Bandwidth vs BFS Width in Matrix Reordering, Graph Reconstruction, and Graph Drawing

David Eppstein
University of California, Irvine, CA, USA

Michael T. Goodrich
University of California, Irvine, CA, USA

Songyu (Alfred) Liu
University of California, Irvine, CA, USA

Abstract

We provide the first approximation quality guarantees for the Cuthull-McKee heuristic for reordering symmetric matrices to have low bandwidth, and we provide an algorithm for reconstructing bounded-bandwidth graphs from distance oracles with near-linear query complexity. To prove these results we introduce a new width parameter, BFS width, and we prove polylogarithmic upper and lower bounds on the BFS width of graphs of bounded bandwidth. Unlike other width parameters, such as bandwidth, pathwidth, and treewidth, BFS width can easily be computed in polynomial time. Bounded BFS width implies bounded bandwidth, pathwidth, and treewidth, which in turn imply fixed-parameter tractable algorithms for many problems that are NP-hard for general graphs. In addition to their applications to matrix ordering, we also provide applications of BFS width to graph reconstruction, to reconstruct graphs from distance queries, and graph drawing, to construct arc diagrams of small height.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Graph algorithms, graph theory, graph width, bandwidth, treewidth

Digital Object Identifier 10.4230/LIPIcs.ESA.2025.69

Funding *David Eppstein*: Supported by NSF grant 2212129, as well for Michael Goodrich and Songyu (Alfred) Liu.

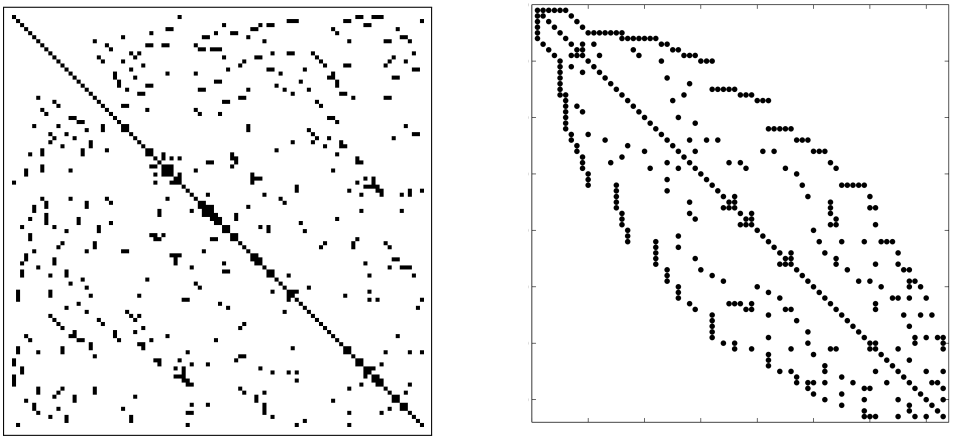
1 Introduction

When a graph has a linear layout of low bandwidth, it is natural to guess that breadth-first search produces a low-width layout. This is the underlying principle of the widely used Cuthill–McKee algorithm [13] for reordering symmetric matrices into band matrices with low bandwidth. In this paper, we quantify this principle, for the first time, by providing the first worst-case analysis of the Cuthill–McKee algorithm, proving polylogarithmic upper and lower bounds for its performance on bounded-bandwidth graphs. Using the same techniques, we also study the problem of reconstructing an unknown graph, given access to a distance oracle. We use a method based on breadth-first search to reconstruct graphs of bounded bandwidth in near-linear query complexity. Additionally, we apply our results in graph drawing, by constructing arc diagrams of low height for any graph of bounded bandwidth.

The key to our results is a natural but previously-unstudied graph width parameter, the **breadth-first-search width** or **BFS width** of a graph, which we define algorithmically in terms of the size of the layers in a breadth-first search tree. To develop our results, we prove that bandwidth and BFS width are polylogarithmically tied: the graphs of bandwidth $\leq b$, for any constant b, have BFS width that is both upper bounded (for all graphs of bandwidth $\leq b$) and lower bounded (for infinitely many graphs of bandwidth b) by functions of the form $\log^{f(b)} n$. To provide context for our contributions, we review the background of our three application areas below and then describe our new results in more detail for each.

Sparse Numerical Linear Algebra

A symmetric matrix, A, is called a **band matrix** if its nonzeros are confined to a **band** of consecutive diagonals centered on the main diagonal; the **bandwidth** is the number of these nonzero diagonals on either side of the main diagonal. That is, it is the maximum value of $|i - j|$ for the indexes of a nonzero matrix entry $A_{i,j}$. Any symmetric matrix corresponds to an undirected graph, with n numbered vertices, and with an edge $\{i, j\}$ for each off-diagonal nonzero entry $A_{i,j}$; the numbering of vertices describes a **linear layout** of this graph and the bandwidth of this layout is the largest difference between the positions of any two adjacent vertices in this linear layout. The bandwidth of the graph itself is the minimum bandwidth of any of its linear layouts; the goal of the Cuthill–McKee algorithm is to permute the vertices of the graph into a layout with bandwidth close to this minimum, and correspondingly to permute the rows and columns of the given matrix to produce an equivalent symmetric band matrix of low bandwidth. See Figure 1. Such a reordering, for instance, can be used in sparse numerical linear algebra to reduce fill-in (zero matrix elements that become nonzero) when applying Gaussian elimination to the matrix.



(a) (b)

Figure 1 Illustrating the Cuthill-McKee algorithm. (a) A sparse matrix; (b) a sparse matrix reordered according to the Cuthill-McKee algorithm. Nonzero entries are shown as black spots. Left: public-domain image by Oleg Alexandrov; right: image by Wikipedia user Kxx licensed under the CC BY-SA 3.0 license.

Graph bandwidth is NP-complete [20] and hard to approximate to within any constant factor, even for trees of a very special form [15]. For any fixed b, testing whether bandwidth $\leq b$ is polynomial, but with b in the exponent of the polynomial [25]. Therefore, practitioners have resorted to heuristics for bandwidth, rather than exact optimization algorithms. Notable among these are the Cuthill–McKee algorithm [13], and the closely related reverse Cuthill–McKee algorithm [21]; both are heavily cited and widely used for this task. More sophisticated methods have also been used and tested experimentally [34].

The best known polynomial time algorithm provides a $O(\log^3 n\sqrt{\log \log n})$-approximation factor for bandwidth on general graphs with high probability using a semi-definite relaxation [16]. A randomized polynomial time $O(\log^{2.5} n)$-approximation algorithm is known for the special cases of trees and chordal graphs using caterpillar decomposition [24]. In contrast, the Cuthill–McKee and reverse Cuthill–McKee algorithms are much simpler since they only rely on BFS and are deterministic. Despite their simplicity, we are unaware of any

past work providing worst-case performance guarantees for the Cuthill–McKee and reverse Cuthill–McKee algorithms. The only theoretical upper bound we are aware of regarding these algorithms concerns random matrices whose optimal bandwidth is sufficiently large, for which this algorithm will produce a constant factor approximation. The same work also provides a logarithmic lower bound on the approximation factor for matrices with optimal bandwidth 2, weaker than our result [35]. Instead, we are interested in deterministic guarantees for these algorithms for matrices of low optimal bandwidth. We prove the following results:

- For $n \times n$ matrices of bounded optimal bandwidth $\leq b$, we show that the Cuthill–McKee algorithm and the reverse Cuthill–McKee algorithm will find a reordering of bandwidth upper bounded by a function of the form $\log^{f(b)} n$. Equivalently, for n-vertex graphs of bandwidth $\leq b$, these algorithms will find a linear layout of bandwidth $\leq \log^{f(b)} n$.
- We show that there exist matrices for which this polylogarithmic dependence is necessary. For every polylogarithmic bound $\log^k n$ on the bandwidth, there is a constant bandwidth bound b such that, for infinitely many n, there exist matrices of optimal bandwidth $\leq b$ that cause both the Cuthill–McKee algorithm and the reverse Cuthill–McKee algorithm to produce reordered matrices of bandwidth $\Omega(\log^k n)$.

