Banana Trees for the Persistence in Time Series Experimentally

Lara Ost
Department of Computer Science
University of Vienna, Vienna, Austria
lara.ost@univie.ac.at

Sebastiano Cultrera di Montesano
ISTA (Institute of Science and Technology Austria)
Klosterneuburg, Austria
sebastiano.cultrera@ist.ac.at

Herbert Edelsbrunner
ISTA (Institute of Science and Technology Austria)
Klosterneuburg, Austria
herbert.edelsbrunner@ist.ac.at

Abstract

In numerous fields, dynamic time series data require continuous updates, necessitating efficient data processing techniques for accurate analysis. This paper examines the banana tree data structure, specifically designed to efficiently maintain persistent homology—a multi-scale topological descriptor—for dynamically changing time series data. We implement this data structure and conduct an experimental study to assess its properties and runtime for update operations. Our findings indicate that banana trees are highly effective with unbiased random data, outperforming state-of-the-art static algorithms in these scenarios. Additionally, our results show that real-world time series share structural properties with unbiased random walks, suggesting potential practical utility for our implementation.

1 Introduction

Time series are pervasive across numerous disciplines, ranging from economics and finance to environmental science and healthcare. Often, time series data is dynamic, not static; for example, wearable devices continuously monitor health metrics such as heart rate or blood sugar levels, requiring robust techniques for instant analysis and decision-making. Effective synthesis of the underlying trends in such dynamic data is crucial for predictive modeling and strategic interventions.

Within the field of topological data analysis, persistent homology [4, 11] is recognized as a powerful framework for capturing multi-scale features in complex datasets. Researchers have long considered the challenge of dynamic persistence, and the vineyard algorithm [6] was the first developed to address updates in persistence diagrams (defined in Section 2.1) as the input data evolves; see [15] for an implementation. This algorithm, while capable of handling data more general than just time series, requires time linear in the size of the input complex per update, which is restricted to swapping the order of two values. This prompts us to question whether faster processing could be achieved for one-dimensional data. To address this challenge, we implement the banana tree data structure [8], recently introduced and tailored specifically for the persistent homology of dynamic time series. Their theoretical analysis promises the processing of each update in time logarithmic in the number of items plus linear in the number of changes in the persistent diagram; compare this with the linear time require to recompute the diagram [12]. This paper investigates what these results mean in practice. We highlight some of the specific contributions:

Preprint. Under review.

- *Efficiency at scale:* we demonstrate that for large datasets, such as those containing over 10^6 items, performing a local or topological update on an unbiased random walk using banana trees is at least 100 times faster than recomputing persistence with `Gudhi` [16], the state of the art static algorithm (see Figure 3).
- *Structure and performance:* we explore how the structure of banana trees, characterized by parameters such as the number of critical items, the nesting depth of bananas, and the lengths of trails, directly impacts the running time of our algorithms. This analysis identifies the types of data best suited for efficient processing (see Figure 2).
- *Worst-case scenarios:* we construct specific examples that challenge our algorithms, and show empirically that these scenarios are rare and highly unstable, reinforcing the robustness of our approach (see Table 2).
- *Real-world data:* through analysis of three specific datasets, we illustrate that real-world data can exhibit structural properties similar to those of unbiased random walks. Preliminary evidence suggests potential for the broad applicability of banana trees in practical scenarios (see Table 3).

In software for topological data analysis, tools like the `Gudhi` library [16], Dionysus [17], and Ripser [2] are established for static datasets but require complete recomputation for updates, making them less suited for dynamic contexts. We aim for our implementation to set a new standard in the topological processing of dynamic time series.

Outline. Section 2 introduces persistent homology tailored to time series data and gives an overview of the banana trees data structure. Section 3 evaluates the performance of banana trees through experiments involving time series generated from random walks, with and without a bias. Section 4 explores three distinct types of input time series to assess the performance of banana trees: worst-case scenarios, quasi-periodic signals, and real-world data. Section 5 concludes the paper with a summary of our findings and their implications.

2 Banana Trees for Time Series

We start by introducing persistent homology, a method to discern features of data across multiple scales [10]. Forfeiting the generality of this theory, we focus on time series data. We also explain what banana trees are and how they relate to persistent homology; see [8] for details on this data structure and its algorithms.

2.1 Persistent Homology of Time Series

By a *time series* we mean a linear list of real numbers, $c_0, c_2, \ldots, c_{n-1}$, which we view as a piecewise linear map, $f\colon [0, n-1] \to \mathbb{R}$, with $f(i) = c_i$ for $0 \leq i \leq n-1$. We refer to i as an *item* and c_i its *value*. A *critical item* is a local minimum or a local maximum of this map, and all other items are *non-critical*, e.g. item i if $f(i-1) < f(i) < f(i+1)$. To simplify the discussion, we assume that the map is *generic*, by which we mean that its items have distinct values. In this case, the endpoints (items 0 and $n-1$) are necessarily critical.

