A Scalable Linear-Time Algorithm for Horizontal Visibility Graph Construction Over Long Sequences

Stephen, C.

DOI 10.1109/BigData52589.2021.9671517
ISBN 978-1-6654-4599-3
ISBN 978-1-6654-3902-2

Publisher: IEEE

A Scalable Linear-Time Algorithm for Horizontal Visibility Graph Construction Over Long Sequences

Colin Stephen
Centre for Computational Science and Mathematical Modelling
Coventry University, UK
colin.stephen@coventry.ac.uk

***Abstract*—The horizontal visibility graph (HVG) representation of a time series is a structured graph whose connectivity properties have been used to study the dynamics of a wide range of nonlinear systems. Applications range from the brain (EEG), the heart (ECG) and the financial markets (bid prices), to the sun (solar intensity readings) and river flows. HVGs have also been extended to image-based pattern recognition. Efficient and scalable online HVG construction is vital to extending HVG-based time series analysis to long, streaming, and distributed real-world time series data.**

The fastest scalable method for constructing HVGs today is the binary search tree (BST) encoding–decoding algorithm, which is $O(n \log n)$ in time series length for balanced data such as noise. However, in practice BST is highly sensitive to the geometric structure of a time series and its performance degrades significantly towards $O(n^2)$ when data possess long term dependencies or when the sample frequency is high, which occur regularly in practice. To avoid these problems we leverage an $O(n)$ ordered rooted tree representation of time series that is (graph) dual to the HVG. We demonstrate that this representation leads to an algorithm for HVG construction that is agnostic with respect to the geometry and auto-correlations of the underlying data. Moreover, it possesses an efficient branch fusion operation for tree merging, leading to the idea of a *bipartite HVG* introduced in this paper, which allows HVGs for very large time series to be constructed efficiently in parallel.

After introducing our method and algorithms for parallel construction of HVGs we report on experimental benchmarks comparing their real-world performance to existing approaches on long time series. On data sampled from fractional Brownian motions, deterministic chaotic systems, brain EEG recordings, and the financial markets, our dual tree algorithms significantly outperform previous methods.

***Index Terms*—time series analysis, graph algorithms**

I. Introduction and Related Work

Recently, the promise of using powerful graph theory methods to solve problems in time series analysis and classification has led to a number of proposed maps from sequential data to graph structures: see [1] and [2] for surveys. The idea is to use well-studied local or global graph connectivity properties, such as vertex degree sequences or graph centrality measures, to infer key properties of the input time series. Methods include phase space based recurrence networks [3], visibility graphs [4]–[6], and Markov chain transition networks [7], [8].

The horizontal visibility graph (HVG) map from sequences to graphs, studied here, is particularly intuitive [9]. Given a time series $\tau = (x_1, \ldots, x_n) \in \mathbb{R}^n$ its HVG is the graph $\mathrm{HVG}(\tau)$ with vertices $\{1, \ldots, n\}$ and edges between any pair $i < j \in V$ whenever the following *horizontal visibility criterion* is satisfied:

$$\forall k \in V, \qquad i < k < j \qquad \Longrightarrow \qquad x_k < x_i, x_j. \tag{1}$$

Despite its structural simplicity, illustrated in Figure 1, this graph captures much of the geometry of τ while remaining invariant under (strictly positive) monotonic transformations to both axes: in other words, warping the time or value axis does not affect the resulting graph. As a result, its invariants capture

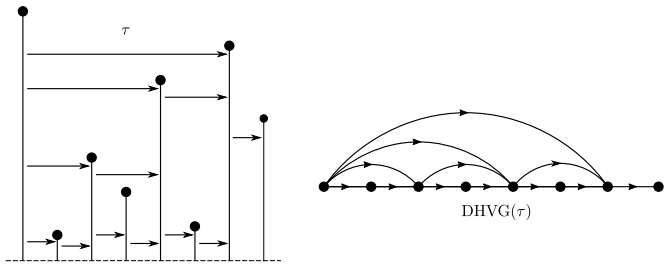

Fig. 1. A time series τ and its directed horizontal visibility graph $\mathrm{DHVG}(\tau)$.

dynamic properties of the physical system generating the time series. For example, the degrees of HVG graph vertices can be used to quantify dynamic irreversibility in nonlinear systems [10] and to estimate Lyapunov exponents and other measures of chaos [11], [12]. In stochastic physics the vertex degrees of HVGs can be used to estimate Hurst exponents to quantify long-term auto-correlations in data [13]. Empirical applications of HVGs extend from analyzing stock market dynamics [14], through pathology detection using medical sensor data including EEGs [15]–[17] and ECGs [18]–[20], to predicting river flow patterns [21], solar activity [22]–[24], turbulence in plasmas [25], optical phenomena [26], and faults in mechanical bearings [27]. Simple extensions of the HVG can also be used for pattern recognition and image classification tasks [28], or for the challenge of characterizing oil-water flow patterns [16].

Extending HVG-based time series analyses like those above into Big Data contexts motivates the search for HVG construction algorithms that are scalable. An algorithm capable of *subsequence batch processing* to create subgraphs of the HVG, followed by their recombination into a final graph, would enable parallel CPU and/or multi-node distributed speedup of the graph construction process. The following two qualities are

desirable for processing long, distributed, or streaming time series data in this way:

1) The algorithm processes each subsequence efficiently, ideally $O(n)$ in the subsequence length.
2) The algorithm combines multiple batch outputs to produce the final HVG efficiently, ideally $O(n)$ in the number of nodes in the final graph.

To date, no algorithm achieves both of these objectives.

Existing Algorithms

The original HVG algorithm implementation [29] provided in Fortran 90/95 alongside the first papers on HVG analysis [9], [30] is $O(n)$ in time series length on average, but only for noisy (stochastic/chaotic) time series. This omits many sequences of interest, such as the non-equilibrium dynamics of an oscillator damped by a non-conservative force or even a simple linear decreasing trend. For arbitrary sequences including these, its worst-case complexity is $O(n^2)$.

A 'fast weighted horizontal visibility graph' (FWHVG) algorithm was developed to help analyze EEGs for epilepsy detection [15]. This is a simple worst-case $O(n)$ algorithm for HVG construction, with edge weights added to the resulting graph to capture temporal relationships relevant to the analysis. However, the geometric information relevant to 'merging' multiple HVG subgraphs together is ignored by the FWHVG algorithm, so it cannot leverage subsequence batching and it is not inherently scalable.

More recently two algorithms were published that have the potential to help scale HVG construction to multiple parallel batch processes over subsequences [31], [32]. They are both suitable for computing the so-called natural visibility graph (NVG) as well, which extends the HVG with new edges that capture convexity relations between subsequences of the data [33]. Here, we only consider their application to the HVG.

The first is a 'divide and conquer' (DC) style algorithm [31] based on the observation that each time series τ can be decomposed into two subsequences τ_1, τ_2 around its global maximum value x_i, so that their concatenation gives $\tau = \tau_1 + (x_i) + \tau_2$. If x_i is maximal, then the horizontal visibility criterion (1) ensures that $\mathrm{HVG}(\tau)$ consists of exactly the disjoint union of those edges whose endpoints include vertex i, with the edges in $\mathrm{HVG}(\tau_1)$ and the edges in $\mathrm{HVG}(\tau_2)$. Using this decomposition, the DC algorithm recursively computes an HVG via the HVGs of its subsequences generated by maximum value splits. In the average case with balanced data such as noise this is an $O(n \log n)$ process, and in the worst case of unbalanced data such as correlated or monotonic sequences this is an $O(n^2)$ process.

Since the DC algorithm decomposes τ into subsequences it offers the possibility of distributing HVG construction over parallel processes. However, in practice the algorithm requires an entire time series to begin batch creation via splitting, so it cannot deal with streaming data, and moreover the batches cannot be fixed in size beforehand and depend entirely on the geometric characteristics of the input time series: while a noisy sequence is likely to split into roughly equal-sized subsequences, trended or correlated stochastic sequences can lead to a large imbalance in batch size across the computation, which negates the benefits of distributing the calculation. Similarly, the depth of the recursion followed by the DC approach on imbalanced data can quickly exceed the default maximum recursion depths of language interpreters, even for medium-sized sequences [34].

