omits all key optimizations and design principles introduced in DimWeaver (zoomed-in area)



(d) Weaved simplification of the original 1D components T_X and T_Y using our DimWeaver algorithm (zoomed-in area). Error tolerance ϵ_r not consumed during the simplification of T_Y is reallocated to the more challenging X dimension, yielding fewer kinematic segments that are amenable to velocity-based grouping.

Fig. 2. Decoupled simplification achieved by using the 1D components of a 2D trajectory (Figure 2c), significantly reduces the number of points compared to a state-of-the-art 2D simplification approach (Figures 2a,2b). This decoupling unlocks additional optimizations (dimension weaving, kinematic grouping), which together yield compound savings (Figure 2d).

Given a trajectory T we can define its 1D projection onto one of its coordinates. For example assuming two dimensional spatiotemporal points $p_i = (x_i, y_i, t_i)$, then the projection on the X dimension is defined as $T_X = [(x_1, t_1), (x_2, t_2), \ldots, (x_n, t_n)]$. Similarly, $T_Y = [(y_1, t_1), (y_2, t_2), \ldots, (y_n, t_n)]$.

2.2 Trajectory Simplification

Trajectory simplification is a fundamental technique in trajectory compression. Given an original trajectory $T = [p_1, p_2, \ldots, p_n]$ of length n, the goal is to compute a simplified trajectory $T' = [p'_1, p'_2, \ldots, p'_m]$ of length $m \ll n$ such that T' closely approximates T, while significantly reducing the number of data points.

Simplification methods can generally be classified, among other criteria [27, 47], based on whether the simplified trajectory features exclusively original points or newly generated ones may also be included. If all points in T' are a subset of the original trajectory, i.e., $T' \subseteq T$, the simplification is referred to as *strong* simplification. In contrast, if one or more points in T' are generated during the process (i.e., they are not present in T), the method is called *weak* simplification [27]. The latter provides greater flexibility, as it is not limited to the original point set and can generate more compact representations. As a result, weak simplification methods achieve higher compression ratios compared to strong ones [27].

2.3 Trajectory Reconstruction

Using a simplified trajectory $T' = [p'_1, p'_2, \ldots, p'_m]$ we would like, given timestamp t_q, $1 \leq q \leq n$ to estimate the spatial location of the moving object based on the simplified trajectory. For instance, for query point $p_q = (x_q, y_q, t_q)$, we would like to estimate $p'_q = (x'_q, y'_q, t'_q)$, where $t'_q = t_q$, using T'.

Let's assume that t'_s and t'_e are two consecutive timestamps in T', such that $t'_s \leq t_q$ and $t_q \leq t'_e$. Assuming uniform motion between p'_s and p'_e, the spatial coordinates of reconstructed point $p'_q = (x'_q, y'_q, t'_q)$ are interpolated as [34]:

$$x'_q = x'_s + \frac{x'_e - x'_s}{t'_e - t'_s}(t'_q - t'_s), \quad y'_q = y'_s + \frac{y'_e - y'_s}{t'_e - t'_s}(t'_q - t'_s)$$

The Synchronized Euclidean Distance [34] (SED) metric accounts for the error in the approximation by computing the Euclidean distance of the time-synchronized points of the original (x_q, y_q) and simplified (x'_q, y'_q) trajectories:

$$SED(p_q, p'_q) = \sqrt{(x_q - x'_q)^2 + (y_q - y'_q)^2} \tag{1}$$

In this work, we focus on trajectory simplification techniques based on SED, which effectively preserves temporal information and supports time-sensitive queries [26, 34, 35]. Specifically, our goal is to derive a simplified trajectory T' that achieves high compression, while maintaining a maximum error threshold ϵ, such that $SED(p_q, p'_q) \leq \epsilon$ for all $1 \leq q \leq n$ points of the original trajectory.

3 Motivation

Trajectories from different domains may present unique, localized challenges for line simplification algorithms. Because these methods must account for variations across all spatial dimensions to preserve proximity, this requirement can at times limit their effectiveness. Figure 2a shows a two-dimensional GeoLife [52–54] trajectory and the points selected by `CISED-W` [26], a state-of-the-art simplification algorithm, when zooming in on the initial part of the trajectory. Notably, the motion occurs predominantly along the X-axis (Longitude), with only minor deviations in the Y-axis (Latitude). Consequently, the simplified trajectory largely selects points needed to preserve proximity to the original path based on variations in the X dimension. This is further shown in Figure 2b, where the simplified trajectory produced by `CISED-W`, is projected into its two one-dimensional components. Most breaks in the Y-projection are redundant, reflecting the much smoother motion along this axis. This suggests that, in this case, a simplification method treating the two dimensions independently could produce *fewer* points.

To evaluate this hypothesis, we implemented `NoWeaver`, a new trajectory simplification algorithm decoupling multi-dimensional trajectory points into independent 1D projections. `NoWeaver` applies `Slide` [10], a space-optimal [37] Piecewise Linear Approximation (PLA) method, separately to 1D components of the original trajectory, distributing the error budget ϵ evenly across dimensions. By

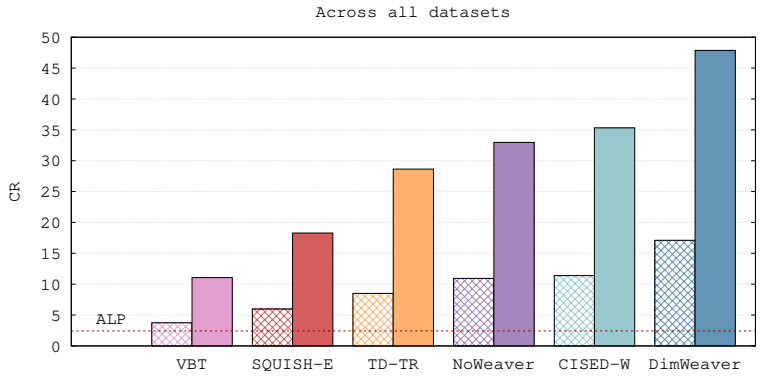


Fig. 3. Compression ratios of various algorithms.

operating on each projection independently, NoWeaver increases the flexibility of the simplification process and reduces the number of segmentation breaks per dimension—an effect particularly evident in the Y component, as shown in Figure 2c.

Figure 3 reports the overall compression results of NoWeaver for eight datasets and two representative error thresholds —small and large— against four well-established trajectory simplification methods: CISED-W, previously discussed; VBT [7], a fast, yet effective simplification algorithm, TD-TR [34], a widely adopted baseline; and SQUISH-E [35], the most effective streaming trajectory simplification algorithm currently available in the literature [27]. Surprisingly, NoWeaver consistently outperforms VBT, TD-TR and SQUISH-E, and nearly matches the performance of CISED-W.

This unexpected outcome revealed the untapped potential of the decoupling approach and motivated a deeper investigation, which led to DimWeaver, the main contribution of this work. DimWeaver extends the core decoupling principle of NoWeaver and introduces several optimizations that significantly improve compression performance. The most important of these novel techniques is the ability of DimWeaver to intelligently allocate greater error tolerance to the more challenging dimension of the trajectory. As a result, we see in Figure 2d, that DimWeaver produces fewer segments in the X-component of our sample trajectory, compared to CISED-W (Figure 2b) and NoWeaver (Figure 2c). Figure 2d also reveals that many of these segments can be clustered into groups, each characterized by nearly identical velocities, a second technique employed by our DimWeaver to boost compression efficiency.

Figure 3 shows that the optimizations of DimWeaver yield a substantial performance improvement over NoWeaver and other state-of-the-art trajectory simplification methods. For reference, Figure 3 also includes a straight dotted line indicating the compression ratio achievable by a recent lossless floating-point compression method, ALP[1]. Unlike floating-point compression, which typically exploits redundancy in the binary representation of consecutive values [24, 25], trajectory simplification algorithms can selectively discard redundant points, thus providing superior compression performance.

4 Our Trajectory Compression Method

4.1 NoWeaver Baseline

NoWeaver decouples trajectories and utilizes a segmentation subroutine to capture linear trends in their 1D components. The resulting line segments approximate the projected points, ensuring

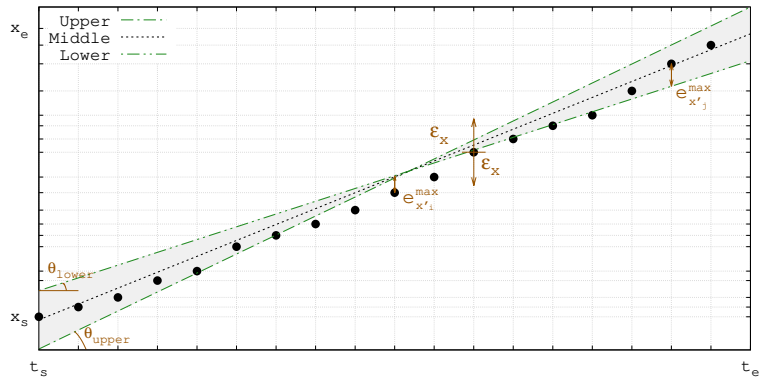


Fig. 4. Given a time segment $[t_s, t_e)$ containing 1D points, NoWeaver computes upper and lower bounding slope lines [10] that approximate all points within the segment, subject to a predefined error threshold ϵ_x. The maximum distance $e^{max}_{x'_i}$ of point x_i from the lines is usually smaller than ϵ_x, leaving residual error budget available for other dimensions.

they remain within the application-specified maximum error threshold ϵ. To enhance clarity and streamline notation, we provide the following description that focuses on the X-component, T_X.

