Supplemental Material – Merge Tree Identity

# Temporal Merge Tree Maps: A Topology-Based Static Visualization for Temporal Scalar Data

**Wiebke Köpp and Tino Weinkauf**

The following text requires the understanding of Section 3.1 from the main paper.

We show that $f$ and $s$ have the same merge tree $M$ that was used to create the linearization. We call this the *merge tree identity* property. This property is important as it guarantees that the original features are represented well and stay intact in the linearization, i.e., they are not broken up as with other linearization approaches.

We need to make one very mild assumption for this property to hold, namely that each super node has at most two children, i.e. all merge trees are binary. A super node with more than two children is a multi-saddle, meaning more than two components merge at the corresponding vertex. This behavior cannot be replicated when linearizing. If this assumption is not fulfilled for a particular data set, then the merge trees are still almost equal to one another, because only a few regular nodes are affected. Our treatment of multi-saddles through repeatedly grouping children means for a single multi-saddle with $n$ children, $(n-1)/2$ regular nodes will turn into supernodes. This models a sequence of consecutive binary merges instead of a single multi-way merge.

We show the merge tree identity using a binary join tree; it works in the same way for split trees. We need to show that the following conditions hold:

I. supernodes of the join tree of $s$ are also supernodes of the join tree of $f$,

II. regular nodes remain regular nodes, and

III. supernodes and regular nodes connect in the same way.

Recall the basics for join trees in 1D: in a one-dimensional function, each data point is adjacent to at most two neighbors. Supernodes of a one-dimensional function are its local minima and maxima. For a minimum, all neighbors are larger, for a maximum, all neighbors are smaller. A data point is a regular node if it has one neighbor with smaller and one neighbor with larger value.

*(condition I. root)* The root of a join tree is the global maximum. Our placement of it at $x = 0$ means the mapped node has only one neighbor. As all other data values are smaller, the mapped node is again a global maximum. It will be traversed last when building the join tree of $f$ and is thus again the root node.

*(condition I. leaves)* Our algorithm terminates recursion at each leaf node, placing the leaf itself in between the last two regular nodes of its parent superarc. Both regular nodes must be associated with larger data values, making the mapped node a local minimum that initiates formation of a new component during join tree construction.

*(condition I. inner supernodes)* We placed inner supernodes in between their children and continued with traversal of its two child arcs. This means the supernode is adjacent to two regular nodes of smaller value from two different superarcs. It is thus a local maximum and causes a merge of the two child components during join tree construction of $f$.

*(condition II.)* All regular nodes are placed adjacently to a node that has already been placed before and another node that will be placed later: the former node has a larger data value, the latter node has a smaller data value. Hence, these nodes are regular. The only exception is the first regular node of the

root superarc (see Figure 4(a) in the main paper), which is placed at $x = m - 1$ and has therefore only one neighbor. However, it will be recognized as a regular node as well, since the join tree construction of $f$ will have one single connected component when reaching this point and only the global root can become a supernode of that last component.

*(condition III.)* The connectivity is retained, since we placed all sample points in decreasing order and the join tree construction picks them up in increasing order. More specifically: all regular nodes are placed in the basin of their supernode to be picked up during the corresponding superarc construction. The superarcs merge in the same manner as in the input, since the corresponding supernode is placed in between merging superarcs.

In summary, this means that $s$ and $f$ have the same join tree. The same exercise can be done with the split tree. Note however, that in general the linearization mapping function $g$ does not preserve both join and split tree *at the same time.* The user will have to decide which merge tree to preserve.

We note that $f(x) = f(g^{-1}(x))$ holds, meaning $f$ is a valid linearization of itself.