The *sublevel set* of f at $t \in \mathbb{R}$, denoted $f_t = f^{-1}(-\infty, t]$, are the points $x \in [0, n-1]$ that satisfy $f(x) \leq t$. When t passes the value of a minimum from below, then the number of connected components of f_t increases by 1, and if t passes the value of a maximum from below, the number of connected components decreases by 1, unless the maximum is a endpoint, in which case the number does not change. Symmetrically, we call $f^t = f^{-1}[t, \infty)$ the *superlevel set* of f at t. Observe that f^t is the sublevel set of $-f$ at $-t$, and that the minima and maxima of $-f$ are the maxima and minima of f. Persistent homology tracks the evolution of the connected components while the sublevel set of f grows, and formally defines when a component is born and when it dies. Complementing this with the same information for the superlevel sets of f, we get what is formally referred to as *extended persistent homology*; see [7] for details. It is best constructed in two phases:

- In *Phase One*, we track the connected components of the sublevel set, f_t, as t increases from $-\infty$ to ∞. A component is *born* at the smallest value of t at which a point of the component belongs to f_t, which is necessarily a minimum. The component *dies* when it merges with another component that was born earlier, which is necessarily at a maximum in the interior of $[0, n-1]$. The *ordinary subdiagram* of f records the birth and death of every component with a point in the plane whose abscissa and ordinate are those values of t at which the component is born and dies, respectively; see Figure 1.

- In *Phase Two*, we track the connected components of the superlevel set, f^t, as t decreases from ∞ to $-\infty$. Birth and death are defined accordingly, and the components are recorded in the *relative subdiagram*.

By construction, the points in the ordinary subdiagram lie above and those of the relative subdiagram lie below the diagonal. The component born at the global minimum of f is special because it does not die during Phase One. Instead, it dies at the global minimum of $-f$, which is the global maximum of f. In topological terms, this happens because the one connected component still alive at the beginning of Phase Two dies in relative homology when its first point enters the superlevel set. This class is represented by the sole point in the *essential subdiagram*. The *extended persistence diagram* is the disjoint union of the three subdiagrams. Hence, the diagram is a multi-set of points in $\mathbb{R}^2$, and so are the three subdiagrams, unless the map is generic, in which case the diagram is a set. See the left panel in Figure 1 for an example but note that it only shows a small number of points in the ordinary subdiagram. An important property of persistence diagrams is their stability with respect to small perturbation of the input data, which was first proved in [5].

The points in the persistence diagram can be characterized using the concept of a *window* introduced in [3]: start with a rectangular frame spanned by a minimum and a maximum in the graph of f such that the minimum lies on the lower edge, and extend the frame horizontally as long as the graph does not intersect the upper edge in its interior. Call the initial frame the *mid-panel*, its extension the *in-panel*, and the symmetric extension for $-f$ the *out-panel*. As proved in [3], the min-max pair defines a point in the persistence diagram iff the graph of f reaches the lower edge in the out-panel. In this case, we call the twice extended frame a *triple-panel window*. The corresponding *double-panel window* consists of the mid-panel and the in-panel but not the out-panel. Any two double-panel windows of f are either disjoint or nested, a property not shared by the triple-panel windows. We use this property to *augment* the persistence diagram by drawing an arrow from a point to another, if the double-panel windows of the first point is nested inside the double-panel window of the other point, without any other window being nested between them. See Figure 1, which shows four double-panel windows and the corresponding points and arrows in the augmented persistence diagram. The displayed frames are indeed windows because each has an out-panel—next to the mid-panel and thus opposite to the in-panel—which extends as far as the horizontal projection of the minimum (a maximum of $-f$) to the graph of the function.

2.2 Banana Trees

We sketch the banana trees while referring to [8] for the details. Suffice to say that they are based on the Cartesian tree introduced by Vuillemin [22], which stores a list of items in order such that each path from a leaf to the root is ordered by value. This tree has a unique decomposition into paths that correspond to double-panel windows, and the banana tree splits each path into two trails representing the windows nested inside the in-panel and the mid-panel of the corresponding window.

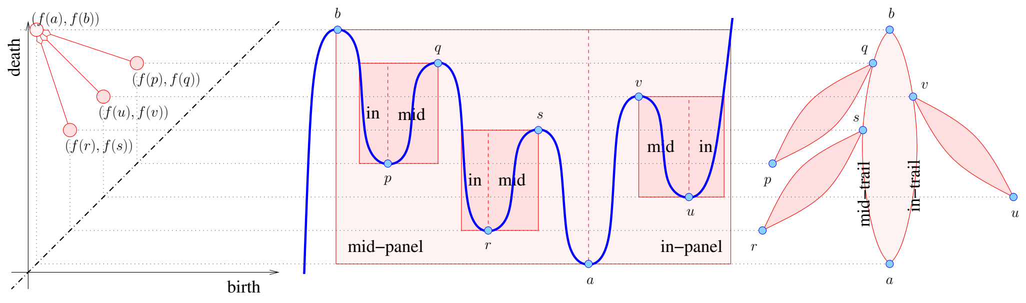

Figure 1: *Middle:* a piece of the graph of f with four double-panel windows of which three are nested inside the fourth. *Left:* the corresponding points in the persistence diagram and the arrows that reflect the nesting relation among the windows. *Right:* the four corresponding bananas in the up-tree of f.