Recall that $T_X = [(x_1, t_1), (x_2, t_2), \ldots, (x_n, t_n)]$ represents a time series of the trajectory's X-coordinates. A PLA algorithm segments a sequence of values subject to a specified maximum error threshold ϵ_x ($\epsilon_x = \epsilon/\sqrt{2}$ for NoWeaver). Such algorithms typically extend the current segment incrementally by incorporating the next value, as long as the resulting set of points can still be approximated by a line within the error bound. Although a wide variety of algorithms exist for generating such segmentations, a particularly notable method is the $O(n)$ time- and space-optimal algorithm Slide originally presented in [37] and later rediscovered in [10].

The Slide algorithm, used internally by NoWeaver, incrementally determines the minimal set of disjoint line segments needed to approximate a series within a maximum error ϵ_x. It maintains online two boundary lines, *upper* and *lower*, that converge as new points arrive, forming a cross-shaped feasible region, as illustrated in Figure 4, where any line within the bounds is a valid approximation. When a new point fits within this region, the boundaries are refined, with convex-hull points not shown in the figure accelerating the update process [10]. Otherwise, the current segment is finalized and a new one begins. When a segment is finalized, NoWeaver selects the internal bisector $\frac{\theta_{lower}+\theta_{upper}}{2}$ for forming the segment.

The DimWeaver algorithm we propose in this work incorporates our NoWeaver baseline as a black-box component, yet its superior performance arises from several crucial extensions that tailor this black-box component to the requirements of the problem at hand. These core optimizations are discussed next.

4.2 Overview of DimWeaver's core optimizations

Kinematic Representation of Segments. We utilize an alternative representation of a segment as (x_s, t_s, v^x_s) where v^x_s denotes the slope of the final line used to represent the points in $[t_s, t_e)$. This slope naturally represents a scalar velocity estimate on the constituent dimension for that interval. PLAs avoid this representation and utilize the endpoints of the segments instead (x_s, t_s, x_e), as in many applications, timestamps are expressed in Unix time (i.e., large integer values) resulting in

rounding errors when using floating-point arithmetic. To improve numerical stability, we choose to reparametrize the line equation using a relative offset:

$$x(t) = x_s + v_s^x \cdot (t - t_s), for\ t \in [t_s, t_e) \tag{2}$$

with

$$v_s^x = \frac{x_e - x_s}{t_e - t_s} \tag{3}$$

Representing motion as 1D linear segments allows `DimWeaver` to group even non-consecutive recurring velocity patterns within and across dimensions (Figure 1), an optimization not achievable with prior 2D velocity-tracking methods [6, 7, 39, 42, 46]. In addition, rather than committing to a single velocity estimate, `DimWeaver` maintains a range of admissible velocities, allowing these ranges to be merged later. This flexibility is key to discovering broader and more robust velocity-recurrence structures.

Initial Allocation of Error. Simplifying a decoupled trajectory calls for distributing the maximum error ϵ across its two dimensions, X and Y, while ensuring that the overall constraint is respected. Using Equation (1), we want for all points in $1 \leq i \leq n$:

$$\sqrt{(x_i - x_i')^2 + (y_i - y_i')^2} \leq \epsilon$$

We can divide the error *symmetrically* across the two dimensions so that:

$$|x_i - x_i'| \leq \delta, |y_i - y_i'| \leq \delta$$

Equivalently, we need:

$$\sqrt{(\delta)^2 + (\delta)^2} \leq \epsilon \Rightarrow \delta \leq \frac{\epsilon}{\sqrt{2}}$$

Therefore, the initial error allocated to each segmentation task on each dimension is $\frac{\epsilon}{\sqrt{2}}$. This allocation naturally adapts to 3D trajectories by allotting $\frac{\epsilon}{\sqrt{3}}$ per dimension. Due to the use of SED, the time dimension is not subject to approximation or independent error allocation.

Fine-grained error allocation. In `DimWeaver`, we allow the error bound on each dimension to vary per point rather than remain constant across the entire time series. While this has limited utility in conventional time series compression where all points adhere to a uniform error tolerance, it becomes particularly valuable in our case while processing decoupled trajectory components due to the dimensional weaving opportunity that arises.

Dimensional weaving. By adopting a fine-grained, per-point, non-uniform error allocation, we enable dimensional weaving, where one dimension temporarily receives additional error residuals available at the particular location from the other dimension, with the overall error always staying within ϵ. The question that remains to be answered involves a proper decision making mechanism for error budget reallocation.

Exponential Decay for Adaptive Dimensional Segmentation. Since trajectory characteristics can vary during movement, dimensional weaving requires a dynamic error allocation scheme that can adapt to such changes. In our approach, we introduce an exponential decay estimation mechanism to regulate the flow of error residuals, thereby coordinating the segmentation processes across dimensions.

Intermediate segment representation. When a segment is finalized, two sentinel lines can be derived: an *upper* line with the maximum admissible slope and a *lower* line with the minimum

admissible slope. Together, these lines define the permissible range of slopes consistent with the specified error threshold, similar to Figure 4. While processing a trajectory segment, DimWeaver utilizes two slope values, v_{lower} and v_{upper}, which define an *interval* of permissible velocity values $[v_{lower}, v_{upper}]$ that can represent points within the defined error bound via Equation 2.

Deferred, Kinematic Grouping of Motion Patterns. As we described in Section 4.1, when a segment closes, NoWeaver immediately selects the internal bisector slope $\frac{\theta_{lower}+\theta_{upper}}{2}$ for the final segment formation. However, in DimWeaver, we defer this decision until after processing all segments across all dimensions. This enables DimWeaver to cherry-pick among admissible velocities and finally assign common velocities to multiple intervals simultaneously, thereby avoiding redundant representations of segments with very similar slopes/scalar velocities.

4.3 DimWeaver

For clarity of presentation, we first describe the DimWeaver algorithm in the 2D setting and subsequently discuss in Section 4.8 the adaptations necessary for 3D operation. The overall workflow of DimWeaver is summarized in Algorithm 1.

We take as input a trajectory T and an application-specified error tolerance ϵ and generate a compressed representation of T that allows for all trajectory points to be reconstructed within ϵ using SED.

The algorithm breaks down T into its one-dimensional projections T_X and T_Y. These projections are processed simultaneously employing the segmentation techniques discussed in Section 4.1. Whenever possible, one of the two dimensions is prioritized, receiving additional error tolerance in the form of *error residuals*. Initially, the error residuals are assigned to a random dimension, for example Y in our exposition (Line 2). We refer to the dimension that receives additional error budget in the form of residuals as the *collector* dimension and the dimension that offers these residuals as the *distributor* dimension.

When a segment in the distributor X-dimension is closed (Line 16), the algorithm shifts to processing the collector Y-dimension on that part of the trajectory. Since some error budget in dimension X may remain unused, the algorithm allows the other dimension to exceed $\frac{\epsilon}{\sqrt{2}}$ for point y_i (Line 24) by applying a threshold:

$$e_{y_i} = \sqrt{(\epsilon)^2 - (e^{max}_{x'_i})^2} \tag{4}$$

where

$$e^{max}_{x'_i} = max(|x_i - x'^{lower}_i|, |x_i - x'^{upper}_i|) \tag{5}$$

Value x'^{lower}_i (or x'^{upper}_i) represents the reconstructed value of x_i, obtained using Equation 2 and applying the lower (resp. upper) slope of the recently closed segment on the X-dimension. Recall that at this stage, the final line for the X-segment has not yet been determined. However, any selected line is guaranteed to lie within interval $[v_{lower}, v_{upper}]$. Equation 5 computes two sentinel estimate values based on these lines and uses the maximum error value to represent the worst-case absolute error in the approximation of x_i, irrespective of the final slope selection. A visualization is provided in Figure 4.

By construction, it holds that $e^{max}_{x'_i} \leq \frac{\epsilon}{\sqrt{2}}$. Therefore, using Equation 4, the adjusted error threshold for y_i cannot be smaller than $\frac{\epsilon}{\sqrt{2}}$. The difference, denoted as the *residual* error $\epsilon^{res}_{y_i}$ for the Y segmentation process at point (y_i, t_i), is defined as:

$$\epsilon^{res}_{y_i} = \epsilon_{y_i} - \frac{\epsilon}{\sqrt{2}} \geq 0 \tag{6}$$

Algorithm 1: DimWeaver

```
1  Function DimWeaver(T, ε):
2      collector ← Y
3      len_X, len_Y = 0, 0
4      segs_X, segs_Y ← [], []
5      buffer ← []
6      foreach p ∈ T do // for each point
7          buffer.add(p) // buffer point for collector
           // proceed with distributor dimension
8          if collector == X then // process Y dimension
               // add (p.t, p.y) to segs_Y using NoWeaver()
9              closed ← NoWeaver(p.t, p.y, e/√2, segs_Y)
10             if closed then // update len_Y
11                 len_Y ← len_Y · (1 − c) + c · segs_Y.last().size()
12         else // process X dimension
13             closed ← NoWeaver(p.t, p.x, e/√2, segs_X)
14             if closed then // update len_X
15                 len_X ← len_X · (1 − c) + c · segs_X.last().size()
16         if closed then // when a distributor segment closes
17             foreach bp ∈ buffer do // for each buffered point
                   // adjust error by assigning residuals
                   // proceed with collector dimension
18                 if collector == X then // process X dimension
19                     e_Adj ← adjustError(segs_Y.last(), p.t, p.y, ε)
20                     closed ← NoWeaver(bp.t, bp.x, e_Adj, segs_X)
21                     if closed then // update len_X
22                         len_X ← len_X · (1 − c) + c · segs_X.last().size()
23                 else // process Y dimension
24                     e_Adj ← adjustError(segs_X.last(), p.t, p.x, ε)
25                     closed ← NoWeaver(bp.t, bp.y, e_Adj, segs_Y)
26                     if closed then // update len_Y
27                         len_Y ← len_Y · (1 − c) + c · segs_Y.last().size()
28             if len_X ≤ len_Y then // assign X as collector
29                 collector ← X
30             else // assign Y as collector
31                 collector ← Y
32             buffer ← []
33     return kinematicGrouping(segs_X ∪ segs_Y) // Algorithm 2
```

This residual error enables the redistribution of the error budget towards the more challenging dimension, adapting to localized movements in the trajectory at the observed timestamp. At the same time, it ensures that the 2D SED error at point $p_i(x_i, y_i, t_i)$ remains within ϵ.