We remark that, for any graph,

$$O(1) \text{ BFS width} \implies O(1) \text{ bandwidth} \implies O(1) \text{ pathwidth} \implies O(1) \text{ treewidth [28]},$$

and graphs of bandwidth b have tree-depth $O(b \log n/b)$ [23]. Thus, many algorithmic results based on width parameters such as pathwidth, treewidth, and tree-depth are also available for BFS-width. However, both pathwidth and tree-depth have approximation algorithms with fixed polylogarithmic approximation ratios [6, 19], better than the polylogarithmic ratio with a variable exponent depending on the optimal bandwidth that we prove for the Cuthill–McKee algorithm.

Graph Reconstruction

We also provide new results for **graph reconstruction**, a well-motivated problem with numerous applications, such as learning road networks [2] and communication network mapping [1]. Suppose we are given the vertex set but not the edge set of a graph G. The **graph reconstruction** problem is to determine all the edges of G, by asking queries about G from an oracle, with the goal of minimizing the **query complexity**, the number of queries to the oracle [27, 32]. Various oracles can be used. In particular, we consider the **distance oracle** from several previous works [27, 32]. This oracle takes a pair of vertices and returns the number of edges on a shortest path between the two given vertices. Following previous work [27, 32], the graph is assumed to be connected, undirected, and unweighted. It is also generally assumed to have bounded degree, because distinguishing an n-vertex star from a graph with one additional edge between the leaves of the star would have a trivial quadratic lower bound. For reconstruction from distance oracles, the following results are known:

- For graphs of maximum degree Δ, there is a randomized algorithm with query complexity $O(\Delta^3 \cdot n^{3/2} \cdot \log^2 n \cdot \log\log n)$, which is $\tilde{O}(n^{3/2})$ when $\Delta = O\big(\text{polylog}(n)\big)$ [27].[1]
- Chordal graphs can be reconstructed in randomized query complexity $\tilde{O}(n)$ when $\Delta = O(\log\log n)$ [27]. The k-chordal graphs can be reconstructed in randomized query complexity $O_{\Delta,k}(n \log n)$ when $\Delta = O(1)$ [4].

[1] We use $\tilde{O}(*)$ to denote asymptotic bounds that ignore polylogarithmic factors.

- Outerplanar graphs can be reconstructed in randomized query complexity $\tilde{O}(n)$ when Δ is polylogarithmic [27].
- For reconstructing uniformly random Δ-regular graphs with maximum degree $\Delta = O(1)$, there is a randomized algorithm using $\tilde{O}(n)$ queries in expectation [32].

Our new results in this area are that the graphs of bounded bandwidth can be reconstructed deterministically in query complexity $\tilde{O}(n)$, by a very simple algorithm. More generally, the graphs of BFS width B can be reconstructed deterministically in query complexity $O(nB)$. No additional assumption on Δ is needed here because bounded bandwidth and bounded BFS width both automatically imply bounded degree.

Graph Drawing

Graph drawing is the study of methods for visualizing graphs; see, e.g., [5,30]. One type of graph visualization tool that has received considerable attention is the **arc diagram** of a graph, $G = (V, E)$, where the vertices of G are laid out as points on a straight line and edges are drawn as semicircular arcs or straight-line segments (for consecutive points) joining pairs of such points. See Figure 2.

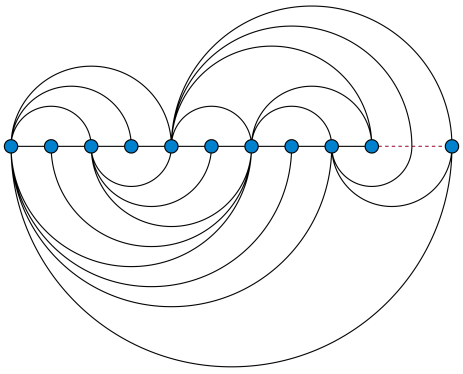


Figure 2 An example arc diagram. Public-domain image by David Eppstein.

Arc diagrams have received considerable attention, both as a visualization tool and as a topic of study in discrete mathematics; see, e.g., [9,11,14,30,31,36]. As a direct consequence of Theorem 14, when an n-vertex graph has an arc diagram of bounded height, we can find a drawing with polylogarithmic height. Here, the height of an arc diagram, with vertices placed at integer positions along the x-axis, is just half the bandwidth of the linear layout describing the vertex placement.

2 Preliminaries

We define BFS width as follows. From a given undirected graph, $G = (V, E)$, choose arbitrarily a starting vertex v in G and perform a breadth-first search of G from v. The edges traversed during this search define a breadth-first search tree T such that the depth of each vertex in T equals its unweighted distance from v, the number of edges in a shortest path from v. We define the **layer** $L_i(v)$ to consist of all vertices at distance i from v; if $u \in L_i(v)$, we say that u has **layer number** i. The maximum size of any $L_i(v)$ set is the BFS width of G from v and the BFS width of G is the maximum BFS width taken over all choices of the starting vertex v in G. That is, we have the following formal definition of BFS width building upon the definition of width in layered graph drawing [5].

▶ **Definition 1** (BFS Width). *Let $G = (V, E)$ be a graph. A **layering** of G is a partition of V into subsets $L_0, L_1, \ldots$, and the **width** of G for this layering is $\max_{i \in \mathbb{Z}_{\geq 0}} |L_i|$. Any vertex, $v \in V$, defines a layering by letting $L_i(v) = \{u \in V : d(v, u) = i\}$. The **BFS width of G from v**, denoted $\mathbf{bfsw}(G, v)$, is defined as $\mathbf{bfsw}(G, v) = \max_{i \in \mathbb{Z}_{\geq 0}} |L_i(v)|$. The **BFS width of G**, denoted $\mathbf{bfsw}(G)$, is defined as $\mathbf{bfsw}(G) = \max_{v \in V} \mathbf{bfsw}(G, v) = \max_{v \in V} \max_{i \in \mathbb{Z}_{\geq 0}} |L_i(v)|$.*

We also define the **minimum BFS width**, $\mathbf{bfsw}_{\min}(G) = \min_{v \in V} \mathbf{bfsw}(G, v)$. These two BFS width parameters can differ significantly, and we will show that there exist graphs such that $\mathbf{bfsw}(G)/\mathbf{bfsw}_{\min}(G) = \Omega(\log n)$. Clearly, $\mathbf{bfsw}(G)$ and $\mathbf{bfsw}_{\min}(G)$ can both easily be computed in $O\big(n(n+m)\big)$ time for any graph, G, with n vertices and m edges by performing n breadth-first searches; see, e.g., [12, 22]. In contrast, it is NP-hard to compute other well-known width parameters for a graph, including bandwidth [18], pathwidth, and treewidth [7]. Nevertheless, we prove that bounded BFS width implies bounded bandwidth for the graph, which, in turn, implies bounded pathwidth and treewidth for the graph.

Some Preliminary Facts About Bandwidth and BFS Width

Bandwidth

Two lower bounds on bandwidth are as follows:

▷ Claim 2 (Degree lower bound [10]). $\mathbf{bw}(G) \geq \lceil \Delta/2 \rceil$, where Δ is the graph's maximum degree.

▷ Claim 3 (Local density lower bound [24]). $\mathbf{bw}(G) \geq D(G) \coloneqq \max_{v \in V, d} \left\lceil \frac{|N(v,d)|}{2d} \right\rceil$, where $D(G)$ is the local density of G, and $N(v, d)$ is vertices at distance at most d from v (excluding v).

When v is a maximum degree vertex and $d = 1$, this implies the degree lower bound.