A time series represented by a piecewise linear function, f, is stored in two banana trees together with a linked list and two standard dictionaries. We call the first banana tree the *up-tree* of f because it corresponds to Phase One of the persistence diagram construction. The second banana tree is the *down-tree* of f, which is the up-tree of $-f$. Because of this symmetry, it suffices to describe the up-tree of f. Its *leaves* are the minima of f, and its *internal nodes* are the maxima of f, with the exception of endpoint maxima, if they exist, since they are not critical in Phase One. Each point in the ordinary persistence diagram is a min-max pair, (a, b), and it is represented by a *banana* that connects the leaf a to the internal node b with two parallel trails. The *mid-trail* is a doubly-linked list connecting a to b with the maxima that span windows nested in the mid-panel of the window of a, b, and the *in-trail* is symmetrically defined; see the right panel in Figure 1. The tree structure arises because the maxima belong to the similarly defined bananas of the nested windows. Each trail is sorted in the order of value but also in the order of location along the time series. For example, the mid-trail of the big banana in Figure 1 stores a, s, q, b, which montonically increases in value and monotonically decreases in location. Compare this with the in-trail storing a, v, b, which monotonically increases in value and in location after we discard b. Indeed, we think of b as part of the min-trail while only connecting the in-trail to the rest of the tree without properly belonging to it. The global minimum of f (not shown in Figure 1) is special because it is not paired in Phase One. Indeed, it is the extra leaf whose path upward does not end at an internal node. We therefore add a *special root* as the parent of the root, connected to the global minimum by the *root banana*. While there is only one root banana, there are possibly many *leaf bananas*, which are the bananas with two empty trails (beside the minimum and maximum they connect). Indeed, we call the number of nodes between the minimum and maximum the *length* of the trail, which for trails of leaf bananas is necessarily zero. Another important concept is the *nesting depth* of a banana, which is the number of windows that contain the corresponding window. The root banana has nesting depth zero, and all other bananas have positive nesting depth.

Each update to a time series stored in a banana tree reduce to a sequence of elementary operations, which we name by their impact on the function, f. An *interchange* happens when two maxima or two minima swap the order of their values. Unless their locations are near each other, there is a good chance that an interchange does not have any effect on the banana tree. Otherwise, it is akin a rotation in a binary search tree, if the interchange is between two maxima, and akin a local rearrangement of the path decomposition, if the interchange is between two minima. A *cancellation* removes a min-max pair or, equivalently, the corresponding leaf banana, an *anti-cancellation* is the inverse of a cancellation, and a *slide* swaps the status of a critical item with that of a neighboring non-critical item. The latter three operations appear only a constant number of times whenever we adjust a value. Nevertheless, anti-cancellations present challenges to fast implementation as we have to find out where it is to happen, and this search is not supported by any special purpose data structure. On the other hand, interchanges may happen in sequence, so it is important to charge each to a change in the persistence diagram. We refer to [8] for details.

3 Experimental Results for Random Walks

This section studies the structural properties of banana trees for random walks, both biased and unbiased. In addition, it compares the running times of local and topological operations with the static algorithm in `Gudhi` [16]. We do not compare it with vineyards [6], which lack optimized software for the one-dimensional case.

3.1 The Experimental Set-up

We write $\gamma \sim \mathcal{N}(\mu, \sigma)$ for a normally distributed random variable with mean μ and standard deviation σ. Given μ, σ, a *random walk* of length n is a real-valued function $r = r_{\mu,\sigma}$ defined by

$$r(0) = 0 \;\text{ and }\; r(i) = r(i-1) + \gamma_i \;\text{ for } 1 \leq i < n, \tag{1}$$

in which the $\gamma_i \sim \mathcal{N}(\mu, \sigma)$ are independent. The random walk is *unbiased* if $\mu = 0$ and *biased* if $\mu \neq 0$. We assume $\sigma = 1$ unless stated otherwise.

To evaluate the performance of local updates, we change the value of a single item, and since it depends on the amount, we fix parameters $\Delta \in \mathbb{R}$ and $k \in \mathbb{N}$ and change the value of the i-th item from $r(i)$ to $r(i) + \delta_j$, with $\delta_j = \frac{j}{k}\Delta$ for $-k \leq j \leq k$. For the experiments, we pick $\Delta = 5.0$ or

50.0 and $k = 10$, while doing the update on a random walk of length between 10^2 and 10^6 generated for $\mu \in [-1.1]$ and $\sigma = 1$. Each experiment is repeated one hundred times and averages as well as deviations from the average are reported. For each experiment, we also measure the time to run `Gudhi` on the walk after the value change. This provides the appropriate reference time, since `Gudhi` needs to recompute the persistence diagram from scratch after each update.

To evaluate the performance of topological updates (split or glue), we pick a fraction, $c \in (0, 1)$, generate a random walk of length n, and then cut it into a left walk of length $\lfloor cn \rfloor$ and a right walk of length $\lceil (1-c)n \rceil$. To *split*, we first construct the banana tree of the entire random walk, which we then cut into two banana trees, one for the left and the other for the right walk. To *glue*, we first construct the banana trees of the left and right walks, which we then combine to a single banana tree for the walk of length n. Doing this for fractions $c \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, we compare the time to split with the time used by `Gudhi` to construct the persistence diagrams of the left and right walks, and the time to glue with the time used by `Gudhi` to construct the persistence diagram of the entire random walk.