A second method for scalable HVG construction is the 'binary search tree' (BST) encoding-decoding algorithm [32] developed to deal with some of the shortcomings of the DC method. This technique uses a novel online process to build a BST encoding of the input time series, whereby new incoming sequence values extend the tree via a query on the existing structure. First, an indexed time series $\tau = (x_1, \ldots, x_n)$ is sorted into decreasing order of x_i then a BST is built in the standard way: beginning with the largest x_i as a root node, each subsequent sorted value is added by comparing its index with the indices of the existing nodes in the BST, recursively descending the tree from the root by moving left when the new index is less than an existing node and right when it is larger, finally adding a child node to an existing value when a vacant spot is found. Once the encoding is computed, which is an $O(n \log n)$ process on time series length on average for balanced data and $O(n^2)$ in the worst case, the vertex-edge representation of the HVG of τ is constructed by processing a set of edge connectivity rules [32, §IV.B]; these 'decoding' steps translate the branching structure in the BST to the edge structure of the HVG and take $O(n \log n)$ operations to complete.

The BST approach has the benefit of being able to run online with new data points being added as they arrive. Moreover, it has the elegant property of being able to 'merge' two BST encodings of subsequences τ_1, τ_2 to give the correct BST of their concatenation $\tau = \tau_1 + \tau_2$. It does this by using the root node of the BST for τ_2, corresponding to the maximum value in the subsequence τ_2, to query the BST of τ_1. However, when a vacant spot is found for the query node to be added to the tree of τ_1, the entire BST of τ_2 is added as a subtree at this point. The connectivity rules for decoding a BST to an HVG then ensure that the resulting HVG is exactly that of the full sequence τ. The merge operation is average-case $O(\log n)$ for balanced data and worst-case $O(n)$ in general, on the length of τ_1.

Unlike the DC algorithm, the BST approach enables genuinely online and fixed batch size parallel processing of a time series to produce its HVG, via the BST merge operation. However, its runtime complexity suffers from the same sensitivity to the geometric properties of the input sequence as the DC approach: on balanced data the algorithm encodes time series in balanced binary search trees, with depth $O(\log n)$ and therefore construction time $O(n \log n)$, but on unbalanced data the encoded BSTs have depth $O(n)$ leading to build times of $O(n^2)$. Thus while the BST approach offers scalability via subsequence batch processing, it does not do this efficiently in general.

Contribution

The horizontal visibility criterion (1) allows intermediate values x_k to 'block' the existence of edges between $i < j$ whenever $x_k \geq x_i$ or $x_k \geq x_j$, so the resulting graphs always consist of a hierarchy of nested edges, as shown in Figure 1. It is the structure of the hierarchy of these nested edges that we exploit in what follows.

In particular we outline worst-case $O(n)$ scalable algorithms for both HVG construction and merging, based on the concept of a 'dual tree HVG' (DTHVG) that encodes the edge nesting. The DTHVG is a graph structure that preserves enough geometric information from input sequences to make merging possible. Our concept of a 'bipartite HVG' is also introduced as a key component of the merging process. We show that our DTHVG merge Algorithm 2 reduces to the FWHVG one [15] when one of the two input subsequences is of length one. However, our merge algorithm extends to arbitrary-length subsequence merging for genuine scalability. In the final sections we observe that the empirical performance of the DTHVG approach outperforms both the DC and BST encoding-decoding approaches, by orders of magnitude, on both balanced and unbalanced synthetic and real-world data.

II. Proposed Method: Visibility via Merge Trees

In this section we leverage a result on the topology of time series which implies that simple constructions on ordered rooted metric trees can be used indirectly to build and manipulate horizontal visibility graphs. We first use this to give a concrete translation from the operation of iterated leaf grafting on such trees to iterated edge nesting in HVGs. Extending beyond single edge grafting to the fusion of entire branches in two trees then leads to the 'bipartite HVG' idea defined in Section II-C below.

A. Duality with Trees

Our method employs a variant of a common geometric representation of time series, used widely in the physics of self-similar systems and more recently in Topological Data Analysis, called the *(sub-level set) merge tree* of the data. For discrete data the merge tree is most often expressed with respect to a linear interpolation of the sequence but in this paper we use a definition with a more direct link to horizontal visibility graphs.

Informally, consider a discrete time series $\tau = (x_1, \ldots, x_n)$. These data can be represented visually in the plane as shown in Figure 2, using a bar graph $\mathbf{B}_\tau \subset \mathbb{R}^2$ consisting of narrow vertical bars with heights x_i spaced out evenly along the horizontal axis in $\mathbb{R}^2$. We can imagine the bars being unbounded and extending downwards in $\mathbb{R}^2$ to $-\infty$. The complement of these bars is then a space $\mathbb{T}_\tau = \mathbb{R}^2 - \mathbf{B}_\tau$ with gaps where the bars appeared. Contracting $\mathbb{T}_\tau$ by the equivalence relation that squashes horizontal lines to points,

$$(x, y) \cong (x', y') \in \mathbb{T}_\tau \text{ iff } \begin{cases} y = y' \text{ and the straight} \\ \text{line joining } (x, y) \text{ to } (x', y') \\ \text{in } \mathbb{R}^2 \text{ lies entirely in } \mathbb{T}_\tau, \end{cases} \tag{2}$$

gives an ordered metric tree we call the *time series merge tree* $T_\tau = \mathbb{T}_\tau / \cong$ of τ. In this setting the tree is unbounded due to infinite half-open edges where the root and leaf vertices would usually attach and it has a natural height function $h : T_\tau \to \mathbb{R}$ inherited from $\mathbb{T}_\tau$ since the equivalence $\cong$ does not identify any pair of points in $\mathbb{T}_\tau$ whose y values differ.

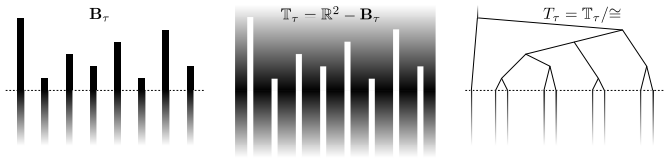

Fig. 2. A bar graph representation $\mathbf{B}_\tau \subset \mathbb{R}^2$ of a discrete time series τ (black bars – left). Its complement $\mathbb{T}_\tau$ in the plane (shaded region – middle). The merge tree T_τ that results from squashing horizontal lines in $\mathbb{T}_\tau$ according to the equivalence relation $\cong$ defined in Equation 2 in the text (right).

The connection between time series merge trees and horizontal visibility graphs stems from a recent result [35, Theorem 11] as follows. First recall that the *dual graph* G^* of an embedded plane graph $G \subset \mathbb{R}^2$ consists of vertices corresponding to each connected region $\rho \in \mathbb{R}^2 - G$ in the complement of G and edges between vertices ρ_1, ρ_2 exactly when their closures intersect along some interval I, meaning there exists a continuous map $\overline{\rho_1} \cap \overline{\rho_2} \to I$. A plane graph dual is illustrated in Figure 3. The main result in [35] shows

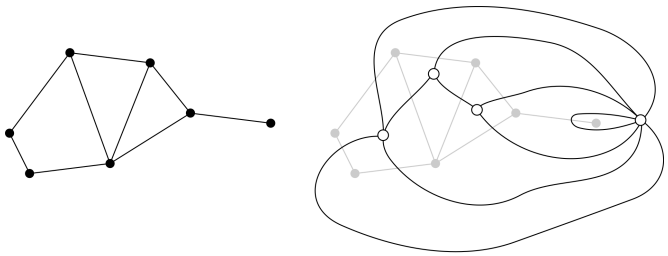

Fig. 3. A plane graph G (left) and its dual graph G^* (right).

that the HVG of τ is in fact a subgraph of the dual T_τ^* of the quotient space $T_\tau = \mathbb{T}_\tau / \cong$ described above.[1] Moreover the HVG omits only two vertices and their corresponding edges from T_τ^*: those corresponding to the two regions whose boundaries intersect along the root edge of the tree T_τ. So for practical purposes horizontal visibility graphs can be thought of as merge trees and the latter can be used to formally reason about HVGs, the main difference being that HVGs 'forget' the induced height function $h : T_\tau \to \mathbb{R}$ on the tree. We call the dual graph T_τ^* the *dual tree horizontal visibility graph* or DTHVG for short.