BFS Width

Recall that a **caterpillar tree** is a tree in which every vertex is within distance 1 of a central path; i.e., the central path contains every vertex of degree 2 or more; see, e.g., [26]. Caterpillar trees have applications in chemistry and physics [17], and they are the graphs of pathwidth one [33]. A simple, but useful, fact regarding BFS width is the following:

▶ **Theorem 4.** *If G is a caterpillar tree with maximum degree, $\Delta \geq 2$, then $\mathbf{bfsw}(G) \leq 2(\Delta - 1)$.*

Proof. To upper bound $\mathbf{bfsw}(G)$, we want to find a root vertex v that can maximize the BFS width of G from v. Choose a vertex, v, that is not one of the end points on the central path as the root for a BFS tree, T. Then layer 1 will have size at most Δ and each vertex of degree Δ in G that is not v will correspond to an interior vertex in T with at most $\Delta - 1$ children. All degree Δ vertices are on the central path. The maximum layer size occurs in T when two such vertices are in the same layer. The layer below it has $2(\Delta - 1)$ children. ◀

We also have the following.

▶ **Theorem 5.** *For any graph, G, its bandwidth $\mathbf{bw}(G) \leq 2\big(\mathbf{bfsw}_{\min}(G)\big) - 1$.*

Proof. Let T be the BFS tree for G rooted at a vertex, v, that achieves the minimum BFS width, $\mathbf{bfsw}_{\min}(G) = \mathbf{bfsw}(G, v)$. By definition, every layer in T has at most $\mathbf{bfsw}_{\min}(G)$ vertices. It is well-known that in a graph, G, with a BFS tree, T, every non-tree edge, (u, v),

connects two vertices whose layer numbers differ by at most 1; see, e.g., [12,22]. Thus, we can number the vertices of T as $1, 2, \ldots, n$ so as to have nondecreasing layer numbers. The BFS ordering is one way to achieve this. This numbering defines a linear layout of the vertices. The maximum distance on the x-axis between two endpoints of an edge, therefore, will be at most that determined by the first vertex in a layer i and the last vertex in a layer $i+1$, which can be at most $2\big(\mathbf{bfsw}_{\min}(G)\big) - 1$. ◀

This immediately implies the following.

▶ **Corollary 6.** *For any graph, G,* $\mathbf{bw}(G) \leq 2\big(\mathbf{bfsw}(G)\big) - 1$.

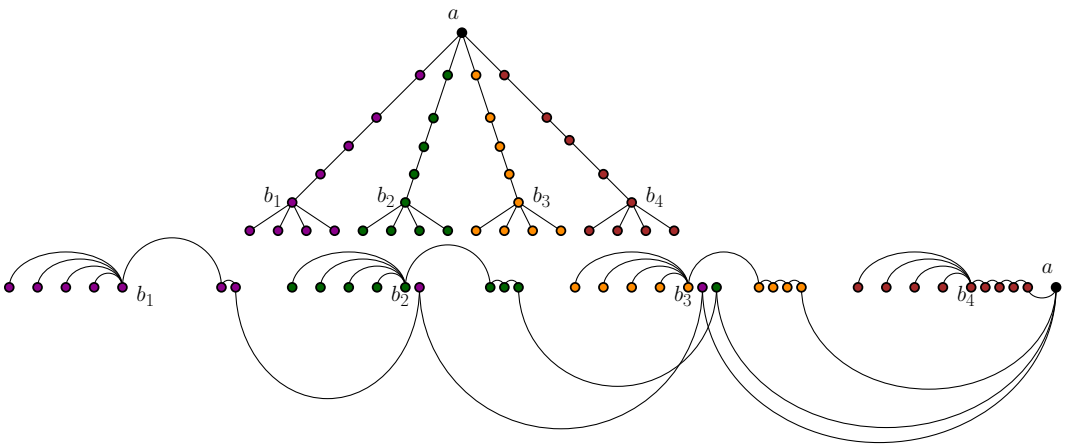


Figure 3 An example graph with $\mathbf{bfsw}(G) = \Theta(n)$ and $\mathbf{bfsw}_{\min}(G) = \Theta(n)$ and a linear layout that achieves its optimal bandwidth $\mathbf{bw}(G) = \Theta(\sqrt{n})$.

Given a linear layout of the vertices of a graph with respect to bandwidth, we say that the **length** of an edge, $\{u, v\}$, is one plus the number of vertices between u and v.

▶ **Theorem 7.** *There is a graph, G, with n vertices such that its bandwidth* $\mathbf{bw}(G) = \Theta(\sqrt{n})$ *but* $\mathbf{bfsw}(G) = \Theta(n)$ *and* $\mathbf{bfsw}_{\min}(G) = \Theta(n)$.

Proof. The example is illustrated in Figure 3. Take a star with $\sqrt{n}$ leaves and subdivide each edge $\sqrt{n}$ times. Then add a star with $\sqrt{n}$ leaves at the end of each path. This example first appeared in [24] but they only gave a lower bound on bandwidth and did not compute the bandwidth. By the degree lower bound, $\mathbf{bw}(G) = \Omega(\sqrt{n})$. Regardless of which vertex is chosen as the root for BFS, BFS will produce a layout with bandwidth $\Theta(n)$. Now it remains to show a linear layout with bandwidth $O(\sqrt{n})$.

The path from a to vertex b_i and the star at b_i are considered as one block and we call this block B_i. Suppose we have already placed $B_{\sqrt{n}-1}, B_{\sqrt{n}-2}, \ldots, B_{i+1}$. Now we describe how to place B_i. All $\sqrt{n}$ leaves of b_i are placed on its left. Next we place the path from a to b_i. Starting from a, we place one vertex to the right of $b_{\sqrt{n}-1}, b_{\sqrt{n}-2}, \ldots, b_{i+1}$. This step uses $\sqrt{n} - i - 1$ vertices. The remaining path segment with $i+1$ vertices are placed to the right of b_i.

Next we analyze the bandwidth. The edge between b_i and its leftmost leaf has length $\sqrt{n}$. Any vertex b_i introduces a path vertex to the right of $b_{\sqrt{n}-1}$. In total there are $\sqrt{n} - 2$ vertices between $b_{\sqrt{n}-1}$ and its parent in the tree. Block $B_{\sqrt{n}}$ has $2\sqrt{n} + 1$ vertices. Thus, the length of the first edge on path ab_1 is at most $4\sqrt{n}$. The linear layout has bandwidth $O(\sqrt{n})$. ◀

This shows that there are graphs for which the linear layout produced by BFS is off by a factor of $\Theta(\sqrt{n})$.

As noted above, there are many algorithmic applications with respect to graphs with bounded bandwidth, which, in turn, implies bounded pathwidth and treewidth. Theorem 5 implies that a graph with bounded minimum BFS width has bounded bandwidth. Thus, any polynomial-time algorithm for graphs with bounded bandwidth, pathwidth, or treewidth (e.g., see [3,8]) also applies to any graph with bounded BFS width for one of its vertices, with the added benefit that we can determine the minimum BFS width for a graph in polynomial time. Unfortunately, as the following theorem shows, bounded treewidth does not imply bounded BFS width.

▶ **Theorem 8.** *There is a graph, G, with n vertices such that its treewidth* $\mathbf{tw}(G) = 1$ *but* $\mathbf{bfsw}(G) = n - 1$ *and* $\mathbf{bfsw}_{\min}(G) = n - 2$.

Proof. Let G be the star graph with internal vertex v. The BFS tree, T, from the vertex v will have $n - 1$ vertices on its layer 1. For any vertex $w \neq v$, the BFS tree, T, from w will have v on its layer 1, and the remaining $n - 2$ vertices of G on its layer 2. ◀

In these subsequent sections, we sometimes drop the graph, G, in width notations when the context is clear. When the graph is a tree with root v, a natural layering is defined by distances to the root and we use the term width to denote the BFS width of G from v.