3.2 Structural Properties

We focus on three structural properties of banana trees, which have direct implications for the running time of our algorithms: the *number of critical items*, the *nesting depth of bananas*, and the *lengths of trails*. Since a banana tree does not store non-critical items, its number of nodes is the number of critical items. In an unbiased random walk, the expected number of critical items is 50% of all items. This fraction decreases when the bias increases, with about 27% for $\mu = \pm 1$; see the left panel in Figure 2 for more detailed information.

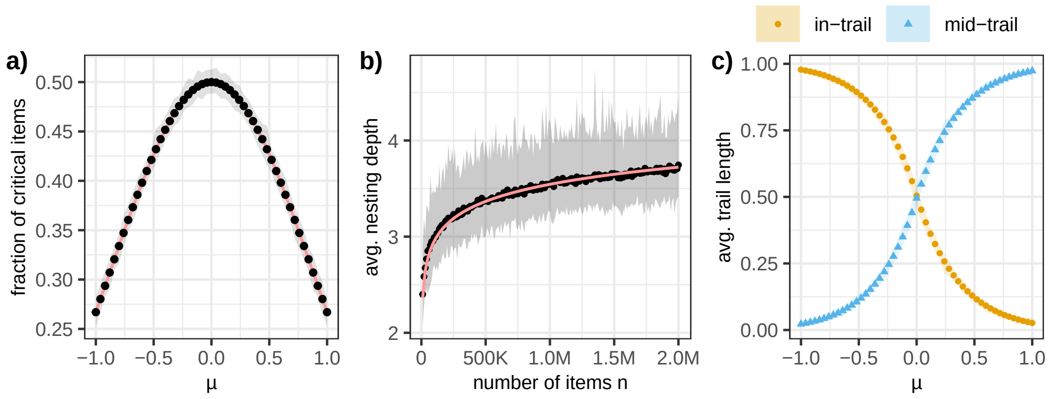

Figure 2: *Left:* the average fraction of items that are critical as a function of the bias. *Middle:* the average nesting depth of leaf bananas in banana trees of unbiased random walks with n items. The *ribbon* extends from the minimum to the maximum observed value for each n; the dots mark the mean. The *red line* is the graph of a constant times $\log n$ obtained by linear regression. *Right:* the average length of in-trails (*orange dots*) and mid-trails (*blue triangles*) in banana trees of random walks with bias μ, averaged over all input sizes.

The middle panel of Figure 2 shows the average nesting depth of a leaf banana for an unbiased random walk, which appears to scale like the logarithm of the number of items. In the unbiased case, even the maximum nesting depth seems to be small (about 13 on average for a random walk of length $2 \cdot 10^6$), and scale logarithmically in the number of items. With increasing bias, the maximum nesting depth decreases, to about 4 at $\mu = \pm 1$. This is because in the biased case, there are fewer critical items but also the bananas tend to arrange in parallel rather than inside each other. We observe that the nesting depth is about the same on average in up-trees and down-trees, both for biased and unbiased random walks, so our results hold for both trees.

Regarding trails, we observe that the length of the in-trails in the up-tree and the mid-trails in the down-tree are equal on average. By symmetry, the same holds for the mid-trails in the up-tree and the in-trails in the down-tree. Let us therefore focus on the up-tree. The right panel in Figure 2 shows how the average length of in- and mid-trails depends on the bias. In the unbiased case, both types of trails have the same length on average. For positive μ, the mid-trails tend to be longer than the in-trails, while the reverse happens for negative μ. The main cause for this trend is the longest trail, which starts to dominate the others in length as the bias increases. Assuming $\mu > 0$, the global minimum and maximum tend to lie to the left and right of the middle, respectively, with the majority of the items between them. By convention, the connecting in-trail and mid-trail contain the items to the left and right of the global minimum, respectively. This explains why the mid-trail dominates in this case, and why the in-trail dominates when $\mu < 0$. Indeed, for $\mu = \pm 1$, we observe about 95% of the internal nodes (the local maxima) on the longest trail.

3.3 Local Maintenance

We evaluate the performance of banana trees when the value of a single item changes by δ. Note that the corresponding update is a combination of some number of interchanges and at most one cancellation, at most one anti-cancellation, and at most two slides [8]. Not surprisingly, the time increases with the amount of change. For example, there is no effect at all on the persistence diagram for about 70% of the updates with $\delta = \pm 0.26$, for about 36% of the updates with $\delta = \pm 0.79$, but only about 1% for updates with $\delta = \pm 3.4$. Taking this into account, it is not surprising that banana trees are faster than `Gudhi` when $\delta = \pm 0.26$ for all lengths of random walks we tried.

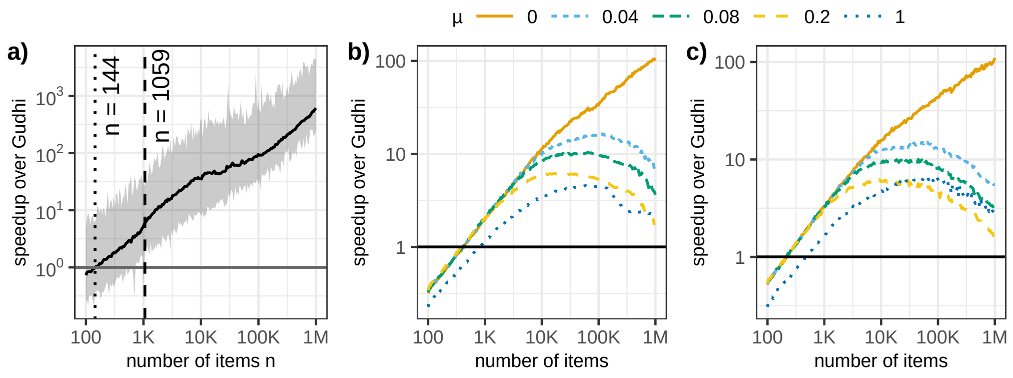

Figure 3: Comparing the maintenance of banana trees with reconstructing the persistence diagram with `Gudhi`, in which the baseline of no speedup ($10^0 = 1$) is marked with a *horizontal gray* line. *Left:* the speedup for updating a value with $\delta = \pm 5.0$ depending on the length of the random walk, n. The ribbon spans the minimum and maximum observed speedup, with the *black curve* tracing the median speedup. *Middle:* the speedup for using banana trees to cut a random walk with bias μ in half. The type and color of the curve encodes the amount of bias, and each curve shows the median speedup over a hundred repeats for each n. *Right:* the speedup for using banana trees to concatenate two equally long random walks with bias μ.

The left panel in Figure 3 focuses on the case $\delta = \pm 5.0$. As indicated by the vertical dashed and dotted lines, banana trees are faster than `Gudhi` for all random walks of length $n \geq 1059$, and faster in the median for all $n \geq 144$. Not shown in the panel is that the speedup decreases with increasing bias. In particular, for $n = 10^6$, the speedup of 612 at $\mu = 0$ decreases to 258 at $\mu = 1$, and for $n = 1059$, the speedup of 6.0 at $\mu = 0$ decreases to 2.9 at $\mu = 1$. This can be explained by recalling that for large bias the majority of the internal nodes belong to the longest trail, which connects the global minimum to the global maximum. Hence, items with similar values are likely to be near each other and thus require interchanges when an update is performed. However, even for a large change, such as for $\delta = \pm 50.0$, maintaining the banana tree is still faster than reconstructing the persistence diagram with `Gudhi`. For example, the median speedup for $n = 10^4$ exceeds 10 and for $n = 10^6$ it exceeds 100.

3.4 Topological Maintenance

We call the operations of *cutting* a list into two, and *concatenating* two lists to one topological because they change the number of lists in the overall organization of the data. We begin with evaluating the performance of cutting a random walk. The middle panel in Figure 3 shows the speedup over `Gudhi` when we cut a biased or unbiased random walk in half. In the unbiased case, the speedup increases with the length of the walk. Recalling the discussion of structural properties, this can be rationalized by noticing that the maximum nesting depth bounds the number of bananas that need to be split, and the trail length bounds the time to reorganize bananas. The maximum nesting depth scales like $\log n$ and the average trail length is only 0.5, so we anticipate logarithmic time for splitting, which we indeed observe in our experiments. Altering the position of the cut increases the speedup, namely by about 2% for $c = 0.5 \pm 0.2$ and by about 7% for $c = 0.5 \pm 0.4$.

The advantage of the banana tree over `Gudhi` diminishes when the random walks get progressively more biased. The reason is again that the larger the bias, the larger the fraction of internals nodes in the longest trail of the banana tree. Splitting the corresponding banana requires the resetting of many pointers stored in nodes along this trail, which in some cases costs time proportional to the number of critical items, which we indeed observe in our experiments. Unfortunately, this more than compensates for the low nesting depth, which guarantees that only a small number of bananas need to be split. The upper half of Table 1 shows the running time and the speedup over `Gudhi` for cutting a random walk in half.

Table 1: *Upper half*: the average time for cutting a random walk of length $n = 10^6$ in half, and the speedup over `Gudhi`. *Lower half*; the average time for concatenating two random walks of length $n = 10^6/2$ each, and the speedup over `Gudhi`. All times are measured in micro-seconds.

		$\mu = 0$	$\mu = 0.04$	$\mu = 0.08$	$\mu = 0.2$	$\mu = 1$
Cut	time	93	1417	2807	5350	1683
	speedup	105.00	7.24	3.54	1.68	2.15
Concatenate	time	90	1809	3041	5138	1228
	speedup	105.00	5.31	3.12	1.74	3.22

We finally address the reverse operation, that of concatenating two random walks or, correspondingly, of gluing two banana trees. The curves in the right panel of Figure 3 are similar to those in the middle panel, which suggests that the performance is similar to that of cutting, which is confirmed by the numbers in Table 1. For $n \leq 10^4$, gluing appears to be slightly fast than splitting, but the difference is less clear already for $n = 10^6$.