This process of assigning residual errors to the points of the Y dimension continues until the current Y segment is finalized. At that moment, the algorithm reassesses which dimension should receive the residual error. Since dimensions that are more difficult to approximate with line segments tend to produce shorter segments, the algorithm employs an exponentially weighted scheme to

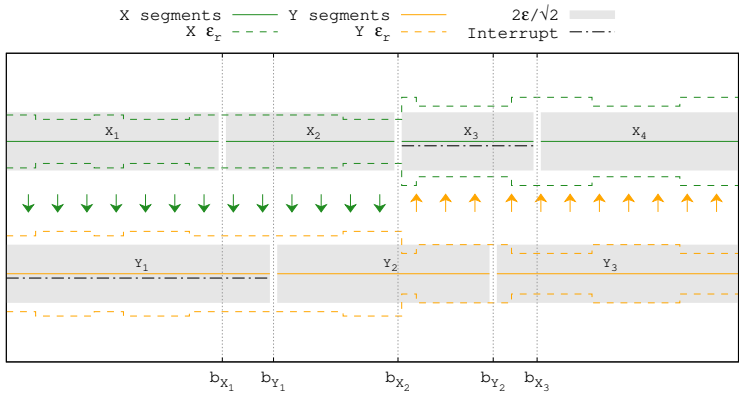


Fig. 5. Visualization of the error distribution occurring with DimWeaver.

estimate the average segment length across each dimension. In particular, whenever a segment in the X dimension closes with seg_x points, we update the estimate len_x for the average length of an X segment using:

$$len_x = (1 - c) \cdot len_x + c \cdot seg_x \tag{7}$$

where c is a decay parameter in the range $[0, 1]$. Initially, len_x is set to the length of the first X segment (Line 15). Similar computations are carried out for the other dimension (Line 11). In our implementation, we use a default decay value of 0.85 to slightly favor recent observations. The exponential-decay update on a dimension is triggered only when a segment is closed.

When deciding which dimension will act as a collector, the residual error is allocated to the one that has exhibited shorter intervals more recently, by comparing estimates len_x and len_y (line 28). This dimension now becomes the collector while the other takes on the role of the distributor. The exponential decay scheme allows the algorithm to adapt to evolving motion patterns while filtering out transient fluctuations.

To summarize, DimWeaver employs a weaved processing pattern to perform two segmentation tasks on par with one another, across the projected one-dimensional components of the trajectory, according to the following logic:

- The distributor dimension processes data until a new segment is completed, assuming a symmetric distribution of the error budget. Once the segment is closed —based on the lower and upper slope lines— the algorithm estimates a safe error bound per point within the segment that remains unused. Residual errors for the collector dimension are then computed using this bound.
- The segmentation of the collector dimension progresses by utilizing the residual errors until it reaches the point where the distributor dimension closed its segment. At that stage, the distributor dimension resumes processing to generate a new segment.
- When a segment in the collector dimension is completed, the algorithm re-evaluates the assignment of the collector and distributor. If the roles change, the update takes effect starting with the next segment produced by the collector.

Figure 5 illustrates the dynamic allocation of residual errors in our DimWeaver algorithm. For ease of presentation, we assume that $c = 1$ (Equation 7), meaning the dimension with the shortest latest completed segment becomes the collecting dimension. Initially, the collecting dimension is randomly chosen as Y. The segmentation of the distributor dimension X proceeds until the first segment, X_1, is completed at timestamp b_{X_1}. Next, the segmentation process for the first Y segment,

Y_1, is initiated utilizing larger errors bounds per point based on Equation 4 and the maximum errors $e^{max}_{x'_i}$ computed for each x'_i-estimate in segment X_1. When the first Y segment reaches the end of X_1 at time b_{X_1}, a new X segment (X_2) is initiated and processed until it closes at timestamp b_{X_2}. The collecting dimension Y then resumes processing segment Y_1, using the residuals generated from X_2. Once segment Y_1 completes at timestamp b_{Y_1}, the dimensions switch roles with dimension X becoming the collector and dimension y the distributor, as dictated by the choice of c. However, this transition occurs after b_{X_2}, since X_2 has already been processed and its segment cannot be modified without backtracking. Thus, between b_{Y_1} and b_{X_2}, Y continues receiving the precomputed residuals from X_2. At time b_{X_2} the switch takes effect and X_3 starts receiving residuals from Y_2. At time b_{Y_2} the distributor Y dimension initiates its final segment Y_3, whose residuals are used by X_3 and X_4 until the trajectory is completed.

As the algorithm sequentially processes points from the X and Y dimensions using modified `NoWeaver`'s $O(n)$ segmentation algorithm, it preserves a linear space and time complexity of $O(n)$. We note that the memory requirements in practical settings remain modest, since the algorithm only stores the points of the segments currently being processed.

4.4 Kinematic Grouping

`DimWeaver` produces kinematic segments that approximate the projected 1D trajectory points and facilitate the identification and grouping of recurring motions with similar velocities. Considering the X dimension, each segment is initially represented by both its upper and lower bound lines (Figure 4) in the form of a tuple:

$$(t_s, x_{s_{lower}}, v_{lower}, x_{s_{upper}}, v_{upper})$$

Here, $x_{s_{lower}}$ and $x_{s_{upper}}$ denote the starting points at $t = t_s$ of the upper and lower lines. A similar representation is used for the Y dimension. In line with strategies for interval grouping [13, 18, 19], we develop a method that partitions overlapping velocity intervals into the minimal number of non-overlapping groups. The procedure operates by first sorting the intervals from all dimensions according to their v_{lower} values and then successively computing their intersections in a forward pass. Each resulting group i is defined by a new interval $[v_{lower_i}, v_{upper_i}]$, where v_{lower_i} is the maximum lower bound among the group's segments, and v_{upper_i} is the minimum of upper bound.

After the groups are formed, retaining both their lower and upper bounding lines is unnecessary. Instead, we store the bisector line because it provides a balanced approximation and tends to yield lower overall error compared to either extreme. Thus, each group g contains:

- A velocity v_g for the whole group corresponding to the slope of the bisector line, which is used to represent all kinematic segments in the group.
- A list of triplets (dim, d_s, t_s), one for each segment assigned to the group, where dim is a flag that denotes the dimension (X or Y), d_s represents the starting point coordinate (x_s or y_s) of the segment based on the middle line selected for this group, and t_s is the starting timestamp of the segment.

The Kinematic grouping algorithm is presented in Algorithm 2. Initially, we sort the segments based on their v_{lower} values (line 4). Next, a new group of segments with intersecting slope values is formed (lines 6-9) by appropriately adjusting the lower and upper lines for the group. When the next segment cannot be merged into the current group, the group is finalized by computing the velocity v_g of its bisector line (line 11).

The subroutine $calcStart()$ is used to update the starting values d_s for all segments added to the group, incorporating the finalized group velocity v_g (lines 12-13). This subroutine first computes the timestamp of the intersection between the upper and lower lines of the segment (line 21), derives

Algorithm 2: Kinematic Grouping

```
1  Function kinematicGrouping(segments):
2      groups, groupedSegments ← [], []
3      v_lower, v_upper ← −∞, ∞
4      sort(segments); // sort segments by segments.v_lower
5      foreach s ∈ segments do
6          if s.v_lower ≤ v_upper and s.v_upper ≥ v_lower then
7              groupedSegments.add((s.dim, s.d_s, s.t_s))
8              v_lower ← max(v_lower, s.v_lower)
9              v_upper ← min(v_upper, s.v_upper)
10         else
11             v_g ← tan((arctan(v_lower)+arctan(v_upper))/2)
12             foreach gs ∈ groupedSegments do
13                 gs.d_s ← calcStart(segments[gs], v_g)
14             groups.add((v_g, groupedSegments))
15             groupedSegments ← []
16             groupedSegments.add((s.dim, s.d_s, s.t_s))
17             v_lower ← s.v_lower
18             v_upper ← s.v_upper
19     return groups
20 Function calcStart(s, v_g):
21     t_inter ← (s.v_upper·s.t_s − s.d_upper − s.v_lower·s.t_s + s.d_lower) / (s.v_upper − s.v_lower)
22     d_inter ← s.v_lower · (t_inter − s.t_s) + s.d_lower
23     return d_inter − v_g · (t_inter − s.t_s)
```

its corresponding point (line 22), and then utilizes the group velocity to compute its new starting point (line 23). Returning to Algorithm 2, the finalized group is appended to the result (line 14), and a new group is initiated containing the next segment (lines 15-18).

Algorithm 2 requires $O(m)$ space and $O(mlogm)$ time due to the sorting step, where m is the number of kinematic segments produced by DimWeaver. We note that, typically, $m << n$, where n is the size of the trajectory.

4.5 Binary Representation of Output

To efficiently encode the output of the kinematic grouping algorithm, we propose a binary representation scheme as outlined in Figure 6. The binary format is structured into three primary blocks: i) the Dimension Block, ii) the Counters Block, and iii) the Groups Block. The sizes of the first two blocks (in bytes) are explicitly stored to facilitate parsing. In contrast, the Groups Block does not require a size field, as it can be read until the end of the binary stream based on the metadata of the previous blocks. Additionally, at the beginning of the binary, we store the starting timestamp $t_{T_{\text{start}}}$ as a frame of reference.