For clarity in the remainder of this paper we visualize time series merge trees as in Figure 4. That is, T_τ is drawn on an upper half-plane $\{(x, y) : y > y_{\min}\} \subset \mathbb{R}^2$ and 'leaves' are anchored along the boundary edge $y = y_{\min}$. Formally the leaf edges are half-open and this boundary actually designates

[1] More formally the finding in [35] states that the HVG is contained in the dual of the (Alexandroff) one-point compactification of T_τ with respect to a natural embedding of this graph in the 2-sphere.

the limit $y \to -\infty$ so the edges are never incident to it. The dual T_τ^* of T_τ for $\tau = (x_1, \dots, x_n)$ in this depiction is constructed by adding vertices for each of the n connected regions $\{\bar{1}, \bar{2} \dots, \bar{n}\}$ 'under' T_τ plus a vertex for each of the two connected regions $\overline{-\infty}$ and $\overline{\infty}$ 'above' T_τ whose boundaries meet along the half-edge corresponding to the root of T_τ. Edges $(\bar{i}, \bar{j})$ are added to T_τ^* to connect any regions in the half-plane that share a boundary edge in the drawing of T_τ.

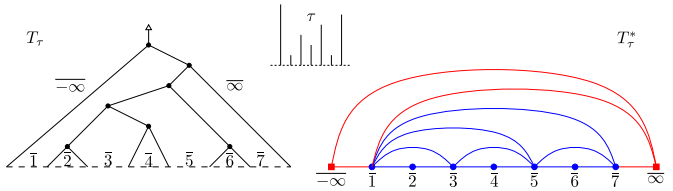

Fig. 4. A time series merge tree T_τ (left) and its dual T_τ^* (right) for an example time series $\tau = (6, 1, 3, 2, 4, 1, 5)$ (middle). Vertices in T_τ^* correspond to the labelled regions in T_τ. In particular the square vertices in T_τ^* correspond to the two regions 'above' the tree T_τ. Note that the subgraph of T_τ^* defined by the circular vertices, corresponding to regions $\bar{i}$ for $i = 1, \dots, 7$ 'below' the tree, is the horizontal visibility graph of τ.

B. From Leaf Grafting to Efficient Online Visibility

By moving from horizontal visibility to a dual tree representation an efficient online construction method for HVGs becomes evident. Consider a time series $\tau = (x_1, \dots, x_n)$. We can process it to create an increasing sequence of trees

$$\varepsilon = T_0 < T_1 < \cdots < T_n = T_\tau$$

whose final tree corresponds to the DTHVG.[2] In following this process the intermediate trees T_i will also correspond to the visibility graphs of the prefix sequences $\tau_i = (x_1, \dots, x_i)$ of τ, meaning that the resulting algorithm will be inherently online.

The following definition and lemma capture the operation required to move from each T_i to T_{i+1}.

Definition 1. Given a time series merge tree T in the plane suppose its leaf edges are labelled $\{l_0, \dots, l_n\}$ in increasing order according to their order in the x direction. Denote by p_i the unique path in T that includes leaf edge l_i and the root edge at the top of the tree. The operation of *grafting* a new leaf on to T involves choosing a point anywhere on the 'leading' or 'rightmost' branch p_n and connecting a new edge l_{i+1} to it.

The two ways in which grafting can extend a merge tree are illustrated in Figure 5. In either case the next Lemma shows that when processing a time series, if the height $h(p) \in \mathbb{R}$ of the graft point p in the tree is chosen correctly on each iteration then the resulting sequence of trees will be the merge trees of the prefix sequences of τ.

Lemma 2. *Given a time series merge tree T_i corresponding to a sequence $\tau_i = (x_1, \dots, x_i)$ and a new value x_{i+1}, the*

[2]The initial tree ε is a single edge from $y = -\infty$ to $y = \infty$.

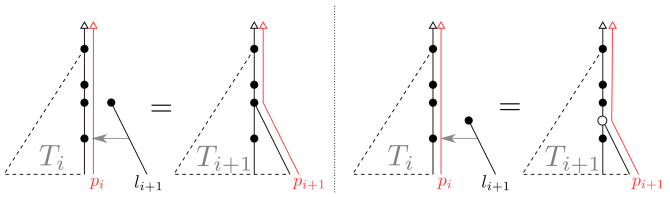

Fig. 5. Leaf grafting on to an existing merge tree T_i corresponds to extending an HVG with a new vertex. The graft can connect a new leaf edge l_{i+1} to an existing vertex on the leading branch p_i of the tree (left) or it can create a new vertex and bisect an existing edge on the leading branch of the tree (right).

operation of grafting a new leaf edge l_{i+1} to T_i at the unique point p in the leading branch p_i in T_i satisfying $h(p) = x_{i+1}$ results in a tree T_{i+1} that is precisely the merge tree of the extended sequence $\tau_{i+1} = (x_1, \dots, x_i, x_{i+1})$.

Proof. When leaf l_{i+1} is merged on to T_i it creates a new enclosed region $\overline{i+1}$ under the tree and thus new shared edges between regions. The set Λ of those edges in T_{i+1} that are on the intersection of p_i and the boundary of $\overline{i+1}$ consists of a single edge e_α from the boundary of each region $\overline{\alpha}$ satisfying the following conditions.

- In the case that l_{i+1} was grafted to an existing vertex in T_i:
  - e_α was previously on the the boundaries of $\overline{\alpha}$ and $\overline{\infty}$ in T_i,
  - $x_\alpha \leq x_{i+1}$, since l_{i+1} was merged at height x_{i+1}.
- In the case that l_{i+1} was grafted to an edge $e_{\alpha^*} \in \Lambda$ in T_i:
  - $\alpha \geq \alpha^*$,
  - e_α was previously on the the boundaries of $\overline{\alpha}$ and $\overline{\infty}$ in T_i,
  - $\alpha > \alpha^* \Rightarrow x_\alpha \leq x_{i+1}$, since l_{i+1} was merged at height x_{i+1}.

In either case for all $e_\alpha \in \Lambda$ there is no k with $\alpha < k \leq i$ such that $x_k \geq x_\alpha, x_{i+1}$. So the regions $\overline{\alpha}$ indexing the edges in Λ are precisely those whose corresponding bars b_α are horizontally visible to the added b_{i+1}. So T_{i+1} is the merge tree of τ_{i+1}. □

Since the merge tree contains the HVG in its dual, the upshot of Lemma 2 is that iterated grafting of leaves on to merge trees is an online process for building HVGs. Moreover the only data required at each step are the edge lengths along the current 'leading branch' (rightmost branch) of the tree. No other information about the existing merge tree is required to correctly graft a new leaf. Symmetrically, we can consider grafting a leaf edge on to the current 'trailing branch' (leftmost branch) of the tree, such as when a previous data point is provided. In this case the only data required are the edge lengths along the trailing branch. Accordingly we define a DTHVG data structure as follows.

Definition 3. Given a time series $\tau = (x_1, \dots, x_n)$ its dual tree horizontal visibility graph $\mathrm{DTHVG}(\tau) = (V, E, \Gamma, \Lambda)$

consists of the vertices and edges (V, E) of the HVG of the sequence $(\infty, x_1, \ldots, x_n, \infty)$ and two stacks of (region index, sequence value) pairs, Γ and Λ, corresponding to the regions with an edge respectively on the trailing or leading branches of the tree.