4 Special Time Series

This section considers three types of time series: *worst-case examples* to expose when banana trees underperform, *quasi-periodic data* to illustrate how banana trees reflect periodicity, and *real-world data* to demonstrate that banana trees are not just theory. While banana trees struggle for data of the first type, they mirror the performance observed in unbiased random walks for the latter two types.

4.1 Worst-case Examples

The size of the augmented persistence diagram (number of points and arrows) is proportional to the number of critical items. There are configurations in which a single update changes almost the entire diagram and thus takes time at least linear in the number of such items. We construct such *worst-case inputs* and study the performance of banana trees when confronted with such data.

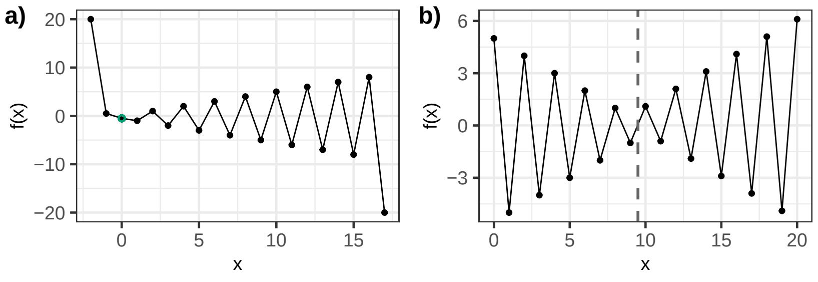

Figure 4: Worst-case examples for local and topological maintenance. *Left:* to increase the value of the marked item triggers a linear number of interchanges. *Right:* to cut the list at the *dashed line* affects every persistent pair. Both operations take time linear in the number of critical items, which for the two time series are all or almost all items.

Consider first the operation that increases the value of the marked item in the left panel of Figure 4. When its value passes that of its left neighbor, an anti-cancellation adds the banana they span to the banana tree. As we increase the value further, the marked item interchanges with the maxima to its right, each time moving up one level within the sequence of windows within which its window is nested. When the marked item finally becomes the global maximum, it will have interchanged with all other maxima, which takes $\Theta(n)$ time. For $n = 10^2$ and $n = 10^6$, we measured $3.4\mu\mathrm{s}$ and $4.7\mathrm{ms}$ on average, respectively. This corresponds to a median speedup of 0.05 and 0.02 if compared with `Gudhi`, which is really a slowdown by a factor of 20 and 50, respectively.

Table 2: Performance of splitting and gluing banana trees for the example time series in the right panel of Figure 4, showing the median slowdown/speedup if compared to `Gudhi`. The parameter interpolates between these time series at $\lambda = 0$ and unbiased random walks at $\lambda = 1$.

	split				glue			
	$\lambda = 0$	10^{-4}	10^{-2}	10^{0}	$\lambda = 0$	10^{-4}	10^{-2}	10^{0}
$n = 10^2$	0.04	0.04	0.04	0.30	0.07	0.07	0.09	0.55
$n = 10^4$	0.02	0.02	4.60	10.00	0.05	0.06	7.80	18.00
$n = 10^6$	0.04	3.60	41.00	100.00	0.05	3.70	55.00	102.00

Consider second the operation that cuts the time series in the right panel of Figure 4 in the middle, which is marked by the vertical dashed line. We observe an average running time between $14\mu\mathrm{s}$ at $n = 10^2$ and $98\mathrm{ms}$ at $n = 10^6$. Compare this with an average running time of $10\mu\mathrm{s}$ at $n = 10^2$ and $80\mathrm{ms}$ at $n = 10^6$ for concatenating the two lists back to the original length. This comparison agrees with the earlier observation that gluing banana trees is slightly faster than splitting them. Table 2 lists the speedup if compared to `Gudhi`, which for a number less than 1 is really a slowdown. To create a more informative experiment, we interpolate between these time series and unbiased walks, with drastic improvements even for small $\lambda > 0$; see Table 2. Indeed, the noise quickly flattens the banana tree by decreasing the length of trails, which explains the improvement.

4.2 Quasi-periodic Data

Random walks are not periodic, but real-world time series often exhibit some amount of periodicity, at least approximately. To study banana trees under these conditions, we construct what we call *quasi-periodic* inputs. This term does not have a mathematical definition, and for our experiments just means a periodic signal that is randomly perturbed. Specifically, we generate such input by modifying the iterative construction in (1):

$$r(0) = 0 \text{ and } r(i) = r(i-1) + \eta_i \text{ for } 1 \leq i < n, \tag{2}$$

in which the $\eta_i \sim \mathcal{N}(\mu_i, \sigma)$ are independent and normally distributed, with the mean changing periodically, $\eta_i = \sin(2\pi\omega i)$, as governed by the *frequency parameter* $0 \leq \omega < 1$, which we fix to $\omega = 5/n$. It will be useful to compare the influence of the standard deviation, so we no longer assume $\sigma = 1$.