The Dimension Block differentiates segments belonging to the two spatial dimensions. Each segment in the Groups Block is represented by a single bit: a 0-bit if the segment belongs to the X-dimension and a 1-bit if it belongs to the Y-dimension. Since the total number of segments may not always be a multiple of 8, up to 7 padding bits may be added to ensure byte alignment and maintain the validity of the binary format.

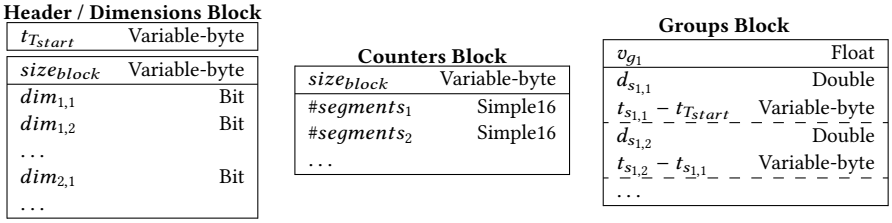


Fig. 6. Final binary encoding of the output of Algorithm 2. We use $dim_{i,j}$, $d_{s_{i,j}}$ and $t_{s_{i,j}}$ to denote the (dim, d_s, t_s) values of the j-th segment of the i-th group. v_{g_i} is the common velocity of the i-th group.

In the Counters Block, we store the number of segments associated with each group. The counter allows us to know how many (d_s, t_s) pairs in the Groups Block, stored next, are associated with each velocity v_{g_i}.

In the Groups Block, each group is written with its segments ordered by their timestamps. The first timestamp in the group is stored as a relative time over $t_{T_{\text{start}}}$ and subsequent timestamps are recorded using delta encoding. For each group i we first write its velocity v_{g_i}, followed by the starting points and timestamps $(d_{s_{i,j}}, t_{s_{i,j}})$ of each kinematic segment j associated with the group. The corresponding Dimension Block bit denotes the origin dimension of each segment, while the associated counter in the Counters Block specifies the number of segments, enabling the identification of group boundaries.

We use double-precision (8 bytes) to store starting points and single-precision (4 bytes) for velocity. For delta-encoded timestamps, we utilize Variable-byte encoding, which significantly reduces storage requirements, particularly for regularly spaced timestamps. For metadata (sizes, counters), where only integer values are used, we apply Variable-byte encoding and Simple16 compression [48], respectively.

4.6 Decompression Algorithm

Decompression of the binary output requires reconstructing the original segments produced by `DimWeaver` in increasing timestamp order, per dimension. Formally, this phase corresponds to the k-way merge step of a merge-sort algorithm, where each group contributes one sorted list of segments. Since all lists are already sorted in the Groups Block (Figure 6), the decompressor performs a single linear pass that merges the sublists in parallel. As a result, both the time and space complexity of this phase are linear ($O(m)$) in the total number of segments m, where typically $m << n$. Once the global (time-ordered) sequence of segments has been reconstructed, trajectory points can be recovered using Equation (2).

4.7 Long-running Trajectories and Updates

`DimWeaver` processes trajectory points in streaming fashion, whereas kinematic grouping conservatively operates only after the full trajectory is recorded. We remove this constraint with the following minor modification. When finalizing a group, instead of replacing the bounding lines with their bisector (Lines 11–14 in Algorithm 2), we retain the slope and intercept of both bounds. This makes kinematic grouping re-entrant; it can be invoked at any time, even while `DimWeaver` is processing the trajectory stream, and resume by merging previously stored groups with new `DimWeaver`-generated segments. The memory requirements of the modified algorithm is still $O(m)$ and the additional space overhead in the output is limited to storing two lines per group which is however amortized across all contained segments.

The re-entrant version of Algorithm 2 supports inserting new points into an already compressed trajectory without reprocessing the entire stream. In the rare case that an out-of-order point, with a timestamp earlier than the last one, must be inserted, we first attempt to approximate the point using the respective segment; if the error is within ϵ, no update is needed. Otherwise, we extract the X and Y segments covering that timestamp and re-run DimWeaver on just those points, including the new one. The resulting segments are then passed through kinematic grouping.

Deletions of trajectory points do not affect the reconstruction of the remaining points, since those points continue to be approximated within ϵ of the bisector line of their respective groups. Naturally, after multiple deletion operations, it may become more space-efficient to re-process the trajectory, as more compact segmentations may then be achievable.

4.8 Extensions to the basic algorithm

Additional extensions of the algorithm are worth noting, the most practical being support for 3D trajectories common in unconstrained motion (e.g., aircraft, drones, mobile robots in uneven terrain). In 3D, DimWeaver runs three segmentation processes, with the main challenge being how to allocate and propagate residual errors across dimensions. As in the 2D case, DimWeaver uses the lightweight predictor of Equation (7) to estimate expected segment length per dimension and orders dimensions accordingly. For example, if $len_y \geq len_z \geq len_x$, the Y dimension distributes residuals to Z, and any unused portion is passed to X. The rest of the pipeline is unchanged. Kinematic grouping is dimension-agnostic; the only adjustment is that the binary encoding now uses two bits to record each segment's originating dimension.

The slope estimation and exponential-decay computations in DimWeaver are lightweight, yielding high compression throughput in our experiments. Because the algorithm runs multiple segmentation tasks, it naturally supports hardware parallelism. These tasks may proceed speculatively under the assumption that their residual-passing order remains unchanged; if not, their partial results are discarded. In parallel, the kinematic grouping stage can begin receiving and ordering segments in Line 4 of Algorithm 2, as soon as each task emits them.

5 Experimental Evaluation

In this section, we first present the dataset used in our experiments. Then, we investigate the performance of our algorithms against earlier methods to address the following research questions:

(1) How effectively does DimWeaver compress data compared to existing approaches across a range of error thresholds?
(2) Is our approach competitive with earlier approaches in terms of compression speed?
(3) How does DimWeaver compare with state-of-the-art techniques in decompression throughput?
(4) Beyond efficiency, does our approach also deliver measurable gains with regard to the quality of the approximation?
(5) How well does DimWeaver generalize to the compression of 3D trajectories?
(6) How does DimWeaver compare with query-driven trajectory compression approaches?
(7) Can DimWeaver be efficiently extended to handle long-running trajectories?

5.1 Experimental Setup and Data

Experiments were conducted on a MacBook Pro equipped with an Apple M4 processor, 24GB of RAM and a 1TB SSD. Our study features comparisons against TD-TR [34], SQUISH-E [35], CISED-W [26] and VBT [7] algorithms, all implemented in Java. Moreover, we include NoWeaver as a baseline algorithm that performs decoupled simplification, to highlight the improvements introduced by

Table 1. Details about our trajectory datasets

Dataset	Domain	Accuracy range (m) corresponding to ϵ	Num. of trajectories	Num. of points	Time precision	Avg. sampling rate in s	Avg. total distance in m	Avg. speed in m/s
AIS	Marine	10–50m	6,842	7,544,913	s	306.583	15,407	1.297
ATC	Indoor	1–5cm	17,557	16,817,737	ms	0.042	41	1.139
GeoLife	Mixed Mobility	10–50m	18,670	24,178,077	s	5.972	67,770	5.416
Hannover	Vehicular	10–50m	1,174	617,052	s	1.098	5,351	6.001
MOPSI	Mixed Mobility	10–50m	6,776	7,602,944	ms	3.279	12,956	5.706
NOAA	Marine	10–50m	6,190,874	2,724,905,519	s	110.704	71,784	2.414
Robots	Robotic	10–50cm	100	150,000	ms	0.1	309	1.219
TrajAir	Aviation	10–50m	3,552	1,371,730	s	1	15,730	41.300

the novel techniques employed in the DimWeaver algorithm. We also compare against MLSimp [41], a recent query-driven simplification algorithm.

To ensure a fair basis for comparison, we applied a consistent evaluation process across all compression algorithms. Specifically, we calculated the compression ratio as the size of the uncompressed trajectory representation in bytes divided by the size of the compressed binary representation produced by each algorithm. For the uncompressed trajectories, we used double-precision (8 bytes) for coordinate values and long integers (8 bytes) for timestamps. In the compressed representations, all algorithms encoded coordinates using double-precision (8 bytes), while timestamps were delta-encoded and compressed using Variable-byte encoding.

We use a total of 8 trajectory datasets in our study that capture a wide range of spatial and motion characteristics: AIS [43], ATC [2], GeoLife [52–54], Hannover[55], MOPSI [33], NOAA [3], Robots [36] and TrajAir [38]. Table 1 presents the properties of our datasets, including the domain, the number of trajectories, the total number of points, the time precision, the average sampling rate, the average trajectory distance, and the average speed.

The source code of DimWeaver is publicly available[1].

5.2 Compression Ratio Comparison

In Figure 7 we report the compression ratio achieved by DimWeaver and the state-of-the-art trajectory simplification algorithms on the different datasets as we vary the maximum error threshold ϵ of the SED distance metric using the values denoted in Table 1. As expected, CISED-W is the best-performing algorithm among prior methods. TD-TR outperforms both SQUISH-E and VBT due to its batch processing capability. DimWeaver consistently emerges as the most space-efficient algorithm. Depending on the dataset and error threshold, its compression ratio is between 1.42 (Hannover with $\epsilon = 4 \times 10^{-3}$) and 2.51 (Robots with $\epsilon = 2 \times 10^{-3}$) times higher than that of the widely used TD-TR algorithm, and it outperforms the previously best-performing CISED-W algorithm by up to 107% (Robots with $\epsilon = 2 \times 10^{-3}$) with an impressive average improvement of 50% for $\epsilon = 10^{-3}$. Unlike NoWeaver, which applies decoupling alone, DimWeaver's advanced optimizations yield much higher compression ratios, reaching up to x1.72 (Robots with $\epsilon = 10^{-3}$) in the results shown in Figure 7. This underscores the effectiveness of the novel techniques introduced by DimWeaver that set it apart from our baseline approach.