This data structure is incorporated naturally into the efficient HVG construction presented in Algorithm 1. The stack Λ is kept up-to-date to reflect the regions with an edge on the current 'leading branch' of the tree. When a new vertex is added to the HVG this stack is processed to add new edges to all of the regions in the stack that share a boundary edge with the new region under the merge tree and to update the stack appropriately. Note that the stack Γ in the DTHVG data structure corresponding to the trailing branch is not used in this computation, other than to populate it with a monotonically increasing subsequence of elements of τ.

In practice the vertex and edge sets of $\mathrm{DTHVG}(\tau) = (V, E, \Gamma, \Lambda)$ can be maintained explicitly by the algorithm during processing since the edges E are in bijective correspondence with the edge set of the corresponding tree. Therefore no encoding or decoding steps are required during operation. This leads to an extremely simple implementation. Moreover the process is optimal with respect to the worst case number of operations needed to process any time series as the next result shows.

Proposition 4. *The dual tree-based algorithm (DT) in Algorithm 1 has worst case time complexity $O(n)$ with respect to the length of input sequence $n = |\tau|$.*

Proof. The code inside the **while** block of Algorithm 1 adds exactly one edge to E each time it is entered. As an outerplanar graph, the number of edges in the HVG of a sequence of length n is bounded above by $2n - 3$ [36]. Thus the loop condition is satisfied at most $2n - 3$ times when processing the entire sequence τ. Since the **while** loop's code block involves only pops, pushes, equality checking and assignment, which are all $O(1)$, the total cost of the algorithm is $O(n)$. □

Indeed this proof shows that Algorithm 1 is $O(n)$ with a very small constant factor of 2, meaning it should be fast in practice as well. Section III below explores the extent to which this is true.

C. From Branch Fusion to Efficient Graph Merging

A key benefit of the binary search tree algorithm over the divide and conquer approach is that it allows 'merging' of pairs of pre-computed HVG encodings to construct larger HVG encodings. This makes BST scalable in the sense that batches of time series values can be converted to small HVG binary search tree representations, say on different cores of a multi-core processor, and these can then be combined together to give the full HVG that would result from taking all batches together and computing the HVG in its entirety 'offline'.

The dual tree approach to HVGs also provides a natural merge operation. It is based on the idea of fusing two merge trees together along their closest branches.

Algorithm 1: Linear-Time Algorithm for DTHVG Construction.

Data: time series $\tau = (x_1, \ldots, x_n) \in \mathbb{R}^n$
Result: dual tree horizontal visibility graph of τ

$V \leftarrow \{1, \ldots, n\}$
$E \leftarrow \emptyset$
$\Gamma \leftarrow Stack()$ `// Regions with an edge in trailing branch of` T_τ
$\gamma \leftarrow -\infty$ `// Store max value seen so far`
$\Lambda \leftarrow Stack()$ `// Regions with an edge in leading branch of` T_τ
for $v \in V$ **do**
  if $x_v > \gamma$ **then**
    $\gamma \leftarrow x_v$
    $\Gamma.push((v, x_v))$
  while Λ *is not empty* **do**
    $(u, x_u) \leftarrow \Lambda.pop()$
    $E \leftarrow E \cup \{(u, v)\}$
    if $x_u > x_v$ **then** $\Lambda.push((u, x_u))$
    if $x_u \geq x_v$ **then break**
  $\Lambda.push((v, x_v))$
return (V, E, Γ, Λ)

Definition 5. Given time series merge trees T, T' in the plane, suppose their leaf edges are labelled $\{l_0, \ldots, l_m\}$ and $\{l'_1, \ldots, l'_n\}$ in increasing order according to their order in the x direction. Denote by p_i the unique path in T that includes leaf edge l_i and the root edge at the top of the tree, and similarly for p'_i in T'. The operation of *fusing* tree T to T', denoted $T \boxplus T'$, involves applying the equivalence relation $\cong$ of Equation 2 to the points in the leading branch p_m in T and the trailing branch p'_0 in T'. That is, points on the two paths are identified if and only if they are at the same height according to the induced height functions $h : T \to \mathbb{R}$ and $h' : T' \to \mathbb{R}$.

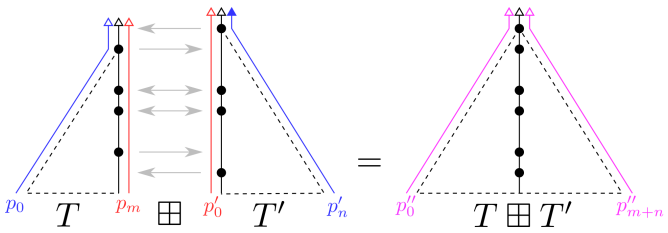

Fig. 6. Fusing two merge trees along their leading and trailing branches p_m and p'_0 corresponds to merging two HVGs. The fused path contains vertices at every height in the union of the heights of vertices in p_m and p'_0. Note that the resulting leading and trailing branches p''_0 and p''_{m+n} in the merged tree $T \boxplus T'$ are not necessarily equal to the input trailing and leading branches, p_0 and p'_n respectively, since one or more vertices in one of the branches being fused may exceed the height of the root of the other tree.

Extending HVGs in this way is illustrated in Figure 6. Similar to leaf grafting, the only data required to construct the combined tree $T \boxplus T'$ are the indices of the regions whose boundaries include an edge on the leading or trailing branches.

In Algorithm 1 the leading edge data were managed in a stack but the trailing edge data were not used. To permit merging we now simply use the second stack in the DTHVG data structure of Definition 3 to represent the (regions incident to) the trailing edge of the merge tree. The work is then in determining how the regions defined by the leading and trailing stacks of T and T' respectively should be connected in the fused tree $T \boxplus T'$. To help with this we make the following definition.

Definition 6. Given two time series $\tau = (x_1, \ldots, x_m)$ and $\tau' = (x_{m+1}, \ldots, x_{m+n})$, the *bipartite horizontal visibility graph* $\mathrm{HVG}_{\mathrm{bp}}(\tau, \tau') = (V_{\mathrm{bp}}, E_{\mathrm{bp}})$ of the pair is the subgraph of $\mathrm{HVG}(\tau'') = (V, E)$ of the concatenated sequence $\tau'' := \tau + \tau'$ defined as follows:

$$V_{\mathrm{bp}} = V,$$
$$E_{\mathrm{bp}} = \{(u, v) \in E : u \leq m \text{ and } v > m\}.$$

Thus the bipartite HVG of any prefix-suffix partition of a time series τ contains precisely those edges from the regular HVG of τ that connect the prefix vertices to the suffix vertices.

Merging can make use of the bipartite HVG as follows. Given a time series $\tau = (x_1, \ldots, x_m)$ with merge tree T and time series $\tau' = (x_{m+1}, \ldots, x_{m+n})$ with merge tree T', suppose the leading branch of T is represented by the stack of strictly increasing indices $\Lambda = (\lambda_1, \ldots, \lambda_a), a \leq m$ with each $\lambda_i \in \{1 \ldots, m\}$. Similarly let the trailing branch of T' be represented by the stack of strictly increasing indices $\Gamma = (\gamma_1, \ldots, \gamma_b), b \leq n$ with each $\gamma_i \in \{m+1, \ldots, m+n\}$. Note that the corresponding subsequences of values $\tau_\Lambda = (x_{\lambda_1}, \ldots, x_{\lambda_a})$ from τ and $\tau'_\Gamma = (x_{\gamma_1}, \ldots, x_{\gamma_b})$ from τ' are then strictly monotonically decreasing and increasing respectively. These two subsequences, τ_Λ and τ'_Γ, give the heights of the vertices that will appear on the fused branch of the tree $T \boxplus T'$, while the edges between them give the (duals of the) edges that need to be added to the edges in $\mathrm{HVG}(\tau)$ and $\mathrm{HVG}(\tau')$ to give the additional edges in the final merged $\mathrm{HVG}(\tau + \tau')$.

Computing the resulting vertex heights, and thus edge lengths, in the fused branch is easy enough: simply compute the unique interleaving of the two sequences Λ and Γ, which are already monotonic sequences, that gives a total order on the set of their values $\tau_\Lambda \cup \tau'_\Gamma$.[3] This is an $O(n)$ operation on the interleaved sequence length.