For large σ, the banana trees of the time series show the features we have seen for unbiased random walks, while for small σ, they locally behave like biased random walks. For example, for $\sigma = 1$, we observe about 37% of the items to be critical, while for $\sigma = 0.5$ and $\sigma = 0.02$ only about 21% and 1% of the items are critical, on average. Like the number of critical items, also the maximum nesting depth for quasi-periodic signals is similar to that of random walks: it scales like $\log n$ for large σ and decreases as σ goes to 0. In constrast, the average nesting depth is independent of n, ranging from 2 to 4.7 with mean 2.4. The behavior of the trail length is illustrated in Figure 5. As shown in the left panel, the average trail length is about 0.5, with consistently shorter in-trails than mid-trails, on average. This difference is more pronounced for small σ, when the quasi-periodic signal behaves more like a biased rather than unbiased random walk. The middle and right panels show how the fraction of internal nodes on the longest trail increases as σ goes to 0.

We conclude that banana trees on quasi-periodic signals with large standard deviation resemble unbiased random walks in terms of their structural parameters, and drift toward biased random walks as the standard deviation goes to 0.

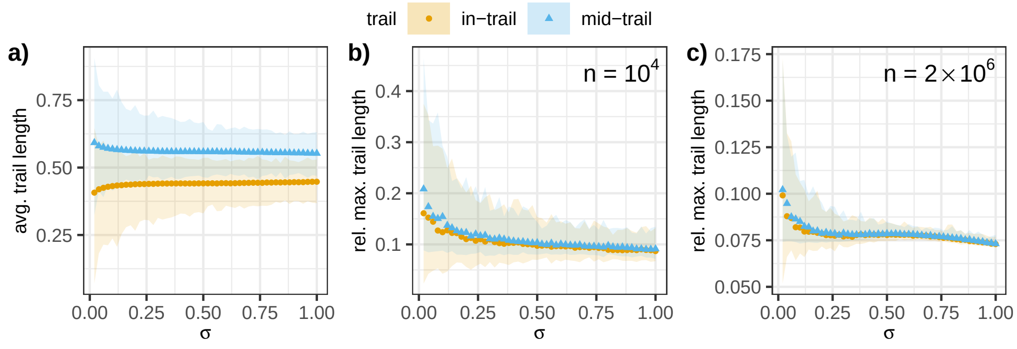

Figure 5: Measuring trail lengths. The in-trails and mid-trails are marked with *orange dots* and *blue triangles*, respectively, and the ribbons extend from the minimum to the maximum observed values. *Left:* the average length of in-trails and mid-trails depending on the standard deviation and averaged over all input sizes. *Middle and right:* the fraction of nodes on the longest in-trail and mid-trail, again depending on the standard deviation but now averaged over inputs of size 10^4 and $2 \cdot 10^6$, respectively.

4.3 Real World Examples

We analyze structural properties of banana trees constructed on three types of real-world time series:

- electrocardiography (ECG) data from PhysioNet [13, 18, 20]; limiting ourselves to recordings of length at least 10^5, each separated into up to 12 leads, gives 32443 time series in total;
- audio data from EasyCom [9], of which we use 1336 audio files;
- physical activity data from PAMAP2 [19] (temperature, heart rate along with recordings from accelerometers, gyroscopes, and magnetometers), of which we use 560 time series.

We preprocess the data by adding uniformly distributed noise to ensure all values are distinct, making sure the noise is small enough so it breaks ties but does not otherwise alter the ordering,

Table 3: Structural properties of the banana trees constructed for time series from three databases. The maximum nesting depth is measured relative to the depth we observe for unbiased random walks with the same fraction of critical items.

	fraction of	max nesting depth			average length	
	critical items	avg	min	max	in-trail	mid-trail
PhysioNet	0.24	1.2	0.8	6.3	0.48	0.52
EasyCom	0.22	2.7	1.6	4.9	0.48	0.52
PAMAP2	0.48	1.9	1.2	3.2	0.49	0.51

We observe that the average max nesting depth for the three data sets is only a small factor larger than for unbiased random walks; see the middle three columns in Table 3. The largest maximum nesting depth for PhysioNet data is a bit higher, by a factor 6.3, but 99% of the time series from this data have maximum nesting depth at most 1.68 times that for unbiased random walks. The average trail lengths are again similar to those of unbiased random walks—see the last two columns, with standard deviation about 0.05 throughout—and so are the fractions of items in the longest trails, whose geometric means are $0.8\%, 0.008\%, 0.1\%$ for the in-trails and $0.8\%, 0.008\%, 0.2\%$ for the mid-trails in the three data sets, compared with about 0.3% and 0.4% in unbiased random walks.

5 Discussion

We provide an implementation of the banana tree data structure and supply experimental evidence demonstrating its efficiency. The work reported in this paper confirms the theoretical advantages of banana trees over static alternatives and opens up avenues for further theoretical inquiries and practical applications. One intriguing question that arises in the context of quasi-periodic data is whether we can determine the (possibly varying) period of the data without prior knowledge, or provide meaningful quantification of the extent to which a signal deviates from being periodic.