This consistent advantage of DimWeaver across diverse datasets and varying ϵ values highlights its robustness and adaptability. Clearly, DimWeaver represents a significant advancement in compression ratio compared to the state-of-the-art in trajectory compression. These improvements translate into substantial storage or data transmission savings, which can be critical in large-scale spatiotemporal applications.

[1]https://github.com/xkitsios/DimWeaver

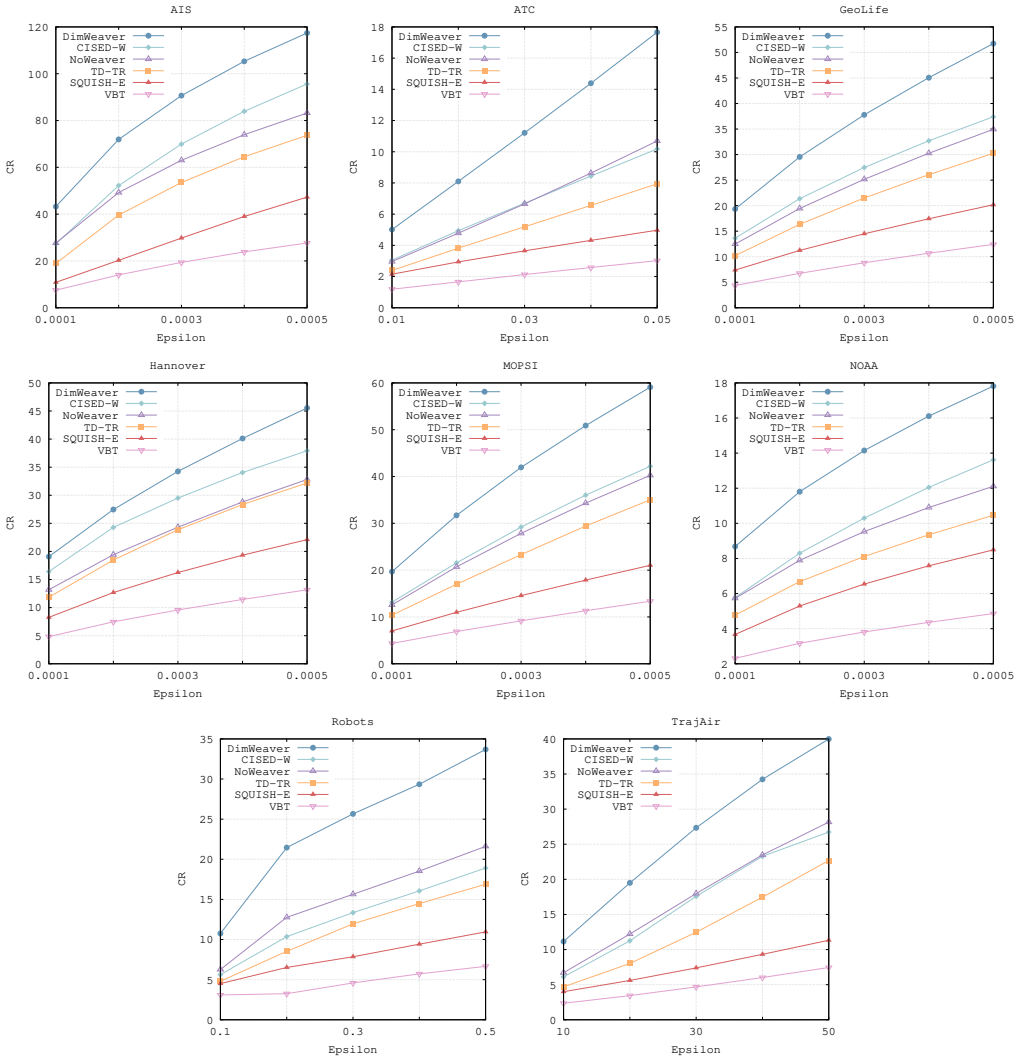
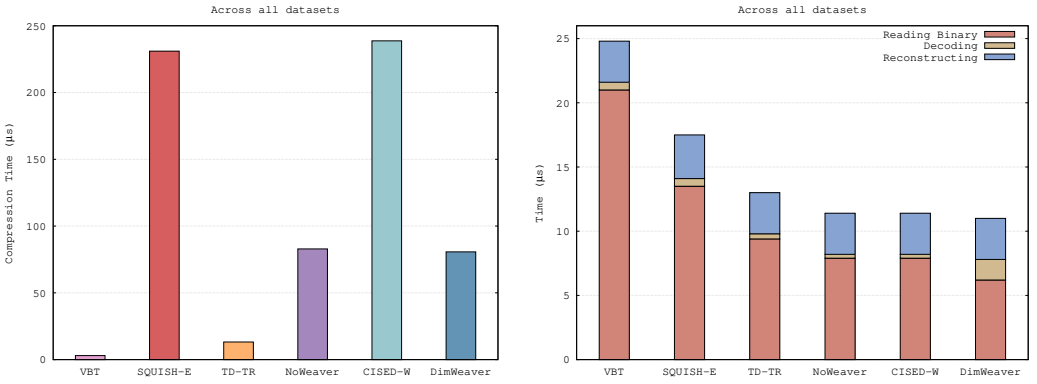


Fig. 7. Compression Ratio Results

5.3 Trajectory Compression Time

Next, we compare the time each algorithm requires to compress the trajectories of our dataset and report respective results in Figure 8a. We executed the compression process ten times over the entire data collection. In each run, we computed the average compression time per trajectory. To summarize these results, we report the median of these averages. We can see in Figure 8a that DimWeaver stands out by providing a very high compression speed, being 2.9x faster than SQUISH-E and 3.0x faster than CISED-W, the second best approach in terms of compression ratio. VBT excels in compression speed thanks to the simplicity of its velocity-estimation procedure. However, this design choice also limits its ability to reduce data volume, resulting in the lowest compression ratio among the evaluated methods, as illustrated in Figure 7. DimWeaver delivers an excellent balance



(a) Trajectory Compression Times.

(b) Trajectory Decompression Time Breakdown.

Fig. 8. Compression and decompression performance.

between runtime and compression effectiveness, thus solidifying its position as the most efficient and effective overall solution.

Both NoWeaver and DimWeaver employ an efficient segmentation process that enhances compression speed. However, DimWeaver outperforms the simpler NoWeaver algorithm due to optimizations that produce fewer segments and consequently enable faster compression. As a result, the kinematic grouping performed in the later stage of the algorithm benefits from the accelerated segmentation, accounting for only 2.4% of the total execution time. This leads to an overall faster execution of DimWeaver compared to NoWeaver.

5.4 Trajectory Decompression Time

Regenerating the trajectory data involves two steps: i) reading and decoding the compressed binary from disk into a machine-readable format and ii) reconstructing the complete list of locations. The first step occurs at fetch time while the second step corresponds to query answering time. Figure 8b shows that DimWeaver achieves the fastest overall regeneration among all algorithms, due to its smaller binary footprint. Although decoding in DimWeaver is slightly more involved than in the other algorithms —requiring the parsing of grouped segments— it remains a linear-time process, since the timestamps within each group are stored in sorted order, allowing segments to be decoded in time order with linear complexity.

The total runtime for all algorithms is clearly dominated by disk (SSD) read operations. In this context, DimWeaver's superior compression ratio directly translates into faster disk reads, which more than offsets its modest decoding cost. Remarkably, DimWeaver completes trajectory decoding before competing methods have even finished reading their data, clearly demonstrating the practical benefits of its compact representation. Finally, during the query reconstruction stage, all algorithms exhibit equivalent behavior, which is expected as this stage depends only on the number of points rather than the compression method.

5.5 Quality of Reconstructed Trajectories

Figure 9 plots the Root Mean Squared Error (RMSE) per reconstructed point for all eight datasets against the compression ratio achieved by each algorithm. The RMSE is computed using the SED distance (Equation 1) of the reconstructed points. SQUISH-E, TD-TR, NoWeaver and CISED-W follow comparable trade-off patterns between compression efficiency and accuracy. VBT exhibits lower

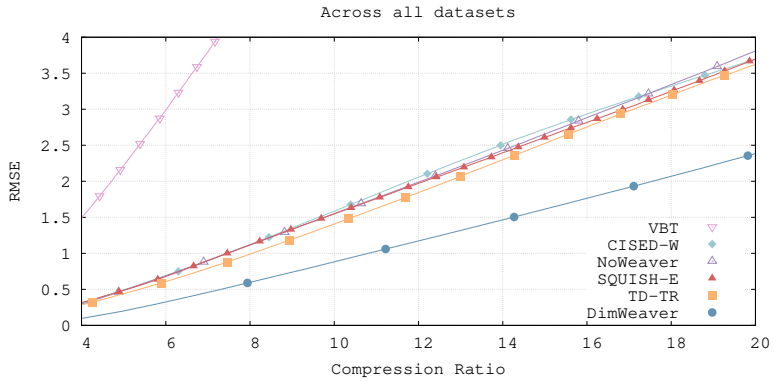


Fig. 9. RMSE of reconstructed trajectory points versus the achieved compression ratio (all datasets). For the same compressed size, DimWeaver produces significantly more accurate reconstruction of the original trajectories.

reconstruction accuracy relative to the other methods, which appears to stem from its intentionally lightweight position-estimation strategy.