However to explicitly represent the dual graph that contains the merged HVG we also need to keep track of which regions in the two input trees now share boundary edges. This is where the bipartite HVG can be used in the tree-fusing operation:

1) Begin with merge trees T, T' for two time series $\tau = (x_1, \ldots, x_m), \tau' = (x_{1+m}, \ldots, x_{n+m})$.
2) Extract lists Λ, Λ' of the regions under T, T' whose boundary has an edge on the leading (rightmost) branch of the tree.
3) Extract lists Γ, Γ' of the regions under T, T' whose boundary has an edge on the trailing (leftmost) branch of the tree.
4) Take the duals of T, T' and exclude the first and last vertices to give the HVGs $(V_\tau, E_\tau), (V_{\tau'}, E_{\tau'})$ of τ, τ'.
5) Compute the bipartite graph $\mathrm{HVG}_{\mathrm{bp}}(\tau_\Lambda, \tau'_{\Gamma'}) = (V_{\mathrm{bp}}, E_{\mathrm{bp}})$ of the leading and trailing branch subsequences to be connected.
6) The vertex and edge sets of the merged HVG $(V_{\tau+\tau'}, E_{\tau+\tau'})$ are the disjoint unions:
$$V_{\tau+\tau'} = V_\tau \cup V_{\tau'},$$
$$E_{\tau+\tau'} = E_\tau \cup E_{\tau'} \cup E_{\mathrm{bp}}.$$
7) For subsequent merges also compute the regions $\Gamma_{T \boxplus T'}$ under $T \boxplus T'$ whose boundary has an edge on the trailing (leftmost) branch of the tree, and the regions $\Lambda_{T \boxplus T'}$ under $T \boxplus T'$ whose boundary has an edge on the leading (rightmost) branch of the tree:
$$\Gamma_{T \boxplus T'} = \Gamma \cup \{i \in \Gamma' : x_i > \max_{j \in \Gamma}(x_j)\},$$
$$\Lambda_{T \boxplus T'} = \{i \in \Lambda : x_i > \max_{j \in \Lambda'}(x_j)\} \cup \Lambda'.$$

Pseudocode to implement HVG merging using this idea of fusing merge trees appears in Algorithm 2. Keeping track of the stacks that represent the indices of the regions with boundary edges in the trailing and leading branches of the trees is the key algorithmic requirement. As with leaf grafting the algorithm does not explicitly represent the tree since its dual is fully determined by the HVG and its leading and trailing branch data. Lines 1-6 construct the subsequences of τ, τ' corresponding to their leading and trailing branches respectively. Line 7 constructs the bipartite horizontal visibility graph joining these subsequences. Lines 8-16 compute the leading and trailing branch indices of the fused tree.

As with online DTHVG construction via leaf grafting, the process is $O(n)$ where in this case n is the number of vertices in the output HVG. All loops are iterated at most $O(n)$ times and contain lines that are worst-case $O(1)$. Line 7 is the only exception, where a function to compute the bipartite HVG is called. As mentioned above this is $O(n)$ on its longest input sequence length since the key computation is an interleaving of monotonic subsequences of the inputs.

III. Numerical Experiments

We now present numerical results to show how the dual tree (DT) algorithm performs in practice, compared to the binary search tree (BST) and divide and conquer (DC) approaches.[4] Test processors were 2.1GHz Intel Core (Haswell, IBRS) CPUs and the test machine had 32GB of RAM available across 64 logical cores. To ensure a fair comparison the maximum recursion depth of the Python interpreter was set to the length of the series, to guarantee that BST and DC would complete on unbalanced data.

[3] For uniqueness in the presence of equal values $x_{\lambda_i} = x_{\gamma_j}$ a convention for their interleaved order is needed: for example indices from Λ precede those from Γ in the interleaving.

[4] All Python code, time series data, and raw results are available online [37] to enable others to reproduce and/or extend the results shown here.

Algorithm 2: Linear-Time Algorithm for DTHVG Merging.

Data: DTHVGs (V, E, Γ, Λ) and $(V', E', \Gamma', \Lambda')$ associated to Time series $\tau = (x_1, \ldots, x_m)$ and $\tau' = (x_{1+m}, \ldots, x_{n+m})$.

Result: DTHVG of concatenated time series $\tau + \tau' = (x_1, \ldots, x_{n+m})$

$\tau_\Lambda \leftarrow \epsilon$
for $i \in \Lambda$ **do**
  $\tau_\Lambda \leftarrow \tau_\Lambda + x_i$
$\tau'_{\Gamma'} \leftarrow \epsilon$
for $i \in \Gamma'$ **do**
  $\tau'_{\Gamma'} \leftarrow \tau'_{\Gamma'} + x_i$
$(V_{\mathrm{bp}}, E_{\mathrm{bp}}) \leftarrow \mathrm{HVG}_{\mathrm{bp}}(\tau_\Lambda, \tau'_{\Gamma'})$
$\Gamma'' \leftarrow \Gamma$
for $i \in \Gamma'$ **do**
  if $x_i > x_\gamma$, *where* γ *is the last entry in* Γ'' **then**
    $\Gamma'' \leftarrow \Gamma'' + i$
$\Lambda'' \leftarrow \epsilon$
for $i \in \Lambda$ **do**
  if $x_i > x_\lambda$, *where* λ *is the first entry in* Λ' **then**
    $\Lambda'' \leftarrow \Lambda'' + i$
$\Lambda'' \leftarrow \Lambda'' + \Lambda'$
return $(V \cup V', E \cup E' \cup E_{bp}, \Gamma'', \Lambda'')$

Note that all times reported for the BST method are for the *encoding step only*. The additional overhead of decoding the binary search tree to give a vertex-edge or adjacency representation of the HVG, such as those returned by the DC and DT methods, is not included.

A. Baseline Performance

The baseline performance of the three algorithms on increasingly long sequences is presented in Figure 7. Details of the corresponding sources can be found in Table I. They show that DT quickly outperforms BST and DC by orders of magnitude and that this performance is consistent across the systems considered. For very short sequences of around five hundred points the performances are closer but the running times diverge rapidly even for relatively short sequence lengths. These are not long sequences but the divergence in runtime performance is already apparent.

As illustrated in Figure 7 and Table II, the performance advantage of DT over the others is lowest for sequences of Random Noise. This is expected because uniformly sampled data lead more probably to balanced search trees and faster data sorting for BST and to more evenly sized splits of the data for DC. In turn this leads to reduced recursion depth during execution and consequently to faster run times. Line-by-line code profiling using Python's `line_profiler` and `cProfile` modules confirms this. Nevertheless the DT method is still faster than BST by a factor of 2.76 for this 'worst' case. On the other hand Random Walk sequences correspond to the greatest advantage for DT with a speed up of 50.94 times,

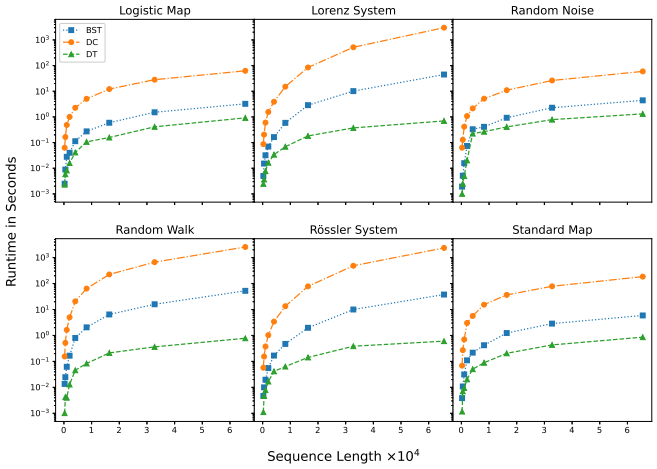

Fig. 7. Running times for BST, DC, and DT algorithms on trajectories generated by a range of dynamical systems (see Table I for details). The Lorenz and Rössler systems here were sampled from ODE solutions of length $t_{\max} = 250$ and $t_{\max} = 1024$ seconds respectively. All times are averages over ten trajectories of each length.

since the resulting binary search trees are less balanced. In the next section we quantify the impact of correlations along an incremental trajectory such as this in more detail. For now, we can conclude that the baseline performance of DTHVG construction, without the benefit of a distributed workload, is significantly better than the baseline performance of the BST construction of HVGs.