The potential applications of banana trees extend beyond academic research into practical, real-world scenarios. Comparisons with other methods and identifying impactful use cases represents a promising next step to provide valuable insights in fields such as healthcare, where real-time analysis of patient data is crucial, or finance, where the swift analysis of market data is essential.

Acknowledgments and Disclosure of Funding

The first author is funded by the Vienna Graduate School on Computational Optimization, FWF project no. W1260-N35. The third author is supported by the Wittgenstein Prize, Austrian Science Fund (FWF), grant no. Z 342-N31, and the DFG Collaborative Research Center TRR 109, Austrian Science Fund (FWF), grant no. I 02979-N35.

References

[1] G. Adelson-Velsky and E.M. Landis. An algorithm for the organization of information. *Proc. of the USSR Academy of Sciences* (in Russian), **146** (1962), 263–266. English translation by M.J. Ricci in *Soviet Mathematics Doklady* **3** (1962), 1259–1263.

[2] U. Bauer Ripser: efficient computation of Vietoris–Rips persistence barcodes. *J. Appl. Comput. Topol.*, (2021), doi.org/10.1007/s41468-021-00071-5.

[3] R. Biswas, S. Cultrera di Montesano, H. Edelsbrunner and M. Saghafian. Geometric characterization of the persistence of 1D maps. *J. Appl. Comput. Topol.*, (2023), doi.org/10.100/ s41468-023-00126-9.

[4] G. Carlsson. Topology and data. *Bull. New Ser. Am. Math. Soc.* **46** (2009), 255–308.

[5] D. Cohen-Steiner, H. Edelsbrunner and J. Harer. Stability of persistence diagrams. *Discrete Comput. Geom.* **37** (2007), 103–120.

[6] D. Cohen-Steiner, H. Edelsbrunner and D. Morozov. Vines and vineyards by updating persistence in linear time. *In* "Proc. 22nd Ann. Sympos. Comput. Geom., 2006", 119-126.

[7] D. Cohen-Steiner, H. Edelsbrunner and J. Harer. Extending persistence using Poincaré and Lefschetz duality. *Found. Comput. Math.* **9** (2009), 79–103. Erratum 133–134.

[8] S. Cultrera di Montesano, H. Edelsbrunner, M. Henzinger and L. Ost. Dynamically maintaining the persistent homology of time series. *In* "Proc. 35th Ann. ACM-SIAM Sympos. Discrete Alg. 2024", 243–295

[9] J. Donley, V. Tourbabin, J.-S. Lee, M. Broyles and H. Jiang. EasyCom: an augmented reality dataset to support algorithms for easy communication in noisy environments. `arXiv:2107.04174 [cs.SD]`, 2021.

[10] H. Edelsbrunner and J.L. Harer. *Computational Topology. An Introduction.* American Mathematical Society, Providence, Rhode Island, 2010.

[11] H. Edelsbrunner, D. Letscher and A. Zomorodian. Topological persistence and simplification. *Discrete Comput. Geom.* **28** (2002), 511–533.

[12] M. Glisse. Fast persistent homology computation for functions on $\mathbb{R}$. `arXiv:2301.04745v1` [cs:CG], 2023.

[13] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P.C. Ivanov, R. Mark, J.E. Mietus, G.B. Moody, C.K. Peng and H.E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *Circulation* [online], **101** (2000), e215–e220.

[14] O. Krzikalla and I. Gaztanaga. Boost.Intrusive C++ library. (version 1.74) `www.boost.org`.

[15] Y. Luo and B.J. Nelson. Accelerating iterated persistent homology computations with warm starts. `arXiv:2108.05022` (2021).

[16] C. Maria, JD. Boissonnat, M. Glisse, M. Yvinec. The Gudhi library: simplicial complexes and persistent homology. In *Mathematical Software – ICMS 2014*, eds. H. Hong and C. Yap, LNCS **8592**, Springer, Berlin, Heidelberg, `doi.org/10.1007/978-3-662-44199-2-28`.

[17] D. Morozov. Dionysus and Dionysus 2. `mrzv.org/software/dionysus`, 2023.

[18] E.A. Perez Alday, A. Gu, A. J. Shah, C. Robichaux, A.-K. I. Wong, C. Liu, F. Liu, A. B. Rad, A. Elola, S. Seyedi, Q. Li, A. Sharma, G.D. Clifford and M.A. Reyna. Classification of 12-lead ECGs: the PhysioNet/computing in cardiology challenge 2020. *Physiol. Meas.* **41** (2020).

[19] A. Reiss. PAMAP2 physical activity monitoring. UCI machine learning repository. `doi:10.24432/C5NW2H`, 2021.

[20] M. Reyna, N. Sadr, A. Gu, E.A. Perez Alday, C. Liu, S. Seyedi, A. Shah and G.D. Clifford. Will two do? Varying dimensions in electrocardiography: the PhysioNet/computing in cardiology challenge 2021 (version 1.0.3). *PhysioNet.* (2022).

[21] D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. *J. ACM* **32** (1985), 652–686.

[22] J. Vuillemin. A unifying look at data structures. *Commun. ACM* **23** (1980), 229–239.