DimWeaver, achieves superior compression ratios while maintaining the lowest RMSE values across all methods for the same output size. This advantage arises due to DimWeaver's use of several novel techniques, including kinematic grouping, that enable more compact representations while also preserving the accuracy of the trajectory segments formed. As a result, DimWeaver delivers the best balance of compression efficiency and reconstruction quality among all evaluated algorithms.

5.6 Compression of 3D trajectories

In Table 2, we evaluate the performance of DimWeaver on the TrajAir dataset for $\epsilon = 50m$ when the algorithm is extended to operate in three dimensions, by incorporating the Z coordinate. The 3D variant achieves a modest improvement in compression ratio, reflecting the additional flexibility for redistributing residual errors across the expanded spatial domain. Notably, this holds despite the fact that the initial error allocation entails distribution of tolerance across three ($\frac{\epsilon}{\sqrt{3}}$), instead of two ($\frac{\epsilon}{\sqrt{2}}$) dimensions (see Sections 4.2, 4.8). As expected, compression times increase due to the computational overhead introduced by processing the extra dimension but remain in the order of microseconds, for a full trajectory.

Table 2. Comparison of 2D/3D Compression for the TrajAir dataset by DimWeaver

		2D	3D
Compression Ratio		40.0	41.8
Times	Compression	27.5 μs	40.5 μs
	Reading Binary	1.7 μs	2.5 μs
	Decoding	0.6 μs	0.9 μs
	Reconstruction	1.3 μs	1.5 μs

5.7 Comparison with Query-Driven Approaches

Recently, ML-based methods such as MLSimp have been proposed for trajectory simplification. We used the official implementation and instructions to train and evaluate MLSimp on the provided

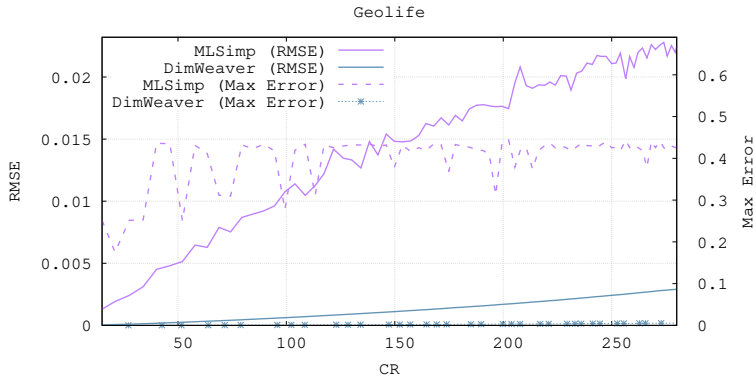


Fig. 10. Comparison against MLSimp [41], a recently proposed query-driven simplification method, on RMSE and maximum error.

subset of the Geolife dataset, consisting of 2,000 training, 1,000 validation and 2,000 testing trajectories.[2] To ensure training finished within a reasonable time frame, we employed a workstation with an Intel Xeon Silver 4210R CPU, 128GB RAM, and an NVIDIA RTX A5000 24GB GPU, which completed the task in 21 hours. Training is an inherent cost of ML-based methods, which must learn the spatial distribution of trajectories. In contrast, DimWeaver is streaming and data-agnostic, requiring no training.

Using the same workstation, we evaluated both algorithms over repeated runs and varying compression ratios. MLSimp offers no explicit approximation-error bound, and this absence manifests as strong instability. Figure 10 shows large fluctuations and significantly higher maximum SED error and RMSE for MLSimp. At comparable compression ratios, these errors can be up to three orders of magnitude larger than those of DimWeaver, which remains consistently far more accurate. This improved accuracy of DimWeaver at the same compression ratio is expected to translate into better performance in downstream analytical tasks, since much smaller simplified trajectories can achieve accuracy comparable to that of the original trajectory.

Even when excluding the 21-hour training phase for MLSimp, its simplification time remains substantially higher than that of DimWeaver. For a compression ratio of 20, MLSimp requires 15.05s to process the testing dataset of 2,000 trajectories, while DimWeaver completes the same task in 0.24s, yielding a speedup of roughly 63x. We note that the trajectories used in [41] are capped at 1,000 points each, a constraint dictated by the underlying neural network architecture.

5.8 Online Extension of Simplified Trajectories

Figure 11 depicts a scenario in which an already simplified trajectory is continuously updated using DimWeaver (denoted as Continuous), without reprocessing the entire trajectory, by leveraging the extensions introduced in Section 4.8. We use a sample of long vessel trajectories from the NOAA dataset and iteratively append to each trajectory the data points collected within the most recent 60-minute window, covering a period of six months. For comparison, we report the compression ratio achieved by DimWeaver when reprocessing the full stream of trajectory points. We observe only a minor loss in compression performance for the incrementally updated trajectory. The re-entrant invocation of kinematic grouping imposes no overhead in execution time, as it runs in parallel while DimWeaver processes new points.

[2]https://github.com/yumengs-exp/MLSimp

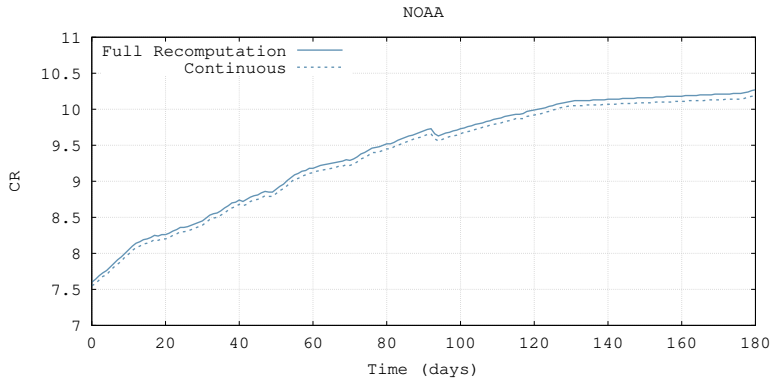


Fig. 11. Comparison of compression ratios for continuous DimWeaver updates and full recomputation on NOAA vessel trajectories.

6 Related Work

According to recent studies [27, 47], trajectory simplification and compression algorithms can be categorized based on 3 major criteria: (i) the quality metric used to measure how close the simplified trajectory resembles the original one, including Synchronized Euclidean Distance (SED), Perpendicular Euclidean Distance (PED), Direction-aware Distance (DAD) or modified versions of them [5], (ii) the problem formulation constraints: memory bounded vs error bounded compression methods, (iii) the operational mode of the algorithm: batch, online or one-pass. Beyond these criteria, the survey of [28] distinguishes compression approaches by whether they operate in free space (line simplification) or in map-constrained settings (map-matching). In the latter case, trajectories are projected onto road networks and stored as road-segment sequences [28].

Batch algorithms assume the complete history of collected trajectories is available beforehand and usually make multiple passes over the trajectories. Techniques operating in online mode consider streaming setups where only a recent window of points forming a trajectory is available and the goal is to maintain the most important points when buffer size is limited. One-pass techniques, apply local distance checking policies, without using any window and process each trajectory point once in a single linear traversal of the trajectory as new points arrive.

Detailed descriptions and experimental comparisons of the techniques of each category can be found in [27, 47]. The advantage of SED is that it accounts for both spatial and temporal alignment, preserving each dimension more faithfully during trajectory simplification [26]. On the contrary, PED and DAD primarily preserve the spatial distance and direction of movement, respectively. DimWeaver falls in the category of one-pass, error-bounded, SED-based, line simplification approaches [27, 28, 47]. As such, we focus on techniques that meet at least two out of these three criteria, discussing error-bounded, SED-based, line simplification algorithms [47].

The Douglas–Peucker (DP) algorithm [9] is a classic trajectory simplification method with a guaranteed error bound. It recursively selects the point with the maximum PED from the segment's endpoints and splits the trajectory if this deviation exceeds ϵ ; otherwise, only the endpoints are kept. Although its worst-case complexity is $O(n^2)$, DP is typically fast due to limited recursion and simple computations. Top-Down Time-Ratio (TD-TR) [34] extends DP by using SED instead of PED. This yields higher accuracy and makes TD-TR one of the most effective batch simplification algorithms.

SQUISH-E [35] operates in an online environment using a fixed-size priority queue. For each point, it maintains a score that serves as an upper bound on the SED for its neighboring points. This mechanism accounts for the fact that, when a point is removed, its associated error is propagated to its two adjacent points. BQS and FBQS [29, 30] are representative online, line simplification methods in resource-constraint environments [27, 28, 47]. However, BQS and FBQS constitute PED-, instead of SED-, based approaches [47]. Additionally, the experimental study of [47] shows that SQUISH-E outperforms [29, 30] speed-wise.

CISED [26] is a one-pass trajectory simplification algorithm with two variants: CISED-W for weak simplification and CISED-S for strong simplification. CISED introduces the concept of a spatiotemporal cone, which extends the sector intersection method [4, 40, 45, 49], originally applied in the 2D space, into the spatiotemporal domain. With a linear time complexity of $O(n)$ and impressive compression ratio results, CISED is considered the state-of-the-art algorithm for one-pass trajectory simplification [27]. In our evaluation, we utilized on CISED-W, as it consistently achieves significantly better compression ratios compared to CISED-S.