3 Graphs of Bounded Bandwidth and Polylogarithmic BFS Width

In this section, we show how to construct a graph to prove that when bandwidth is bounded, BFS width can be as large as polylogarithmic in the number of vertices. This graph also proves that BFS width and minimum BFS width can differ by a logarithmic factor. The graph that we use to establish this lower bound is an arbitrarily large binary tree. Our construction is an induction based on defining trees in terms of a level parameter, k, which is the primary inductive term, plus an independent height parameter, h, which determines the size, n, of the tree.

Level 1. A level 1 tree, $T_{1,i}$, is simply a path, which defines a tree when viewed as being rooted at one of the ends of this path. Thus, $\mathbf{bw}(T_{1,i}) = 1$ and $\mathbf{bfsw}(T_{1,i}) = 2$. Before next jumping to the general case, for $k > 1$, let us first describe the level 2 trees, which introduce a constructive pattern we repeat for the general case.

Level 2. A level 2 tree $T_{2,j}$ is defined for height, $h = 2^j$, for any integer, $j > 1$. We start with $j + 1$ level 1 trees, where $T_{1,i}$ has height $h_i = 2^i - 1$, for $i = 0, 1, \ldots, j$. We also define another level 1 tree of height $h = 2^j$, which we call the **spine**. We connect each tree, $T_{1,i}$, with an edge to a vertex v_i on the spine so that every leaf in the resulting tree has the same depth. See Figure 4.

Thus, each subtree "hanging off" the spine is a path with height $h_i = 2^i - 1$, where $i = 0, 1, \ldots, j$. The height of the level 2 tree is $h = 2^j$, where j is independent of the level number 2. As mentioned above, all leaves in a level 2 tree have the same depth; hence, the number of edges on the spine between two consecutive roots must be the height difference between the corresponding subtrees: $d(v_{i+1}, v_i) = h_{i+1} - h_i = 2^i = h_i + 1$, where d is the shortest path distance. Therefore, when we interleave a subtree, $T_{1,i}$, and the spine edges between v_{i+1} and v_i in the linear layout, we can alternately take one vertex from each. This is illustrated in Figure 4. This layout has bandwidth 2, which is the minimum possible.

Let us now consider the width (BFS width from the root) of a level 2 tree, $T_{2,j}$. This width lower bounds BFS width. Since $h = 2^j$, $j = \log h$ where log is base 2. The total number of subtrees is $j + 1$, each contributing one vertex to the widest layer. The spine also

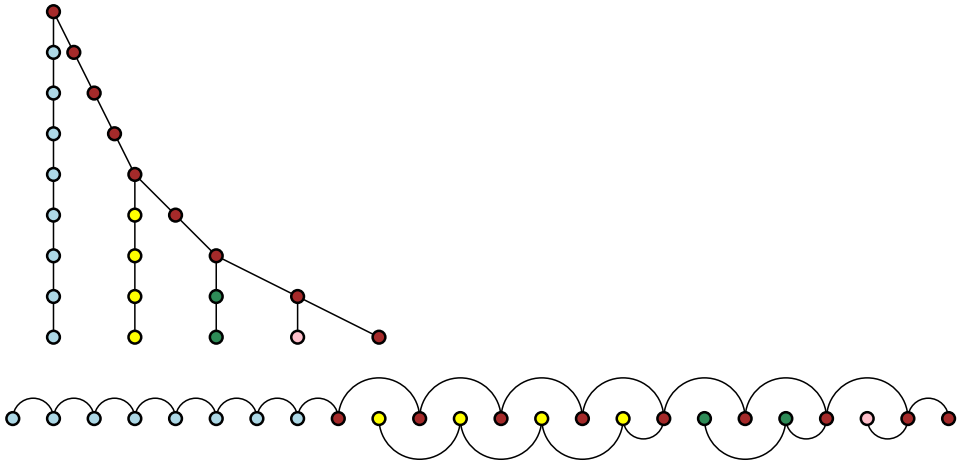


Figure 4 Top: the level 2 tree is constructed by attaching level 1 trees on a spine. The heights of these level 1 subtrees approximately follow a geometric sequence. Bottom: an arc diagram illustrating the linear layout of the level 2 tree. Each edge is a semicircle. The maximum diameter of these semicircles equals the bandwidth of the linear layout. This linear layout achieves the minimum bandwidth 2.

has one vertex at the widest layer. Therefore, the width of $T_{2,j}$ is $w = j + 1 + 1 = \log h + 2$. We want to express this width in terms of the number of vertices, n. Any vertex of this tree is either in a subtree or on the spine. The spine has height h and $h + 1$ vertices. Subtrees have $n_s = \sum_{i=0}^{j}(h_i + 1) = \sum_{i=0}^{j} 2^i = 2h - 1$ vertices. Thus, $n = h + 1 + 2h - 1 = 3h$; hence, we can define a level 2 tree $T_{2,j}$ to have an arbitrarily large number, n, of vertices. Therefore, $T_{2,j}$ has bandwidth $\mathbf{bw}(T_{2,j}) = 2$, and width, $w = j+2$, which implies $\mathbf{bfsw}(T_{2,j}) = \Omega(\log n)$, where $n = 3h = 3 \cdot 2^j$ can be arbitrarily large.

Following the structural pattern we used to build a level 2 tree, to construct a tree at level k, we take trees from level $(k - 1)$ whose heights approximately follow a geometric sequence and attach them to a path, which we again call the **spine**. These level $(k-1)$ trees are called **subtrees**. Their placement on the spine is such that the leaves of these trees have the same depth in the level k tree, thus achieving their width in a single (last) level of a breadth-first layering. To get a bounded bandwidth layout of the level k tree, we interleave vertices from the spine and level $(k - 1)$ subtrees. These constructions are illustrated in Figure 5.

Let us, therefore, consider the bandwidth and BFS width of a level k tree. Suppose the subtree below vertex v_i (excluding v_i) is $T_{k-1,i}$. Given this notational background, let us prove the following theorem, which is one of the two main technical theorems in this paper:

▶ **Theorem 9** (Lower bound on BFS width)**.** *For each non-negative integer, k, and for arbitrarily large values of n, there exists a (level k) tree, T, with n vertices that satisfies:*

$$\mathbf{bw}(T) = O_k(1),$$
$$\mathbf{bfsw}(T) = \Omega(\log^k n),$$

where $O_k(1)$ denotes a number that depends on k but is bounded when k is bounded.

Proof. We construct the level k tree to have an arbitrarily large integer, n, of vertices, by induction, from level $(k - 1)$ trees. To keep track of the tree and subtree sizes, we let $n_{k,j}$ denote the size of the tree, $T_{k,j}$, we are constructing (to roughly have height that is arbitrarily large as a power of j). The base case is the construction given above, which in this notation,

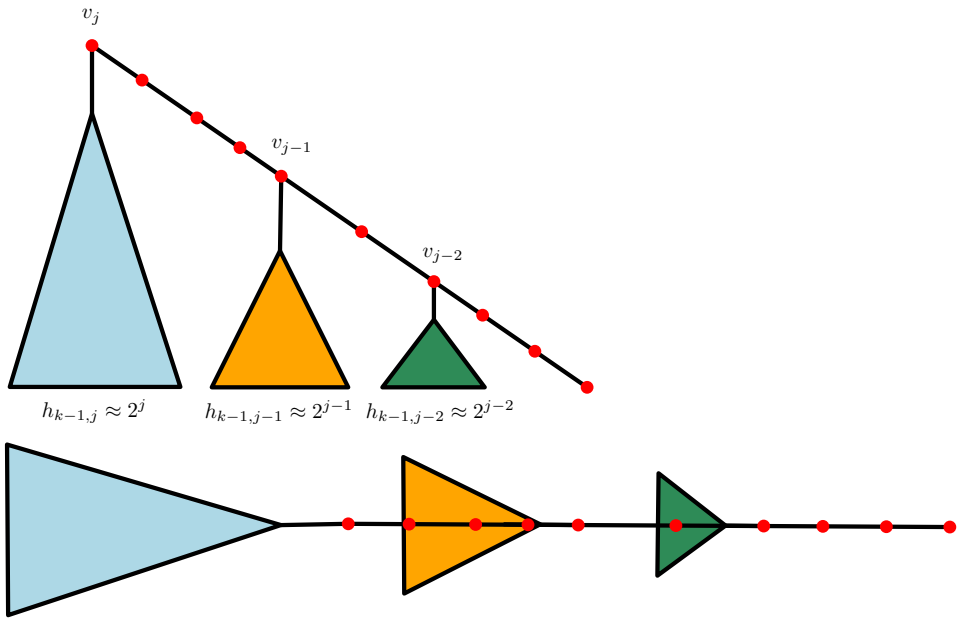


Figure 5 Top: the level k tree is constructed by attaching level $(k-1)$ trees on a spine. The heights of these level $(k-1)$ subtrees approximately follow a geometric sequence. The leaves of these subtrees have the same depth in the level k tree. Bottom: a diagram illustrating the linear layout of the level k tree. This does not necessarily achieve the minimum possible bandwidth. Intuitively, this is the level k tree rooted at the rightmost vertex of the spine. We interleave vertices from the spine and level $(k-1)$ subtrees. Vertices from a level $(k-1)$ subtree are arranged according to their linear layout from the previous level.

gives a level 2 tree with $n_{2,j} = 3h_{2,j}$ vertices and height $h_{2,j} = 2^j$ and to have bandwidth 2 and BFS width at least $d_2 j + 2$, for the constant, $d_2 = 1$. Accordingly, in our analysis below, we focus on the dominant components sufficient for our asymptotic analysis.

Inductive step. Suppose, for $i = 0, 1, 2, \ldots, j$, we have a level $(k-1)$ tree, $T_{k-1,i}$, with height $h_{k-1,i} = 2^i + k - 3$ having width $w_{k-1,i}$ and $n_{k-1,i}$ vertices such that

$$n_{k-1,i} = c_{k-1} \cdot 2^i + p_{k-1}(i), \qquad w_{k-1,i} = d_{k-1} i^{k-2} + q_{k-1}(i), \qquad \mathbf{bw}'(T_{k-1,i}) = O_{k-1}(1),$$

where $\mathbf{bw}'$ is the bandwidth of the linear layout we construct, c_{k-1} and d_{k-1} are constants, and p_{k-1} and q_{k-1} are polynomial functions with degree $k-3$. Let us now construct a level k tree where the height of the leftmost subtree is $h_{k-1,j} = 2^j + k - 3$, so this level k tree has height $h_{k,j} = 2^j + k - 2$. Using Faulhaber's formula (e.g., see Knuth [29]), we can characterize the total number of vertices in all the subtrees as

$$n_s = \sum_{i=0}^{j} (c_{k-1} \cdot 2^i + p_{k-1}(i)) = c_{k-1} \cdot (2^{j+1} - 1) + p_k(j),$$

where p_k is a polynomial function with degree $k-2$. The spine has $h_{k,j} + 1$ vertices. The total number of vertices in this level k tree, therefore, is

$$\begin{aligned} n_{k,j} &= n_s + h_{k,j} + 1 = c_{k-1} \cdot (2^{j+1} - 1) + p_k(j) + 2^j + k - 2 + 1 \\ &= (2c_{k-1} + 1) \cdot 2^j + p'_k(j) = c_k \cdot 2^j + p'_k(j), \end{aligned}$$

where p'_k is a polynomial function with degree $k-2$, and $c_k \coloneqq 2c_{k-1} + 1$. Since $c_2 = 3$, we can solve this recurrence relation to yield $c_k = 2^k - 1$.

Bandwidth. Let us next upper bound the bandwidth, $\mathbf{bw}(T_{k,j})$, of the tree, $T_{k,j}$. We interleave each subtree $T_{k-1,i}$ and the spine path between v_{i+1} and v_i in the linear layout. $T_{k-1,i}$ has height $h_{k-1,i}$ and $n_{k-1,i}$ vertices. The spine path $v_{i+1}v_i$ has length $d(v_{i+1}, v_i) = h_{k-1,i+1} - h_{k-1,i} = 2^i$. We use the linear layout of subtree $T_{k-1,i}$ obtained from the previous level and insert vertices from the spine path. Let $r_i \coloneqq n_{k-1,i}/2^i = c_{k-1} + p_{k-1}(i)/2^i$. We alternately place $\lceil r_i \rceil$ vertices from $T_{k-1,i}$ and one vertex from the spine path $v_{i+1}v_i$, until all vertices from $T_{k-1,i}$ have been placed, in which case we place the remaining vertices from $v_{i+1}v_i$ consecutively. Any edge connecting two vertices from $T_{k-1,i}$ with length equal to $\mathbf{bw}'(T_{k-1,i})$ may have at most $\left\lceil \frac{\mathbf{bw}'(T_{k-1,i})}{\lceil r_i \rceil + 1} \right\rceil$ spine vertices inside it. Any edge connecting two vertices from the spine path $v_{i+1}v_i$ has at most $\lceil r_i \rceil$ vertices from $T_{k-1,i}$ inside it. These are the only two factors that cause the bandwidth to increase from level $(k-1)$ to level k, so the recurrence relation is

$$\mathbf{bw}'(T_{k,j}) \leq \max_{i=0}^{j} \max \left(\mathbf{bw}'(T_{k-1,i}) + \left\lceil \frac{\mathbf{bw}'(T_{k-1,i})}{\lceil r_i \rceil + 1} \right\rceil, \lceil r_i \rceil + 1 \right).$$

Since $\lim_{i \to \infty} r_i = c_{k-1}$, for large values of i, $r_i < c_{k-1} + 1$, and for small values of i, r_i can be upper bounded by another constant. Thus, this level k linear layout has bandwidth $O_k(1)$.

BFS width. BFS width is lower bounded by width, which can be computed using Faulhaber's formula (e.g., see Knuth [29]):

$$w_{k,j} = \sum_{i=0}^{j} (d_{k-1} \cdot i^{k-2} + q_{k-1}(i)) = \frac{d_{k-1}}{k-1} j^{k-1} + q_k(j) = d_k j^{k-1} + q_k(j),$$

where $d_k \coloneqq \frac{d_{k-1}}{k-1}$ and q_k is a polynomial function with degree $k-2$. Since $d_2 = 1$, we can solve this recurrence relation to yield $d_k = \frac{1}{(k-1)!}$.

We have therefore shown that the tree $T_{k,j}$ has height $h_{k,j} = 2^j + k - 3$ and $n_{k,j}$ vertices, with

$$\begin{aligned} n_{k,j} &= c_k \cdot 2^j + p'_k(j), \\ \mathbf{bw}(T_{k,j}) &= O_k(1), \quad \text{and} \\ \mathbf{bfsw}(T_{k,j}) &\geq w_{k,j} = d_k j^{k-1} + q_k(j) = \Omega(\log^{k-1} n_{k,j}). \end{aligned}$$

◀

Thus, a graph with bounded bandwidth can have polylogarithmic BFS width. Also:

▶ **Theorem 10** (Lower bound on minimum BFS width). *For each non-negative integer, k, and for arbitrarily large values of n, there exists a (level k) tree, T, with n vertices that satisfies:*

$$\begin{aligned} \mathbf{bw}(T) &= O_k(1), \\ \mathbf{bfsw}_{\min}(T) &= \Omega(\log^k n), \end{aligned}$$

where $O_k(1)$ denotes a number that depends on k but is bounded when k is bounded.

Proof. A construction similar to Figure 5 can be used to place the root at one end of the linear layout. Taking two reflected copies of the resulting tree, connected at the root, gives a lower bound for minimum BFS width (with different constants than Theorem 9). ◀

▶ **Theorem 11.** *For arbitrarily large values of n, there exists a binary tree, T, with n vertices that satisfies* $\mathbf{bfsw}(T)/\mathbf{bfsw}_{\min}(T) = \Omega(\log n)$.

Proof. The level 2 tree, $T = T_{2,j}$, is originally rooted at the leftmost spine vertex v_j. Consider rooting T at the rightmost spine vertex v_s, as in the bottom of Figure 4. The width of the new tree is 2. Thus,

$$\begin{aligned}
&\mathbf{bfsw}(T, v_j) = j + 2,\\
&\mathbf{bfsw}(T, v_s) = 2,\\
&\frac{\mathbf{bfsw}(T)}{\mathbf{bfsw}_{\min}(T)} \geq \frac{\mathbf{bfsw}(T, v_j)}{\mathbf{bfsw}(T, v_s)} = \frac{j+2}{2} = \frac{\log\left(\frac{n}{3}\right) + 2}{2}, \quad \text{and}\\
&\frac{\mathbf{bfsw}(T)}{\mathbf{bfsw}_{\min}(T)} = \Omega(\log n).
\end{aligned}$$

◀

4 Polylogarithmic BFS Width for Graphs of Bounded Bandwidth

In this section, we show that if bandwidth is bounded, then BFS width is at most polylogarithmic in the number of vertices. The construction in Section 3 shows that this upper bound is tight. The proof follows a structure similar to our lower-bound construction. First, we make several simplifying assumptions with respect to the graph, which we make without loss of generality (wlog). Next, we identify spines and subtrees as in our construction above. Then we use strong induction to prove our bound.

The inductive hypothesis is as follows: for arbitrarily large values of n, any graph on n vertices with bandwidth $\mathbf{bw} \leq k$, for a fixed constant, $k \geq 1$, has BFS width $\mathbf{bfsw} = O\big(\log^{k-1}(n)\big)$. The base case, $k = 1$, is clearly true, since a connected graph with bandwidth 1 is a path. For the induction step, suppose that if any graph, G, on n vertices has bandwidth $\mathbf{bw}(G) \leq k$, for a fixed constant, $k \geq 1$, then $\mathbf{bfsw}(G) = O\big(\log^{k-1}(n)\big)$. To show the induction hypothesis holds for $k+1$, we consider a widest BFS tree of G, and show that there are at most a logarithmic number of subtrees in T and each subtree has width $O\big(\log^{k-1}(n)\big)$. Consequently, when these subtrees are combined, the overall width remains polylogarithmic, with a higher (constant) exponent.

Suppose we have a linear layout of an n-vertex graph, G, that achieves minimum bandwidth, let T be a widest BFS tree of G, and let r be the root of T. We want to show that T has width $O\big(\log^{k}(n)\big)$. Thus, without changing the BFS width, we can delete non-tree edges from G as well as all vertices and edges below the first widest layer of T. This may reduce the number of vertices, but, wlog, let us nevertheless refer to the reduced number of vertices as n and the reduced tree as T, since we are interested in an upper bound in terms of the number of vertices in the original graph. Thus, let us focus our attention on the reduced tree, T, instead of the original graph.

Furthermore, we can make extreme points of the linear layout leaves of T. Suppose this is not the case. Wlog, we consider two scenarios:

1. The leftmost vertex v of the linear layout is the root. If the root only has one child, then we can remove it and make its child the new root. After repeating this step, either the new leftmost vertex is not the root, or it is the root with at least two children. In the latter case, we can take the subtree rooted at one child and flip its linear layout around the root. Thus, extreme points of the linear layout will not be the root.
2. The rightmost vertex v of the linear layout is neither the root nor a leaf. All leaves of the subtree T_v rooted at v are to the left of v in the linear layout. We can flip this subtree around v to move non-root vertices of T_v to the right side of v. After this, we check the new rightmost vertex $w \in T_v$. If it is not a leaf, then $T_w \subset T_v$ and $v \notin T_w$. Since the subtree T_w contains fewer vertices than T_v, each iteration of this process reduces the size of the subtree rooted at the rightmost vertex. Therefore, after repeating this process finitely many times, the rightmost vertex will be a leaf.

After these operations, extreme points of the linear layout are now leaves of T. Note that the width of the tree does not change and the bandwidth of this new ordering cannot be greater than before. The **left spine** is defined as the path from the root to the leftmost vertex, and the **right spine** is defined similarly. All non-spine vertices belong to a **subtree** "hanging off" the spine. In this section, it is more convenient to treat the root vertices of these subtrees as the intersection of subtrees and the spine. Now we upper bound the number of subtrees that contribute to the width of T.

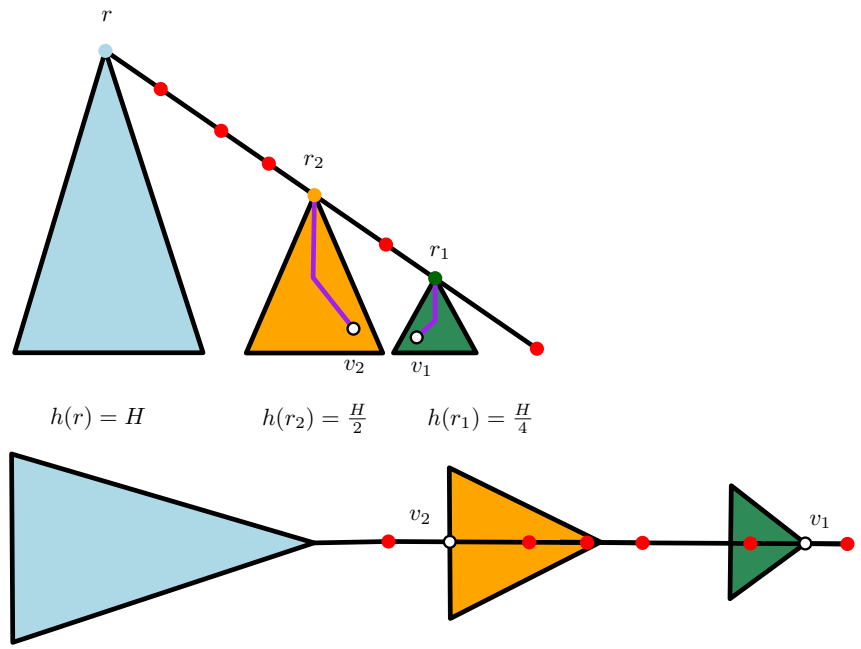


Figure 6 Proof of Lemma 12 illustrated. When $i = 1$, S_1 is the set of all subtrees with height $h(r_s) \in I := [\frac{H}{4}, \frac{H}{2}]$. In this figure, it contains the two subtrees on the right. We want to upper bound the total number of vertices in these subtrees.

▶ **Lemma 12.** *The number of subtrees that have at least one leaf at the widest layer of the tree T is $O(\log n)$.*

Proof. Because of symmetry, we only bound the number of subtrees whose root is on the right spine. Suppose the tree T rooted at r has height H. For integer $i = 0, 1, \ldots, \lfloor \log H \rfloor$, let S_i be the set of all subtrees with height $h(r_s) \in I := [\frac{H}{2^{i+1}}, \frac{H}{2^i}]$ where r_s is the root of a subtree. See Figure 6. We do not need to consider subtrees with height 0.

Now we upper bound the total number of vertices in all subtrees in S_i. Consider the position of these subtrees in the linear layout. Suppose the rightmost vertex is v_1 and it belongs to the subtree rooted at r_1, and the leftmost vertex v_2 belongs to the subtree rooted at r_2. Recall that d denotes the shortest path distance in the graph. For any root r_s of a subtree in S_i, $d(r, r_s) = H - h(r_s) \in H - I := [H - \frac{H}{2^i}, H - \frac{H}{2^{i+1}}]$. The shortest paths from r to all such root vertices are part of the right spine. Thus,

$$
\begin{aligned}
&d(r_1, r_2) \le |H - I| = |I| \\
&d(v_1, v_2) \le d(v_1, r_1) + d(r_1, r_2) + d(r_2, v_2) \le h(v_1) + |I| + h(v_2) \\
&\qquad \le \frac{H}{2^i} + \left(\frac{H}{2^i} - \frac{H}{2^{i+1}}\right) + \frac{H}{2^i} = \frac{3H}{2^i} - \frac{H}{2^{i+1}} = \frac{5H}{2^{i+1}}.
\end{aligned}
$$

Because of the bounded bandwidth, the number of vertices between v_1 and v_2 in the linear layout is at most $\frac{5H}{2^{i+1}}(\mathbf{bw})$.

Each subtree with height in I must have at least $\frac{H}{2^{i+1}}$ vertices to reach the minimum height. Consequently, the total number of subtrees with height in any given interval I is at most a constant and this completes the proof. ◀

Next, we prove that removing spine vertices from the linear layout reduces the bandwidth of subtrees. Since the roots of these subtrees also belong to the spine, each subtree may decompose into a forest. Given that bounded bandwidth implies bounded maximum degree, the number of trees in each resulting forest is bounded. If these trees have smaller bandwidth, we can apply the inductive hypothesis to show that they have width $O\big(\log^{k-1}(n)\big)$. Thus, the original subtree also has width $O\big(\log^{k-1}(n)\big)$. Thus, the following lemma is essential to the induction:

▶ **Lemma 13.** *If the tree T has bandwidth* **bw***, then each tree in the forests obtained after removing the spine has bandwidth at most* $\mathbf{bw} - 1$.

Proof. The proof outline is as follows. By definition, the extreme points of the original linear layout are spine vertices and thus subtrees with one vertex. We denote them as v_l and v_r respectively. Consider any subtree T_v rooted at $v \neq v_l, v_r$. After removing the spine vertices, the linear layout for T naturally defines a linear layout for $T_v \setminus v$ by preserving the relative position of its vertices. We show that the maximum length of any edge of $T_v \setminus v$ will become less than **bw**.

In the linear layout for T, the length of any edge is at most **bw**. If an edge has length equal to **bw**, we say that this edge is saturated. Let S be the set of all saturated edges of $T_v \setminus v$. If this is an empty set, the proof is complete. Otherwise, suppose there is one edge $e \in S$ with no spine vertex inside it. v_l and v_r are connected through edges of the spine and one of these edges must enclose the two endpoints of e. The length of e is **bw**, so the length of this edge is greater than **bw**, and this is a contradiction. Thus, there is a spine vertex inside every saturated edge of $T_v \setminus v$. After removing the spine vertices, all edges of $T_v \setminus v$ have length at most $\mathbf{bw} - 1$. $T_v \setminus v$ may be a forest, so this upper bounds the bandwidth of each connected component. ◀

With these lemmas, we can apply strong induction on bandwidth, which gives us:

▶ **Theorem 14** (Upper bound on BFS width)**.** *For any constant, $k \geq 1$, and n-vertex graph, G, if* $\mathbf{bw}(G) = k$*, then* $\mathbf{bfsw}(G) = O\big(\log^{k-1}(n)\big)$.

5 Applications

In this section, we prove the application results claimed in the introduction.

5.1 Numerical Linear Algebra

For the purposes of this section, the only facts we need about the Cuthill–McKee and reverse Cuthill–McKee algorithms are:

- The bandwidth of a reordered matrix PAP^T, where A is any symmetric matrix and P is any permutation matrix, is the same as the bandwidth of the linear layout of the graph of nonzeros of A obtained by laying out the vertices of this graph in the order given by permutation P.
- Both the Cuthill–McKee and reverse Cuthill–McKee algorithms, applied to a matrix A choose their orderings (interpreted as linear layouts of graphs) by the breadth-first layer of the graph of nonzeros of A, and then carefully permute the vertices within each layer.

We can directly relate the performance of these algorithms to BFS width:

▶ **Theorem 15.** *Let G be the graph of nonzeros of a symmetric matrix A, and let v be an arbitrary vertex of G. Then the Cuthill–McKee and reverse Cuthill–McKee algorithms, using a breadth-first layering rooted at v, produce layouts of bandwidth $\Theta\big(\mathbf{bfsw}(G,v)\big)$.*

Proof. Let u be a vertex in the widest layer of the breadth-first layering of G, chosen among the vertices of this layer to be the one that the Cuthill–McKee or reverse Cuthill–McKee algorithm places farthest from the previous layer. Then the positions of u and its BFS ancestor differ by at least $\mathbf{bfsw}(G,v)$, so the bandwidth of the layout is at least $\mathbf{bfsw}(G,v)$. Every edge in G extends over at most two consecutive layers, and has endpoints whose positions differ by at most $2\mathbf{bfsw}(G,v)-1$, so the bandwidth of the layout is at most $2\mathbf{bfsw}(G,v)-1$. ◀

By combining Theorem 15 with Theorem 14, we have:

▶ **Theorem 16.** *If a symmetric matrix A can be reordered to have bounded bandwidth, the Cuthill–McKee and reverse Cuthill–McKee algorithms will produce a reordered matrix PAP^T of at most polylogarithmic bandwidth.*

Note that this theorem holds even if we do not know the optimal bandwidth of the matrix A. Moreover, the bound of Theorem 16 is is tight, up to the dependence of the polylogarithmic bound on the bandwidth; by combining Theorem 15 with Theorem 10, we have:

▶ **Theorem 17.** *For any k, there exist $n \times n$ symmetric matrices A of bounded bandwidth (for arbitrarily large n) for which the Cuthill–McKee and reverse Cuthill–McKee algorithms produce reordered matrices PAP^T of bandwidth $\Omega(\log^k n)$.*

Theorems 16 and 17 provide the first non-trivial deterministic analyses for the Cuthill-McKee algorithm and its reversal.

5.2 Graph Reconstruction

To reconstruct graphs of low bandwidth or low BFS width, we apply Algorithm 1.

Algorithm 1 Reconstructing graphs with low BFS width.

```
v ← an arbitrary vertex in V                          ▷ this is the root of the BFS tree
for u ∈ V \ {v} do
    Query(v, u)                                       ▷ this is a distance query
Ê ← {{a, b} : {a, b} ⊆ V \ {v} ∧ |d(v, a) − d(v, b)| ≤ 1}
for {a, b} ∈ Ê do
    Query(a, b)
return {{u, v} : u ∈ V ∧ d(v, u) = 1} ∪ {{a, b} : {a, b} ∈ Ê ∧ d(a, b) = 1}
```

Less formally, Algorithm 1 first picks a vertex v as the root of the BFS tree. Then, it obtains distances from this root to every other vertex to determine their layer number. It constructs the set of all vertex pairs from the same layer or in two consecutive layers, and queries each such pair. Recall that in the proof of Theorem 5, non-tree edges connect two vertices whose layer numbers differ by at most one, so each pair of adjacent vertices in G must belong to the set of queried pairs. The algorithm then returns the set of all pairs for which a

query found two vertices to be at distance one. [4] used a similar approach to reconstruct trees and k-chordal graphs. They divided the graph into BFS layers and reconstructed edges in the same layer and in two consecutive layers, but querying each pair would incur a high query complexity for those graph classes.

▶ **Theorem 18.** *For any graph G with n vertices and BFS width B, Algorithm 1 reconstructs G with deterministic query complexity $O(nB)$.*

Proof. It takes $n-1$ queries to find the distances from the chosen root v and partition the vertices into layers. Then, the algorithm makes at most $3B-1$ queries involving each remaining vertex, between it and at most $3B-1$ other vertices in its layer and the two adjacent layers. This double-counts the queries, so the total number of queries is at most $n-1+(n-1)(3B-1)/2 = O(nB)$. ◀

▶ **Corollary 19.** *For any constant bound b on the bandwidth of graphs, Algorithm 1 reconstructs n-vertex graphs of bandwidth $\leq b$ with deterministic query complexity $\tilde{O}(n)$.*

Proof. This follows immediately from Theorem 18 and from the relation between bandwidth and BFS width of Theorem 14. ◀

5.3 Graph Drawing

In our graph drawing application, we specifically focus on arc diagrams, which were used to study the crossing number problem [31] and visualize structure in strings [36]. In an arc diagram, vertices of a graph are positioned along a straight (e.g., horizontal) line, and edges are semicircles drawn on either side of the line [31], joining pairs of points (possibly using straight-line segments for consecutive points). Examples are shown above in Figures 2 and 4.

To analyze such diagrams, we constrain the vertices to be placed at consecutive integer coordinates. Thus, the horizontal line is a linear layout of the vertices. The drawing fits within a bounding box $n-1$ units wide and $b/2$ units tall, where n is the number of vertices in the graph and b is the bandwidth of the given layout.

We immediately obtain the following result:

▶ **Theorem 20.** *Suppose that a given graph G has n vertices and has an arc diagram of bounded height h. Construct an arc diagram for G by ordering the vertices by their breadth-first layer numbers, starting from an arbitrary root vertex. Then if h is bounded by a constant, the height of the arc diagram that we construct will be bounded by $O\big(\text{polylog}(n)\big)$.*

References

1 Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda. Mapping networks via parallel kth-hop traceroute queries. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:21, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.STACS.2022.4.

2 Ramtin Afshar, Michael T. Goodrich, and Evrim Ozel. Efficient exact learning algorithms for road networks and other graphs with bounded clustering degrees. In Christian Schulz and Bora Uçar, editors, *20th International Symposium on Experimental Algorithms (SEA 2022)*, volume 233 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.SEA.2022.9.

3 Mugurel Ionut Andreica. A dynamic programming framework for combinatorial optimization problems on graphs with bounded pathwidth, 2012. arXiv:0806.0840.

4 Paul Bastide and Carla Groenland. Optimal distance query reconstruction for graphs without long induced cycles, October 2024. arXiv:2306.05979 [cs]. doi:10.48550/arXiv.2306.05979.

5 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

6 Hans L. Bodlaender, John R. Gilbert, Hjálmtýr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *J. Algorithms*, 18(2):238–255, 1995. doi:10.1006/jagm.1995.1009.

7 Hans L. Bodlaender and Ton Kloks. Efficient and constructive algorithms for the pathwidth and treewidth of graphs. *Journal of Algorithms*, 21(2):358–402, 1996. doi:10.1006/jagm.1996.0049.

8 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, May 2008. doi:10.1093/comjnl/bxm037.

9 Michael Burch, Kiet Bennema Ten Brinke, Adrien Castella, Ghassen Karray Sebastiaan Peters, Vasil Shteriyanov, and Rinse Vlasvinkel. Dynamic graph exploration by interactively linked node-link diagrams and matrix visualizations. *Visual Computing for Industry, Biomedicine, and Art*, 4:1–14, 2021. doi:10.1186/s42492-021-00088-8.

10 Julia Böttcher, Klaas P. Pruessmann, Anusch Taraz, and Andreas Würfl. Bandwidth, expansion, treewidth, separators and universality for bounded-degree graphs. *European Journal of Combinatorics*, 31(5):1217–1227, July 2010. doi:10.1016/j.ejc.2009.10.010.

11 Jean Cardinal, Michael Hoffmann, Vincent Kusters, Csaba D. Tóth, and Manuel Wettstein. Arc diagrams, flip distances, and Hamiltonian triangulations. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 30 of *LIPIcs*, pages 197–210. Schloss Dagstuhl, 2015. doi:10.4230/LIPIcs.STACS.2015.197.

12 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 4th edition, 2022.

13 E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, ACM '69, pages 157–172, New York, NY, USA, August 1969. Association for Computing Machinery. doi:10.1145/800195.805928.

14 Alberto Debiasi, Bruno Simoes, and Raffaele De Amicis. Schematization of node-link diagrams and drawing techniques for geo-referenced networks. In *International Conference on Cyberworlds (CW)*, pages 34–41, 2015. doi:10.1109/CW.2015.68.

15 Chandan Dubey, Uriel Feige, and Walter Unger. Hardness results for approximating the bandwidth. *Journal of Computer and System Sciences*, 77(1):62–90, January 2011. doi:10.1016/j.jcss.2010.06.006.

16 John Dunagan and Santosh Vempala. On euclidean embeddings and bandwidth minimization. In Michel Goemans, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques*, pages 229–240, Berlin, Heidelberg, 2001. Springer. sentenceCase:1. doi:10.1007/3-540-44666-4_26.

17 Sherif El-Basil. Applications of caterpillar trees in chemistry and physics. *Journal of Mathematical Chemistry*, 1(2):153–174, July 1987. doi:10.1007/BF01205666.

18 Uriel Feige. Coping with the NP-hardness of the graph bandwidth problem. In Magnús M. Halldórsson, editor, *Algorithm Theory - SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, Bergen, Norway, July 5-7, 2000, Proceedings*, volume 1851 of *Lecture Notes in Computer Science*, pages 10–19. Springer, 2000. doi:10.1007/3-540-44985-X_2.

19 Uriel Feige, Mohammadtaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum-weight vertex separators. In *Proc. 37th ACM Symposium on Theory of Computing (STOC 2005)*, pages 563–572, 2005. doi:10.1145/1060590.1060674.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

21 Alan George and Joseph W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, 1981.

22 Michael T. Goodrich and Roberto Tamassia. *Algorithm Design and Applications*. Wiley, 2015.

23 Hermann Gruber. On balanced separators, treewidth, and cycle rank. *J. Combinatorics*, 3(4):669–681, 2012. doi:10.4310/JOC.2012.v3.n4.a5.

24 Anupam Gupta. Improved bandwidth approximation for trees and chordal graphs. *Journal of Algorithms*, 40(1):24–36, July 2001. doi:10.1006/jagm.2000.1118.

25 Eitan M. Gurari and Ivan Hal Sudborough. Improved dynamic programming algorithms for bandwidth minimization and the MINCUT linear arrangement problem. *Journal of Algorithms*, 5(4):531–546, 1984. doi:10.1016/0196-6774(84)90006-3.

26 Frank Harary and Allen J. Schwenk. The number of caterpillars. *Discrete Mathematics*, 6(4):359–365, 1973. doi:10.1016/0012-365X(73)90067-8.

27 Sampath Kannan, Claire Mathieu, and Hang Zhou. Graph reconstruction and verification. *ACM Transactions on Algorithms*, 14(4):40:1–40:30, August 2018. doi:10.1145/3199606.

28 Haim Kaplan and Ron Shamir. Pathwidth, bandwidth, and completion problems to proper interval graphs with small cliques. *SIAM J. Comput.*, 25(3):540–561, 1996. doi:10.1137/S0097539793258143.

29 Donald E. Knuth. Johann Faulhaber and sums of powers. *Mathematics of Computation*, 61(203):277–294, 1993. doi:10.2307/2152953.

30 Ales Komarek, Jakub Pavlik, and Vladimir Sobeslav. Network visualization survey. In Manuel Núñez, Ngoc Thanh Nguyen, David Camacho, and Bogdan Trawiński, editors, *Computational Collective Intelligence: 7th International Conference, ICCCI 2015, Madrid, Spain, September 21–23, 2015, Proceedings, Part II*, volume 9330 of *Lecture Notes in Computer Science*, pages 275–284, Cham, 2015. Springer. doi:10.1007/978-3-319-24306-1_27.

31 S. Masuda, K. Nakajima, T. