ParkView: Visualizing Monotone Interleavings

Thijs Beurskens[1], Steven van den Broek[1], Arjen Simons[1],
Willem Sonke[1], Kevin Verbeek[1], Tim Ophelders[1,2],
Michael Hoffmann[3], and Bettina Speckmann[1]

1 Dept. of Mathematics and Computer Science, TU Eindhoven, The Netherlands
[t.p.j.beurskens | s.w.v.d.broek | a.simons1 | w.m.sonke | k.a.b.verbeek | b.speckmann]@tue.nl

2 Dept. of Information and Computing Science, Utrecht University, The Netherlands
t.a.e.ophelders@uu.nl

3 Dept. of Computer Science, ETH Zürich, Switzerland
hoffmann@inf.ethz.ch

Abstract

We introduce ParkView: a schematic, scalable encoding for monotone interleavings on ordered merge trees. ParkView captures both maps of the interleaving using an optimal decomposition of the trees into paths. We prove several structural properties of monotone interleavings that enable a sparse visual encoding using a maximum of 6 colors for merge trees of arbitrary size.

Related Version arXiv:2501.10728

1 Introduction

A merge tree is a topological summary of a scalar field, which shows how the minima, maxima, and saddle points of the scalar field are connected (see Figure 1). The interleaving distance [4, 5, 7] is a similarity measure that captures how far two merge trees are from being isomorphic. Intuitively, it "weaves" the two trees together via two *shift maps* that take points from one tree to points a fixed distance higher in the other tree while preserving ancestry. Computing the interleaving distance is NP-hard [1] and in practice it is often desirable to introduce additional geometric constraints. The *monotone interleaving distance* [2] implements such constraints; it requires a prior ordering on the leaves of the merge trees that respects the tree structure. Given such an ordering, for example based on the spatial structure of the data, the monotone interleaving distance can be computed efficiently.

An *ordered merge tree* is a tree T equipped with a height function f and a total order on its leaves that respects T's structure. We think of T as a topological space; as such, we refer

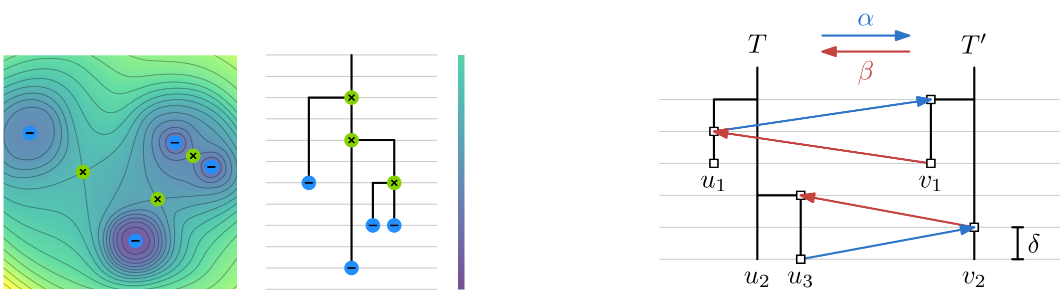

Figure 1 Left: a scalar field with its merge tree. Right: a δ-interleaving (α, β). We draw the trees rectilinearly; each horizontal line segment represents a single point, namely a non-leaf vertex.

41st European Workshop on Computational Geometry, Liblice, Czech republic, April 9–11, 2025.
This is an extended abstract of a presentation given at EuroCG'25. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

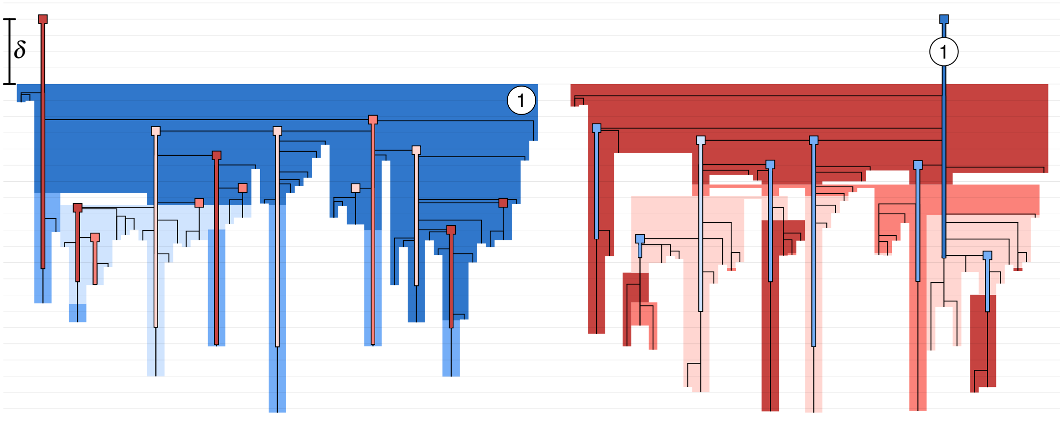

Figure 2 Example ParkView visualization of a monotone interleaving.