B. Long Range Dependence, Sample Frequency, and Noise

As the DTHVG construction algorithm does not recurse its performance is far less dependent on the geometric structure of an input time series. Here we consider the sensitivity of HVG algorithms to statistical correlations, different sample rates on continuous sources and also to the presence of noise. All time series have fixed length 2^{15}. Only the BST and DT runtimes are compared since DC is orders of magnitude slower.

1) Sensitivity to Long Range Dependence: We use fractional Brownian motion (fBm) trajectories to quantify the impact of short versus long range dependence between points in a stochastic incremental trajectory. The fBm generalizes random walks by inclusion of a parameter that controls the extent to which previous time series values influence the value of the next sample to be drawn. In particular an fBm is a continuous trajectory $B_h(t)$ on some interval $[0, t_{\max}]$, where $B_h(0) = 0$, $E[B_h(t)] = 0$, and whose auto-covariance is parameterized by a *Hurst exponent* $h \in (0, 1)$ according to $E[B_h(t)B_h(s)] = \frac{1}{2}\left(|t|^{2h} + |s|^{2h} - |t-s|^{2h}\right)$. For low values of the Hurst exponent $h < 0.5$ there is a negative correlation between increments on the trajectory, leading to higher frequency variations over shorter time spans as in the left panel of Figure 8. For higher values $h > 0.5$ the increments exhibit long range dependence leading to sustained trends over longer durations, as in the right panel of Figure 8. For $h = 0.5$ the trajectory increments are independent, giving

the continuous time version of the discrete Random Walk used in the previous section.

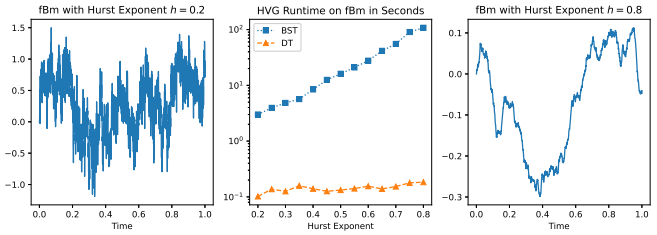

Fig. 8. Running times for BST and DT algorithms on fractional Brownian motion (fBm) trajectories with increasing Hurst exponents (centre). All times are averaged over ten trajectories of length 2^{15} with the given exponent. Example trajectories from a fBm with $h = 0.2$ (left) and $h = 0.8$ (right) are also shown. The trajectories were generated using the method of Davies and Harte [38] as implemented in the Python `fbm` module.

The centre panel in Figure 8 shows that as the strength of positive correlation between points in a trajectory increases, so does the processing time of the BST algorithm. This indicates that if used to process a time series from a system whose auto-covariance decays as a power law the BST processing time will be significantly worse than if the auto-covariance decay were exponential. By contrast the DT algorithm maintains a stable performance profile even for sequences with very strong long range dependence.

2) Sensitivity to Sample Frequency: In practice time series are sampled from physical sources using a sample frequency that depends on the application and on the sensitivity of measuring equipment. To illustrate the sensitivity to sample frequency of the preprocessing step of computing an HVG, we consider the continuous deterministic Rössler trajectory defined in Table I. Equal length sequences of this system trajectory were sampled at a range of frequencies from 8Hz to 128Hz, and the BST and DT algorithm run times were recorded. The results are shown in Figure 9.

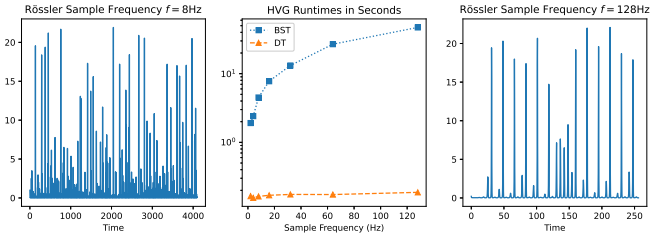

Fig. 9. Running times for BST and DT algorithms on sequences of fixed length 2^{15} sampled from Rössler trajectories at increasing frequencies. All times are averaged over ten trajectories. An example trajectory with $f = 8$Hz (left) and $f = 128$Hz (right) are also shown.

The effect of increasing frequency on the run time of the Dual Tree algorithm is negligible, while the run time of the Binary Search Tree algorithm increases almost one hundred fold for the range of frequencies considered. A similar pattern is witnessed for other continuous deterministic systems. Again this is due to the increased recursion depth of BST when there are increasingly longer monotonic subsequences in a time series, a situation that occurs in practice when the sample frequency is increased.

3) Sensitivity to Noise: There is also an effect of varying the signal to noise ratio of a signal on its HVG construction time. Two cycles of a pure sine wave with varying levels of additive noise were processed using the BST and DT algorithms and the results are shown in Figure 10.

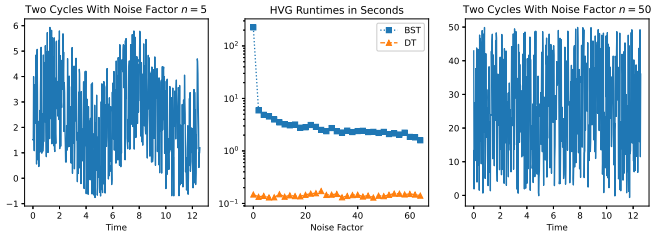

Fig. 10. Running times for BST and DT algorithms on two cycles of noisy periodic motion $y = \sin(t) + n\epsilon$ where $t \in [0, 4\pi]$, ϵ is uniformly sampled from $[0, 1)$ and n is a scale factor (centre). All times are averaged over ten trajectories of length 2^{15} with the given noise factor. An example trajectory with $n = 5$ (left) and $n = 50$ (right) are also shown.

As suggested by the benchmark results in Section III-A above, a signal that is mostly random leads to more balanced binary search trees and thus faster processing. Thus as the signal is lost to the increasing effects of noise, the processing time of the BST algorithm improves. In particular adding *any* amount of noise to the pure underlying signal improves the BST processing time by an order of magnitude and its performance slowly improves as the signal is dominated by the noise. Of course, the payoff for the improved speed of BST in this case is loss of access to the signal that supports the data. By contrast the signal to noise ratio has a negligible effect on the dual tree algorithm processing time.

C. Scaling Performance on Batched Data

Next we compare the HVG merging performance of the binary search tree and dual tree algorithms. In this section, figures record only the merge times and exclude the baseline subsequence HVG creation times for individual batches, which as we observed above is much lower for the DT construction method.

In Figure 11 we see a summary of merge times for the two algorithms on balanced EEG and unbalanced financial data sets, both of large scale >1M data points per sequence, as the input subsequence length (that is: batch size) is varied. As expected the more balanced data leads to a similar performance between the two algorithms with a slight advantage for the DTHVG approach. However, on the financial price data there is a rapid divergence in runtimes. The DT method maintains a relatively flat runtime performance while the BST approach increases its runtime rapidly as the batch size increases beyond around 50,000 sequence values. By the time the input subsequences approach length 0.5M, the difference in performance between the algorithms is substantial.