Many simplification methods estimate a 2D velocity vector to decide which trajectory points to retain. Vector-Based Tracking [6, 7] predicts future positions as a linear function of time defined by an initial point and a velocity vector. Similar ideas appear also in [39, 42, 46]. Our approach decouples motion into independent 1D components and tracks linearities per dimension. Unlike vector-based techniques which fix the velocity estimate for an entire segment, DimWeaver updates this estimate continuously as points are processed, yielding a space-optimal segmentation [37]. Moreover, instead of committing to a single velocity value, DimWeaver maintains a range of admissible velocities, which can be merged seamlessly across space and time, providing a richer and more robust characterization of recurring kinematic patterns than standard 2D velocity-based methods.

The work of [7] introduces Segment-Based Tracking, where future positions are represented by a constant-speed traversal along a road-network segment. A $O(n^3)$ algorithm consolidates consecutive road segments in a network of size n, thereby reducing the number of transmissions required as a client moves across segment boundaries. While this line of work is intuitive and effective within its intended domain, the underlying assumptions and problem setting differ fundamentally from those considered in our work. First, it presumes the existence of an explicit road network that constrains the feasible positions of the moving object. Second, the proposed segment-consolidation algorithm operates on the network topology itself rather than on the observed trajectories. Third, the method effectively concatenates consecutive road segments. In contrast, our kinematic grouping strategy operates directly on projected 1D trajectories, naturally extends to 3D, runs in $O(mlogm)$ for grouping m segments and requires no underlying network. It also supports optimal space-wise grouping [13] of non-consecutive segments across dimensions based on kinematic 1D similarity, as illustrated in Figure 1. Overall, we note that map-matching-based compression [28] follows an orthogonal design, leveraging road-network structure to store trajectories as compact road-edge sequences. Representative examples include referential, real-time compression of streaming road-network trajectories (e.g., TRACE [21]) and uncertainty-aware road-network compression [23].

Query-driven trajectory simplification (QDTS) represents a line of work that is conceptually distinct from the distance-based approaches studied here. QDTS methods aim to preserve the outcomes of downstream queries—typically similarity search or kNN—so that results obtained on simplified trajectories closely match those on the original data [44]. MLSimp [41] is a recent QDTS framework that combines a Graph Neural Network (GNN) for point-importance estimation with a diffusion model for generating simplified trajectories. Methods such as MLSimp are designed to optimize query fidelity rather than to provide explicit geometric error guarantees, and therefore they do not offer the strong, deterministic bounds required in distance-based simplification. Furthermore, ML-based approaches inherently rely on learning the spatial distribution of trajectories;

as a consequence, they require large training corpora, costly preprocessing, and global knowledge of the dataset [44]. In contrast, our DimWeaver algorithm is a streaming, distribution agnostic simplification method. It processes each trajectory independently, without any training phase or assumptions about the underlying data distribution or access to global information. This design enables DimWeaver to deliver provable error guarantees while maintaining extremely low computational cost. Another practical distinction is that GNN architectures require long trajectories to be segmented into fixed-size chunks for encoding, which can introduce additional complexity and potential artifacts. DimWeaver, by contrast, imposes no constraints on trajectory geographic spread (i.e., point locations are not confined to a predefined region), nor on trajectory length.

The work of [22] extends the density-based clustering approach DBSCAN [11] to moving objects by evaluating the consistency of clusters before and after smoothing object locations. The approach provides valuable insights for trajectory-driven analytics and could benefit from decentralized cluster-formation mechanisms [8, 12]; however, it is oriented toward a different class of problems.

Finally, a separate line of work expresses the overall admissible distortion into explicit spatial ϵ_{space} and temporal tolerance ϵ_{time}. Recent frameworks such as REST [50] enforce separate, user-specified spatial and temporal deviation thresholds. REST adopts *MaxDTW*, the maximum distance among all matched pairs under the best warping alignment, and, when needed, applies explicit time correction to satisfy ϵ_{time} [50]. For DimWeaver, the idea of splitting tolerance into $\epsilon_{\text{space}} = (\epsilon_x, \epsilon_y)$ and ϵ_{time} can, in principle, be achieved by maintaining incremental threshold checks for both bounds. However, this requires extending the strict SED setting with an explicit temporal deviation constraint or a joint spatio-temporal distance, and may involve alternative time-aware error measures beyond SED (e.g., Time Synchronized Velocity Error [15]).

7 Conclusions

In this work, we built upon the idea of decoupling trajectory dimensions to exploit kinematic linearities in *unconstrained* object movement in 2D or 3D during trajectory simplification. Although decoupling alone does not improve the current state-of-the-art, it serves as a key enabler for further optimizations explored in our study. Building on this foundation, we introduced a novel trajectory compression method, DimWeaver, which leverages recurring velocity patterns within and across one-dimensional projections resulting from the decoupling process. Additionally, DimWeaver intelligently interleaves processing across these projections, enabling greater error tolerance when needed, in the dimension most sensitive to approximation, thereby enhancing compression efficiency. Experimental results demonstrate that DimWeaver significantly outperforms top-performing methods from the literature in terms of space-efficiency, speed and approximation quality.

References

[1] Azim Afroozeh, Leonardo X. Kuffo, and Peter A. Boncz. 2023. ALP: Adaptive Lossless floating-Point Compression. *Proc. ACM Manag. Data* 1, 4 (2023), 230:1–230:26. doi:10.1145/3626717

[2] Dražen Brščić, Takayuki Kanda, Tetsushi Ikeda, and Takahiro Miyashita. 2013. Person Tracking in Large Public Spaces Using 3-D Range Sensors. *IEEE Transactions on Human-Machine Systems* 43, 6 (2013), 522–534. doi:10.1109/THMS.2013.2283945

[3] Bureau of Ocean Energy Management, National Oceanic and Atmospheric Administration, and U.S. Coast Guard Navigation Center. [n. d.]. MarineCadastre.gov Vessel Traffic Data (AIS Data: 2024 – June 2025). Available at https://hub.marinecadastre.gov/pages/vesseltraffic. Accessed: 2026-01-10.

[4] W. S. Chan and F. Y. L. Chin. 1996. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry & Applications* 6, 1 (1996), 59–77. doi:10.1142/S0218195996000058

[5] Minjie Chen, Mantao Xu, and Pasi Franti. 2012. A Fast $O(N)$ Multiresolution Polygonal Approximation Algorithm for GPS Trajectory Simplification. *Trans. Img. Proc.* 21, 5 (May 2012), 2770–2785. doi:10.1109/TIP.2012.2186146

[6] Alminas Civilis, Christian S. Jensen, Jovita Nenortaite, and Stardas Pakalnis. 2004. Efficient Tracking of Moving Objects with Precision Guarantees. In *1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services, 22-25 August 2004, Cambridge, MA, USA*. IEEE Computer Society, 164–173. doi:10.1109/MOBIQ.2004.1331723

[7] Alminas Civilis, Christian S. Jensen, and Stardas Pakalnis. 2005. Techniques for Efficient Road-Network-Based Tracking of Moving Objects. *IEEE Trans. Knowl. Data Eng.* 17, 5 (2005), 698–712. doi:10.1109/TKDE.2005.80

[8] Antonios Deligiannakis, Yannis Kotidis, and Nick Roussopoulos. 2007. Dissemination of compressed historical information in sensor networks. *VLDB J.* 16, 4 (2007), 439–461. doi:10.1007/s00778-005-0173-5

[9] David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica* 10, 2 (1973), 112–122. doi:10.3138/FM57-6770-U75U-7727

[10] Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. 2009. Online Piece-wise Linear Approximation of Numerical Streams with Precision Guarantees. *Proc. VLDB Endow.* 2, 1 (2009), 145–156. doi:10.14778/1687627.1687645

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*. AAAI Press, 226–231. http://www.aaai.org/Library/KDD/1996/kdd96-037.php

[12] Nikos Giatrakos, Yannis Kotidis, Antonios Deligiannakis, Vasilis Vassalos, and Yannis Theodoridis. 2010. TACO: tunable approximate computation of outliers in wireless sensor networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*. ACM, 279–290. doi:10.1145/1807167.1807199

[13] Udaiprakash I. Gupta, D. T. Lee, and Joseph Y.-T. Leung. 1982. Efficient algorithms for interval graphs and circular-arc graphs. *Networks* 12, 4 (1982), 459–467. doi:10.1002/net.3230120410

[14] John Hershberger and Jack Snoeyink. 1992. *Speeding Up the Douglas-Peucker Line-Simplification Algorithm*. Technical Report. CAN.

[15] Haibao Jiang, Dezhi Han, Han Liu, Jiuzhang Han, and Wenjing Nie. 2021. Time Synchronized Velocity Error for Trajectory Compression. *CMES - Computer Modeling in Engineering and Sciences* 130, 2 (2021), 1193–1219. doi:10.32604/cmes.2022.017663

[16] Bingqing Ke, Jie Shao, and Dongxiang Zhang. 2017. An Efficient Online Approach for Direction-Preserving Trajectory Simplification with Interval Bounds. In *18th IEEE International Conference on Mobile Data Management, MDM 2017, Daejeon, South Korea, May 29 - June 1, 2017*. IEEE Computer Society, 50–55. doi:10.1109/MDM.2017.17

[17] Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. 2013. Map-matched trajectory compression. *J. Syst. Softw.* 86, 6 (2013), 1566–1579. doi:10.1016/J.JSS.2013.01.071

[18] Xenophon Kitsios, Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2023. Sim-Piece: Highly Accurate Piecewise Linear Approximation through Similar Segment Merging. *Proc. VLDB Endow.* 16, 8 (2023), 1910–1922. https://www.vldb.org/pvldb/vol16/p1910-liakos.pdf

[19] Xenophon Kitsios, Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2024. Flexible grouping of linear segments for highly accurate lossy compression of time series data. *VLDB J.* 33, 5 (2024), 1569–1589. doi:10.1007/S00778-024-00862-Z

[20] Ralph Lange, Frank Dürr, and Kurt Rothermel. 2011. Efficient real-time trajectory tracking. *VLDB J.* 20, 5 (2011), 671–694. doi:10.1007/S00778-011-0237-7

[21] Tianyi Li, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen. 2021. TRACE: Real-time Compression of Streaming Trajectories in Road Networks. *Proc. VLDB Endow.* 14, 7 (2021), 1175–1187. doi:10.14778/3450980.3450987

[22] Tianyi Li, Lu Chen, Christian S. Jensen, Torben Bach Pedersen, Yunjun Gao, and Jilin Hu. 2022. Evolutionary Clustering of Moving Objects. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2399–2411. doi:10.1109/ICDE53745.2022.00225

[23] Tianyi Li, Ruikai Huang, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen. 2020. Compression of Uncertain Trajectories in Road Networks. *Proc. VLDB Endow.* 13, 7 (2020), 1050–1063. doi:10.14778/3384345.3384353

[24] Panagiotis Liakos, Katia Papakonstantinopoulou, Thijs Bruineman, Mark Raasveldt, and Yannis Kotidis. 2024. How to Make your Duck Fly: Advanced Floating Point Compression to the Rescue. In *Proc. 27th Int. Conf. on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*. 826–829. doi:10.48786/EDBT.2024.80

[25] Panagiotis Liakos, Katia Papakonstantinopoulou, and Yannis Kotidis. 2022. Chimp: Efficient Lossless Floating Point Compression for Time Series Databases. *Proc. VLDB Endow.* 15, 11 (2022), 3058–3070.

[26] Xuelian Lin, Jiahao Jiang, Shuai Ma, Yimeng Zuo, and Chunming Hu. 2019. One-pass trajectory simplification using the synchronous Euclidean distance. *VLDB J.* 28, 6 (2019), 897–921. doi:10.1007/S00778-019-00575-8

[27] Xuelian Lin, Shuai Ma, Jiahao Jiang, Yanchen Hou, and Tianyu Wo. 2021. Error Bounded Line Simplification Algorithms for Trajectory Compression: An Experimental Evaluation. *ACM Trans. Database Syst.* 46, 3 (2021), 11:1–11:44. doi:10.

1145/3474373

[28] Chenxi Liu, Zhu Xiao, Wangchen Long, Tong Li, Hongbo Jiang, and Keqin Li. 2025. Vehicle Trajectory Data Processing, Analytics, and Applications: A Survey. *ACM Comput. Surv.* 57, 9, Article 231 (May 2025), 36 pages. doi:10.1145/3715902

[29] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, and Raja Jurdak. 2015. Bounded Quadrant System: Error-bounded trajectory compression on the go. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*. IEEE Computer Society, 987–998. doi:10.1109/ICDE.2015.7113350

[30] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, Jae-Gil Lee, and Raja Jurdak. 2016. A Novel Framework for Online Amnesic Trajectory Compression in Resource-Constrained Environments. *IEEE Trans. Knowl. Data Eng.* 28, 11 (2016), 2827–2841. doi:10.1109/TKDE.2016.2598171

[31] Cheng Long, Raymond Chi-Wing Wong, and H. V. Jagadish. 2013. Direction-Preserving Trajectory Simplification. *Proc. VLDB Endow.* 6, 10 (2013), 949–960. doi:10.14778/2536206.2536221

[32] Antonios Makris, Ioannis Kontopoulos, Panagiotis Alimisis, and Konstantinos Tserpes. 2021. A Comparison of Trajectory Compression Algorithms Over AIS Data. *IEEE Access* 9 (2021), 92516–92530. doi:10.1109/ACCESS.2021.3092948

[33] Radu Mariescu-Istodor and Pasi Fränti. 2017. Grid-Based Method for GPS Route Analysis for Retrieval. *ACM Trans. Spatial Algorithms Syst.* 3, 3, Article 8 (Sept. 2017), 28 pages. doi:10.1145/3125634

[34] Nirvana Meratnia and Rolf A. de By. 2004. Spatiotemporal Compression Techniques for Moving Point Objects. In *9th International Conference on Extending Database Technology, Heraklion, Crete, Greece, March 14-18, 2004, Proceedings*, Vol. 2992. Springer, 765–782. doi:10.1007/978-3-540-24741-8_44

[35] Jonathan Muckell, Paul W. Olsen, Jeong-Hyon Hwang, Catherine T. Lawson, and S. S. Ravi. 2014. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica* 18, 3 (2014), 435–460. doi:10.1007/S10707-013-0184-0

[36] Ourania Ntouni, Dimitrios Banelas, and Nikos Giatrakos. 2025. NeuroFlinkCEP: Neurosymbolic Complex Event Recognition Optimized across IoT Platforms. *Proc. VLDB Endow.* 18, 12 (2025), 5355–5358.

[37] Joseph O'Rourke. 1981. An On-Line Algorithm for Fitting Straight Lines Between Data Ranges. *Commun. ACM* 24, 9 (1981), 574–578. doi:10.1145/358746.358758

[38] Jay Patrikar, Brady Moon, Jean Oh, and Sebastian Scherer. 2021. Predicting Like A Pilot: Dataset and Method to Predict Socially-Aware Aircraft Trajectories in Non-Towered Terminal Airspace. arXiv:2109.15158 [cs.RO] https://arxiv.org/abs/2109.15158

[39] Michalis Potamias, Kostas Patroumpas, and Timos K. Sellis. 2006. Sampling Trajectory Streams with Spatiotemporal Criteria. In *18th International Conference on Scientific and Statistical Database Management, SSDBM 2006, 3-5 July 2006, Vienna, Austria, Proceedings*. IEEE Computer Society, 275–284. doi:10.1109/SSDBM.2006.45

[40] Jack Sklansky and Víctor M. González. 1980. Fast polygonal approximation of digitized curves. *Pattern Recognit.* 12, 5 (1980), 327–331. doi:10.1016/0031-3203(80)90031-X

[41] Yumeng Song, Yu Gu, Tianyi Li, Yushuai Li, Christian S. Jensen, and Ge Yu. 2024. Quantifying Point Contributions: A Lightweight Framework for Efficient and Effective Query-Driven Trajectory Simplification. *Proc. VLDB Endow.* 18, 2 (2024), 453–465. doi:10.14778/3705829.3705858

[42] Goce Trajcevski, Hu Cao, Peter Scheuermann, Ouri Wolfson, and Dennis Vaccaro. 2006. On-line data reduction and the quality of history in moving objects databases. In *Fifth ACM International Workshop on Data Engineering for Wireless and Mobile Access, Mobide 2006, June 25, 2006, Chicago, IL, USA, Proceedings*. ACM, 19–26. doi:10.1145/1140104.1140110

[43] Andreas Tritsarolis, Yannis Kontoulis, and Yannis Theodoridis. 2022. The Piraeus AIS dataset for large-scale maritime data analytics. *Data in Brief* 40 (2022), 107782. doi:10.1016/j.dib.2021.107782

[44] Zheng Wang, Cheng Long, Gao Cong, and Christian S. Jensen. 2024. Collectively Simplifying Trajectories in a Database: A Query Accuracy Driven Approach. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*. IEEE, 4383–4395. doi:10.1109/ICDE60146.2024.00334

[45] Charles M. Williams. 1978. An efficient algorithm for the piecewise linear approximation of planar curves. *Computer Graphics and Image Processing* 8, 2 (1978), 286–293. doi:10.1016/0146-664X(78)90055-2

[46] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha. 1999. Updating and Querying Databases that Track Mobile Units. *Distributed Parallel Databases* 7, 3 (1999), 257–387. doi:10.1023/A:1008782710752

[47] Dongxiang Zhang, Mengting Ding, Dingyu Yang, Yi Liu, Ju Fan, and Heng Tao Shen. 2018. Trajectory Simplification: An Experimental Study and Quality Analysis. *Proc. VLDB Endow.* 11, 9 (2018), 934–946. doi:10.14778/3213880.3213885

[48] Jiangong Zhang, Xiaohui Long, and Torsten Suel. 2008. Performance of compressed inverted list caching in search engines. In *Proc. of the 17th Int. Conf. on World Wide Web* (Beijing, China) *(WWW '08)*. 387–396. doi:10.1145/1367497.1367550

[49] Z. Zhao and A. Saalfeld. 1997. Linear-time sleeve-fitting polyline simplification algorithms. In *Proceedings of AutoCarto*. 214–223.

[50] Kai Zheng, Yan Zhao, Defu Lian, Bolong Zheng, Guanfeng Liu, and Xiaofang Zhou. 2020. Reference-Based Framework for Spatio-Temporal Trajectory Compression and Query Processing. *IEEE Trans. Knowl. Data Eng.* 32, 11 (2020), 2227–2240. doi:10.1109/TKDE.2019.2914449

[51] Yu Zheng. 2015. Trajectory Data Mining: An Overview. *ACM Trans. Intell. Syst. Technol.* 6, 3 (2015), 29:1–29:41. doi:10.1145/2743025

[52] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding mobility based on GPS data. In *Proceedings of the 10th International Conference on Ubiquitous Computing* (Seoul, Korea) *(UbiComp '08)*. Association for Computing Machinery, New York, NY, USA, 312–321. doi:10.1145/1409635.1409677

[53] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39. http://sites.computer.org/debull/A10june/geolife.pdf

[54] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th International Conference on World Wide Web* (Madrid, Spain) *(WWW '09)*. Association for Computing Machinery, New York, NY, USA, 791–800. doi:10.1145/1526709.1526816

[55] Stefania Zourlidou, Jens Golze, and Monika Sester. 2022. GPS Trajectory Dataset of the Region of Hannover, Germany. doi:10.25835/9BIDQXVL

Received October 2025; revised January 2026; accepted February 2026