to not just the vertices, but also each point on the interior of an edge, as a *point* of T. The highest vertex of T is called the *root*, from which an edge extends upwards to infinity. The height function f has to be continuous and strictly increasing along each leaf-to-root path of T. A *monotone δ-shift map* α takes points in T and maps them continuously to points in T' exactly δ higher such that it preserves the order of any two points of T. A *monotone δ-interleaving* consists of two monotone δ-shift maps (α from T to T' and β from T' to T) such that for any point $x \in T$, the point $\beta(\alpha(x))$ is an ancestor of x and for any point $y \in T'$, the point $\alpha(\beta(y))$ is an ancestor of y. Figure 1 shows an example. The *monotone interleaving distance* is then the smallest δ for which a monotone δ-interleaving exists. In the remainder of this paper, we use "interleaving" to mean "monotone interleaving".

Interleavings on merge trees can have a complex structure, and hence to gain insight in their behavior, it is useful to visualize them. However, existing visualizations (e.g. [1, 3, 4, 5, 6, 7]) are mostly designed to visually explain the concept of interleavings on small examples, and not suitable for actual data exploration. We introduce ParkView: a schematic and scalable visual encoding for interleavings. To represent a shift map, ParkView decomposes the two merge trees into few components such that a component in one tree maps entirely to one component in the other tree. See Figure 2: the points in the left tree enclosed by shape 1 (a *hedge*) map to the points in the right tree on segment 1 (an *active path*). ParkView draws a merge tree rectilinearly, with the leaves drawn in separate columns according to the leaf order (Figure 3). The properties of a monotone interleaving allow us to match components

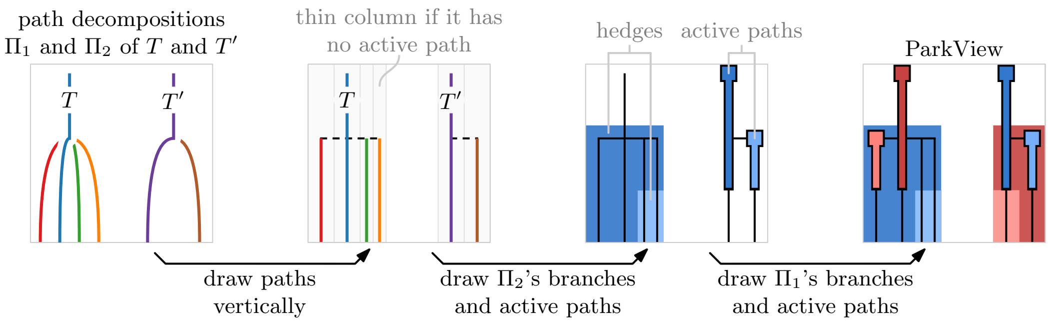

Figure 3 ParkView draws an interleaving (α, β) by superimposing drawings of heavy path-branch decomposition of both α and β.

left to right, based on the position of the lowest leaf for hedges and the x-position for active paths. Matching components are also assigned the same color. The drawings of the two shift maps combine and together show the interleaving.

In this paper, we detail two aspects of ParkView. First we define an optimal way of decomposing merge trees and show how to compute it (Section 2). Then we explain how we draw hedges and show that the set of hedges is 3-colorable (Section 3). The full version details the algorithmic pipeline for computing ParkView, and includes a showcase of ParkView on several real-world datasets.

2 Path-Branch Decomposition

The input for ParkView consists of two ordered merge trees T and T' and two shift maps α and β. We now describe the decomposition based on the shift map α; the decomposition based on β is symmetric. We decompose T' into a *path decomposition* Π: a set of height-monotone paths π that each start at a leaf (the *bottom* of π) and end at an internal vertex of T' (the *top* of π) or, for one path, at infinity. To make sure the paths of Π are disjoint and exactly cover T', we consider each path π to be open at its top. Alternatively, we can define a path decomposition bottom-up. For a vertex v of T', let the *up edge* be the one edge with increasing height incident to v, and let the *down edges* be the other edges incident to v. We now define a path decomposition by selecting, for each internal vertex v, one of the down edges of v as the *through edge* of v. The path decomposition is then built by starting a path at each leaf of T', and for each internal vertex v letting the incoming path from the through edge continue, while the incoming paths from the remaining down edges end at v.

Each path $\pi \in \Pi$ induces a *branch* B_π in T: the part of T that α maps to π. The branch B_π can either be empty, consist of a single connected component (a *simple branch*), or consist of multiple connected components (a *compound branch*) (see Figure 4). The complete set of branches B_π forms a decomposition of T, which we call the *branch decomposition* of T. Together, we call the paths in T' and the branches in T a *path-branch decomposition* for α. To minimize visual complexity, we now show how to construct an *optimal* path-branch decomposition: one that minimizes (1) the maximum number of branch components per path and (2) the total number of branch components.

As noted before, we can define a path decomposition of T' by selecting a through edge for each internal vertex v. For an edge e, let B_e be the part of T that α maps to the interior of e, and let the *weight* of e be the number of connected components of B_e. We define a *heavy path decomposition* by selecting the through edge of v to be a down edge of v with maximum weight. We now prove that a heavy path-branch decomposition is optimal. We refer to the highest edge π traverses as its *top edge*. We define the *size* of a branch B as the number of connected components it consists of. We first show that for a given path π, the size of its induced branch is equal to the weight of π's top edge.

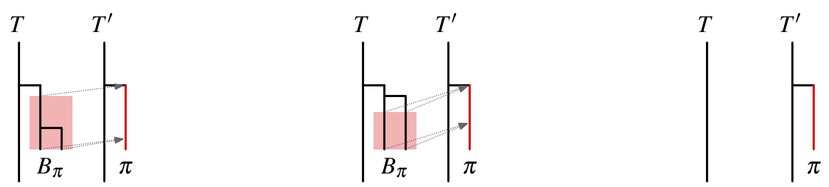

Figure 4 Examples of a simple branch, a compound branch, and an empty branch B_π.

▶ **Lemma 1.** *Let π be a path with top edge e. Then the size of B_π is equal to e's weight.*

Proof. Let v be the top of π and let $h := f(v) - \delta$. As e is in π, we have that $B_e \subseteq B_\pi$. It hence suffices to argue that each connected component C of B_π contains exactly one connected component of B_e. To show that C contains at least one connected component of B_e, we show that C contains a point x in B_e. Take any point $x' \in C$. If $\alpha(x')$ lies in the interior of e, then we take $x := x'$. Otherwise, we continuously follow the path from x' to the root of T. As α is continuous, the images of the points on the path (in T') also form a continuous path. Furthermore, as α is a δ-shift map, the images of these points also have a continuously increasing height value. It follows that there is a point x that maps to e. By definition $x \in B_e$ (and thus also in B_π). Furthermore, all points between x' and x on our path map to points on π in T'. Therefore, they are all part of B_π; hence, they are all part of the same connected component of B_π, namely C.

To show that C contains at most one connected component of B_e, assume for a contradiction that there are two distinct connected components C_1 and C_2 of B_e in C. As before, these components respectively contain points x_1 and x_2, both at height $h - \varepsilon$ for some $\varepsilon > 0$ chosen such that no vertices of T have height between h and $h - \varepsilon$. Now there is a path ρ from x_1 to x_2 entirely within C, as C is connected. There also is a distinct path ρ' from x_1 to x_2 via the lowest common ancestor x_3 in T of x_1 and x_2. Note that $f(x_3) \geq h$, so ρ' is not entirely within C; that is, $\rho \neq \rho'$. The union of ρ and ρ' hence contains a cycle, contradicting the fact that T is a tree. ◀

▶ **Theorem 2.** *Any heavy path-branch decomposition is optimal.*

Proof. Let Π be a path decomposition. Recall that Π selects one through edge for each vertex v in T'. Define the *cost* of v as the sum of the weights of v's down edges, excluding its through edge. As these edges are exactly the top edges ending at v, by Theorem 1, the cost of v is the number of branch components belonging to the paths ending at v. Then, the sum of costs of all vertices in T' is the total number of branch components induced by Π. This sum is minimized by minimizing the cost for each vertex v. This is achieved by maximizing the weight of its through edge, that is, picking a heavy edge as the through edge. A similar argument holds for minimizing the maximum number of branch components per path. ◀

3 Hedge Coloring

We represent each branch B_π by a *hedge* H_π: a rectilinear shape enclosing B_π (see Figure 5). Each hedge is a *histogram*: the union of a set of axis-aligned rectangles called *bars* whose tops are aligned. We call the height of the highest (lowest) point in a branch B_π its *top* (*bottom*) *height*. A hedge consists of three types of bars: *tree bars*, *fillers*, and *bridges*. For each path σ in the path decomposition of T that contains points in B_π, in the column of σ we add a *tree bar* whose bottom height is the height of the lowest point on σ that is in B_π.

Figure 5 The types of bars that make up a hedge (left) and the resulting hedge (right).

Figure 6 Illustrations of Observation 3 (left) and Observation 4 (right).

The union of these bars may not be connected; in this case, we connect consecutive leaves in the same branch component by adding *fillers* in the columns between them. The height of such a sequence of fillers is the smallest height of the two bars they connect (Figure 5). For a compound branch B_π, we draw its connected components like before, and then between them we add a *bridge*: a horizontal connector at the top of the hedge (Figure 5). The height of the bridge is less than the height of the shortest bar in the hedge.

A hedge H has a *left* (*right*) side which is the left (right) side of its leftmost (rightmost) bar. Two distinct hedges are *adjacent* if their boundaries, excluding corners, overlap. A hedge P is the *parent* of H if P is adjacent to the top of H; then H is a *child* of P.

It is desirable to use as few colors as possible for the hedges, while ensuring adjacent hedges have distinct colors. In fact, we show that the set of hedges in ParkView is 3-colorable. The proof makes use of three properties: hedges (i) are pairwise interior disjoint, (ii) have at most one parent, and (iii) have no hedge adjacent to the bottom of their longest bar. Our proofs of these properties rely on two observations about our drawing of T (see Figure 6).

▶ **Observation 3.** *No point of T is between two points of another branch at the same height.*

▶ **Observation 4.** *No leaves are positioned vertically above a horizontal segment.*

▶ **Lemma 5.** *Hedges in ParkView satisfy property (i).*

Proof sketch. Consider a horizontal line h that intersects a number of hedges. As hedges have a complicated shape, instead of studying the intersection of each hedge with h, we use Observation 3 to partition h into a number of interior disjoint intervals, one for each hedge. We then show that these intervals are supersets of the intersection of the corresponding hedge with h, from which it follows that the hedges are interior disjoint. ◀

▶ **Lemma 6.** *Hedges in ParkView satisfy property (ii).*

Proof sketch. For any hedge H_π, we can show that (a) it needs to have a point of T on the top, which is adjacent to some tree bar in a parent hedge, and (b) any other bars adjacent to the top of H_π need to be part of the same parent hedge. ◀

▶ **Lemma 7.** *Hedges in ParkView satisfy property (iii).*

Proof sketch. Let b be a longest bar in a hedge H_π. We can show that b is a tree bar: if it were a filler, this would violate Observation 4. We prove a key property: a tree bar that is a longest bar of its hedge has a leaf of T on its bottom. Hence, b has such a leaf. Now assume that there is another hedge H_ρ adjacent to the bottom of b. Then on the top of H_ρ, there is a point via which H_ρ connects to the rest of T. As each hedge has at most one parent (Lemma 6) this connection is via a bar b' of H_π. However, then b' is a longest tree bar. This contradicts our key property that b', being a longest tree bar, has a leaf on its bottom. ◀

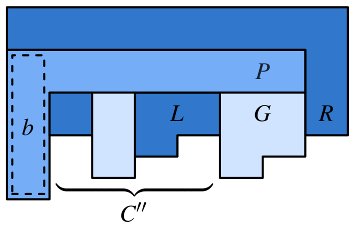

Figure 7 A set of histograms where P is the parent of G.

▶ **Theorem 8.** *Any set C of histograms that satisfies properties (i)–(iii) is 3-colorable.*

Proof. We use induction on $n = |C|$. The base case ($n = 1$) is trivial. Assume that C contains $n + 1$ histograms, and let G be a histogram whose top is lowest; it follows that no histogram in C is adjacent to the bottom side of any bar of G, and at most one histogram in C is adjacent to the left (or right) of G. Lastly, G can have at most one parent by (i), so G has at most three adjacent histograms.