Figure 12 shows similar runtime profiles for merging HVGs of fractional Brownian motion trajectories, across different

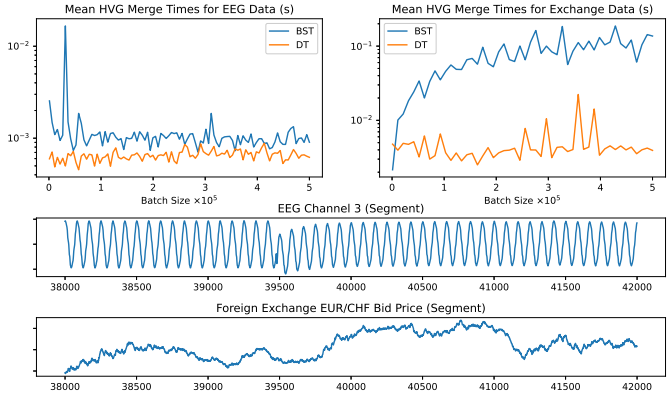

Fig. 11. Mean HVG merge times in seconds, plotted against size of HVG being merged (top row). Results are over five EEG channels each of length 3×10^6 (top left) and five foreign exchange currency tick-level bid prices with sequence lengths between 1.5×10^6 and 2.8×10^6 (top right). For each batch size the HVG of the entire sequence was computed by repeatedly merging the batch-sized HVGs. Note that construction times of the input HVGs are not included since the BST construction time dominates and obscures the underlying merge times. Segments of example trajectories from the two classes of sequence are also shown for reference (bottom).

values of the Hurst index h. By the time the subgraphs being merged are of length approaching 0.25M data points, the DT algorithm is already orders of magnitude faster for all of the sampled trajectories. For lower values of the Hurst index, the two algorithms maintain relatively stable runtimes, but for higher values of the Hurst index, corresponding to more positively correlated data, there is a clear divergence in performance. Additionally, in all cases the DTHVG merging algorithm shows much less variance in its runtime performance across batch sizes than does the BST algorithm.

IV. Conclusion

In this paper we have introduced a time series representation, the dual tree horizontal visibility graph (DTHVG) data structure, that directly captures enough of the geometry of a sequence to enable efficient construction and merging of HVGs. We have shown that this offers both theoretical advantages, namely $O(n)$ worst-case runtime for both graph construction and merging, and empirical advantages, namely significant runtime performance improvements over the best existing scalable algorithms for HVG construction and merging. Our dual tree HVG merging algorithm reduces to the fast FWHVG algorithm [15] in the case of online/streaming data, but it offers significant scalability via its ability to merge outputs of parallel computations on arbitrary length subsequences. It does this with a much better theoretical and practical performance than previous methods, remaining performance-agnostic with respect to balanced and unbalanced time series data sets. These benefits enable practical HVG construction on much longer time series than existing methods allow. Key features are summarized in comparison Table III, while Python code and data for the above experiments are also available in a Github repository associated with this paper [37] to enable reproduction and improvement of the results.

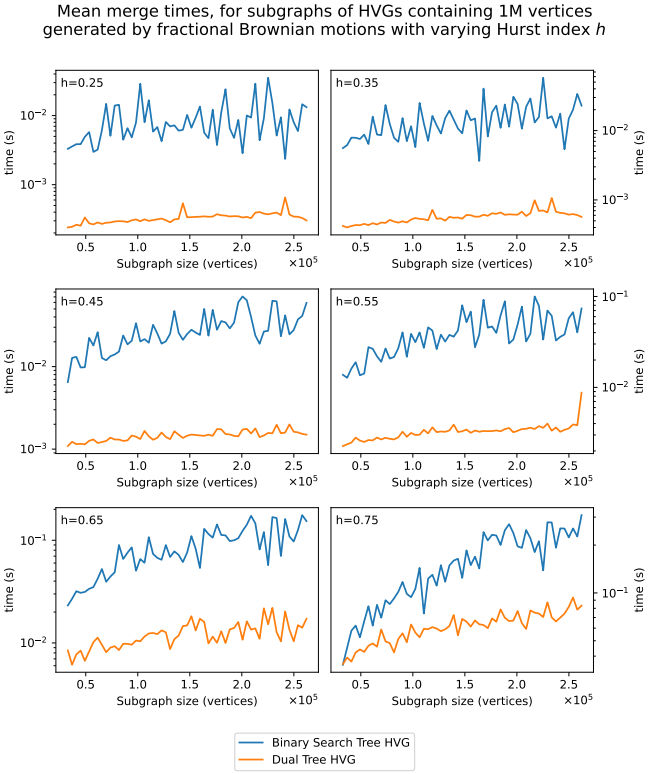

Fig. 12. Mean HVG merge times in seconds, plotted against size of HVG being merged.

V. TABLES

TABLE I

DYNAMICS OF TIME SERIES $\{x_i\}$ USED FOR BASELINE PERFORMANCE ANALYSIS. WHEN REQUIRED, INITIAL CONDITIONS WERE SELECTED UNIFORMLY FROM $[0, 1)$. THE PARAMETER VALUES CHOSEN IMPLY THAT DETERMINISTIC TRAJECTORIES ARE CHAOTIC.

Name	**Discrete**	**Stochastic**	**Dynamics**
Logistic Map	✓	×	$x_{i+1} = rx_i(1 - x_i)$ $r = 3.9995$
Lorenz System	×	×	$\dot{x} = \sigma(y - x)$ $\dot{y} = x(\rho - z) - y$ $\dot{z} = xy - \beta z$ $(\sigma, \rho, \beta) = (10, 28, 8/3)$
Random Noise	✓	✓	$P(x_i \leq x) = \begin{cases} 0: & x < 0 \\ x: & 0 \leq x \leq 1 \\ 1: & x > 1 \end{cases}$
Random Walk	✓	✓	$P(x_{i+1} = x_i + 1) = 0.5$ $P(x_{i+1} = x_i - 1) = 0.5$
Rössler System	×	×	$\dot{x} = b + x(z - c)$ $\dot{y} = z + ay$ $\dot{z} = -y - x$ $(a, b, c) = (0.2, 0.2, 5.7)$
Standard Map	✓	×	$x_{i+1} = x_i + K \sin(\theta_i)$ $\theta_{i+1} = \theta_i + x_{i+1}$ $K = 1.2$

TABLE II

MEAN RUNNING TIMES IN SECONDS OF HVG ALGORITHMS OVER NINETY TRAJECTORIES FROM THE GIVEN SOURCES (TEN TRAJECTORIES FOR EACH OF NINE LENGTHS $2^8, 2^9, \ldots, 2^{16}$). SUBSCRIPTS ON THE CONTINUOUS SOURCES INDICATE THE DURATION OF THE SAMPLED TRAJECTORIES IN SECONDS. THE SPEEDUP IS THE RATIO $t_{\text{BST}}/t_{\text{DT}}$ OF THE MEAN COMPUTE TIMES.

	BST	DC	DT	DT vs BST Speedup
Logistic Map	0.65	12.45	0.19	3.42
Lorenz_{250} System	6.52	402.58	0.15	43.47
Random Noise	0.94	11.73	0.34	2.76
Random Walk	8.66	394.13	0.17	50.94
Rössler_{1024} System	5.60	326.75	0.14	40.00
Standard Map	1.20	36.09	0.19	6.32

TABLE III

FEATURE COMPARISON WITH PUBLISHED HVG ALGORITHMS.

	HVG	**FWHVG**	**DC**	**BST**	**Dual Tree**
Source	[29] 2009	[15] 2014	[31] 2015	[32] 2020	-
Balanced Data	$O(n)$	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Arbitrary Data	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n)$
Online	×	Possible	×	✓	✓
Scalable	×	×	×	✓	✓
Non-Recursive	✓	✓	×	×	✓

REFERENCES

[1] Y. Zou, R. V. Donner, N. Marwan, J. F. Donges, and J. Kurths, "Complex network approaches to nonlinear time series analysis," *Physics Reports*, vol. 787, pp. 1–97, Jan. 2019.

[2] V. F. Silva, M. E. Silva, P. Ribeiro, and F. Silva, "Time series analysis via network science: Concepts and algorithms," *WIREs Data Mining and Knowledge Discovery*, vol. n/a, no. n/a, p. e1404, 2021.

[3] R. V. Donner, Y. Zou, J. F. Donges, N. Marwan, and J. Kurths, "Recurrence networks—a novel paradigm for nonlinear time series analysis," *New Journal of Physics*, vol. 12, no. 3, p. 033025, 2010.

[4] A. M. Nuñez, L. Lacasa, J. P. Gomez, and B. Luque, "Visibility algorithms: A short review," *New frontiers in graph theory*, pp. 119–152, 2012.