The set $C' := C \setminus \{G\}$ still satisfies (i)–(iii) and has size n. By the induction hypothesis, C' is 3-colorable; fix a 3-coloring c_1 for C'. We edit c_1 into a 3-coloring for C. If the histograms adjacent to G use fewer than three colors, we use the third color for G to obtain a 3-coloring for C. Otherwise, let L and R be the histograms adjacent to the left and right of G, and let P be the parent of G. Since P, L, and R have distinct colors, we can assume without loss of generality that c_1 assigns colors 1, 2, and 3 to P, L, and R, respectively. By (iii) there is no histogram adjacent to the bottom of a longest bar of P, so P extends below the top of G. Without loss of generality, assume P extends left of G and call the rightmost such extending bar b (Figure 7). Consider the descendants C'' of P that lie to the left of G and to the right of b. As L is contained in C'', the set C'' is nonempty. This means that $C \setminus C''$ again satisfies (i)–(iii) and has size at most n, and is hence 3-colorable by the induction hypothesis. Let c_2 be a 3-coloring of $C \setminus C''$ such that without loss of generality P has color 1 and G has color 3. We now define a coloring c_3 for C where the histograms of $C \setminus C''$ take its color from c_2, and the histograms in C'' take their color from c_1.

Note that G and P are the only two histograms of $C \setminus C''$ that are adjacent to histograms in C''. So, one of four cases applies to any two adjacent histograms of C: (a) both lie in $C \setminus C''$, (b) both lie in C'', (c) one is P and the other lies in C'' or (d) one is G and the other lies in C'' (i.e., the other is L). For c_3 to be a 3-coloring, it suffices to show that in each case, c_3 assigns them distinct colors. In case (a), c_3 assigns the same colors as c_1. In case (b), c_3 assigns the same colors as c_2. In case (c), P has color 1 in both c_1 and c_2, so c_3 again assigns the same colors as c_1. In case (d), L has color 2 and G has color 3. ◀

Since hedges are histograms and satisfy (i)–(iii), the set of hedges in ParkView is 3-colorable.

Acknowledgments. Research on the topic of this paper was initiated at the 7th Workshop on Applied Geometric Algorithms (AGA 2023) in Otterlo, The Netherlands. Thijs Beurskens, Willem Sonke, Arjen Simons, and Tim Ophelders are supported by the Dutch Research Council (NWO) under project numbers OCENW.M20.089 (TB, WS), VI.Vidi.223.137 (AS), and VI.Veni.212.260 (TO).

References

1 P.K. Agarwal, K. Fox, A. Nath, A. Sidiropoulos, and Y. Wang. Computing the Gromov-Hausdorff distance for metric trees. *ACM Transactions on Algorithms*, 14(2):1–20, April 2018. `doi:10.1145/3185466`.

2 T. Beurskens, T. Ophelders, B. Speckmann, and K. Verbeek. Relating interleaving and Fréchet distances via ordered merge trees. In *Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA25)*, pages 5027–5050. Society for Industrial and Applied Mathematics, 2025. `doi:10.1137/1.9781611978322.170`.

3 J. Curry, H. Hang, W. Mio, T. Needham, and O.B. Okutan. Decorated merge trees for persistent topology. *Journal of Applied and Computational Topology*, 6(3):371–428, February 2022. `doi:10.1007/s41468-022-00089-3`.

4 E. Gasparovic, E. Munch, S. Oudot, K. Turner, B. Wang, and Y. Wang. Intrinsic interleaving distance for merge trees. *La Matematica*, pages 1–26, 2024. `doi:10.1007/s44007-024-00143-9`.

5 D. Morozov, K. Beketayev, and G. Weber. Interleaving distance between merge trees. Manuscript (accessed on 06-03-2025), 2013. URL: `https://mrzv.org/publications/interleaving-distance-merge-trees/manuscript/`.

6 M. Pegoraro. A graph-matching formulation of the interleaving distance between merge trees. `arXiv:2111.15531`.

7 E.F. Touli and Y. Wang. FPT-algorithms for computing the Gromov-Hausdorff and interleaving distances between trees. *Journal of Computational Geometry*, 13(1):89–124, April 2022. `doi:10.20382/jocg.v13i1a4`.