[5] M. Stephen, C. Gu, and H. Yang, "Visibility Graph Based Time Series Analysis," *PLOS ONE*, vol. 10, no. 11, p. e0143015, Nov. 2015.

[6] Z.-K. Gao, M. Small, and J. Kurths, "Complex network analysis of time series," *EPL (Europhysics Letters)*, vol. 116, no. 5, p. 50001, Dec. 2016.

[7] M. McCullough, M. Small, T. Stemler, and H. H.-C. Iu, "Time lagged ordinal partition networks for capturing dynamics of continuous dynamical systems," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 25, no. 5, p. 053101, May 2015.

[8] J. Zhang, J. Zhou, M. Tang, H. Guo, M. Small, and Y. Zou, "Constructing ordinal partition transition networks from multivariate time series," *Scientific Reports*, vol. 7, no. 1, p. 7795, Aug. 2017.

[9] B. Luque, L. Lacasa, F. Ballesteros, and J. Luque, "Horizontal visibility graphs: Exact results for random time series," *Physical Review E*, vol. 80, no. 4, p. 046103, Oct. 2009.

[10] L. Lacasa and R. Flanagan, "Time reversibility from visibility graphs of nonstationary processes," *Physical Review E*, vol. 92, no. 2, p. 022817, Aug. 2015.

[11] Á. M. Núñez, B. Luque, L. Lacasa, J. P. Gómez, and A. Robledo, "Horizontal visibility graphs generated by type-I intermittency," *Physical Review E*, vol. 87, no. 5, p. 052801, May 2013.

[12] Á. M. Núñez, L. Lacasa, and J. P. Gómez, "Horizontal Visibility graphs generated by type-II intermittency," *Journal of Physics A: Mathematical and Theoretical*, vol. 47, no. 3, p. 035102, Dec. 2013.

[13] W.-J. Xie and W.-X. Zhou, "Horizontal visibility graphs transformed from fractional Brownian motions: Topological properties versus Hurst index," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 20, pp. 3592–3601, Oct. 2011.

[14] M. D. Vamvakaris, A. A. Pantelous, and K. M. Zuev, "Time series analysis of S&P 500 index: A horizontal visibility graph approach," *Physica A: Statistical Mechanics and its Applications*, vol. 497, pp. 41–51, May 2018.

[15] G. Zhu, Y. Li, and P. P. Wen, "Epileptic seizure detection in EEGs signals using a fast weighted horizontal visibility algorithm," *Computer Methods and Programs in Biomedicine*, vol. 115, no. 2, pp. 64–75, Jul. 2014.

[16] Z.-K. Gao, Q. Cai, Y.-X. Yang, W.-D. Dang, and S.-S. Zhang, "Multiscale limited penetrable horizontal visibility graph for analyzing nonlinear timeseries," *Scientific Reports*, vol. 6, no. 1, p. 35622, Dec. 2016.

[17] L. Wang, X. Long, J. B. A. M. Arends, and R. M. Aarts, "EEG analysis of seizure patterns using visibility graphs for detection of generalized seizures," *Journal of Neuroscience Methods*, vol. 290, pp. 85–94, Oct. 2017.

[18] T. Madl, "Network Analysis of Heart Beat Intervals Using Horizontal Visibility Graphs," in *2016 Computing in Cardiology Conference*, Sep. 2016.

[19] J. Iacovacci and L. Lacasa, "Sequential visibility-graph motifs," *Physical Review E*, vol. 93, no. 4, p. 042309, Apr. 2016.

[20] G. I. Choudhary, W. Aziz, I. R. Khan, S. Rahardja, and P. Franti, "Analysing the Dynamics of Interbeat Interval Time Series Using Grouped Horizontal Visibility Graph," *IEEE Access*, vol. 7, pp. 9926–9934, 2019.

[21] A. C. Braga, L. G. A. Alves, L. S. Costa, A. A. Ribeiro, M. M. A. de Jesus, A. A. Tateishi, and H. V. Ribeiro, "Characterization of river flow fluctuations via horizontal visibility graphs," *Physica A: Statistical Mechanics and its Applications*, vol. 444, pp. 1003–1011, Feb. 2016.

[22] Z. G. Yu, V. Anh, R. Eastes, and D.-L. Wang, "Multifractal analysis of solar flare indices and their horizontal visibility graphs," *Nonlinear Processes in Geophysics*, vol. 19, no. 6, pp. 657–665, Nov. 2012.

[23] V. Suyal, A. Prasad, and H. P. Singh, "Visibility-Graph Analysis of the Solar Wind Velocity," *Solar Physics*, vol. 289, no. 1, pp. 379–389, Jan. 2014.

[24] Y. Zou, R. V. Donner, N. Marwan, M. Small, and J. Kurths, "Long-term changes in the north–south asymmetry of solar activity: A nonlinear dynamics characterization using visibility graphs," *Nonlinear Processes in Geophysics*, vol. 21, no. 6, pp. 1113–1126, Nov. 2014.

[25] B. Acosta-Tripailao, D. Pastén, and P. S. Moya, "Applying the Horizontal Visibility Graph Method to Study Irreversibility of Electromagnetic Turbulence in Non-Thermal Plasmas," *Entropy*, vol. 23, no. 4, p. 470, Apr. 2021.

[26] A. Aragoneses, L. Carpi, N. Tarasov, D. V. Churkin, M. C. Torrent, C. Masoller, and S. K. Turitsyn, "Unveiling Temporal Correlations Characteristic of a Phase Transition in the Output Intensity of a Fiber Laser," *Physical Review Letters*, vol. 116, no. 3, p. 033902, Jan. 2016.

[27] Y. Gao and D. Yu, "Total variation on horizontal visibility graph and its application to rolling bearing fault diagnosis," *Mechanism and Machine Theory*, vol. 147, p. 103768, May 2020.

[28] J. Iacovacci and L. Lacasa, "Visibility Graphs for Image Processing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 974–987, Apr. 2020.

[29] L. Lacasa, "Horizontal and Directed Horizontal visibility graphs (Fortran 90/95)," http://www.maths.qmul.ac.uk/~lacasa/Software.html, 2009.

[30] L. Lacasa, A. Nuñez, É. Roldán, J. M. R. Parrondo, and B. Luque, "Time series irreversibility: A visibility graph approach," *The European Physical Journal B*, vol. 85, no. 6, p. 217, Jun. 2012.

[31] X. Lan, H. Mo, S. Chen, Q. Liu, and Y. Deng, "Fast transformation from time series to visibility graphs," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 25, no. 8, p. 083105, Aug. 2015.

[32] D. Fano Yela, F. Thalmann, V. Nicosia, D. Stowell, and M. Sandler, "Online visibility graphs: Encoding visibility in a binary search tree," *Physical Review Research*, vol. 2, no. 2, p. 023069, Apr. 2020.

[33] L. Lacasa, B. Luque, F. Ballesteros, J. Luque, and J. C. Nuño, "From time series to complex networks: The visibility graph," *Proceedings of the National Academy of Sciences*, vol. 105, no. 13, pp. 4972–4975, Apr. 2008.

[34] S. Ghosh and A. Dutta, "An efficient non-recursive algorithm for transforming time series to visibility graph," *Physica A: Statistical Mechanics and its Applications*, vol. 514, pp. 189–202, Jan. 2019.

[35] C. Stephen, "Horizon Visibility Graphs and Time Series Merge Trees are Dual," *arXiv:1906.08825 [nlin, physics:physics]*, Jun. 2019.

[36] G. Gutin, T. Mansour, and S. Severini, "A characterization of horizontal visibility graphs and combinatorics on words," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 12, pp. 2421–2428, Jun. 2011.

[37] C. Stephen, "Dual Tree Horizontal Visibility Graphs: Python Code for Linear-Time HVGs," https://github.com/colinstephen/dual_tree_hvg, 2020.

[38] R. B. Davies and D. S. Harte, "Tests for Hurst effect," *Biometrika*, vol. 74, no. 1, pp. 95–101, 1987.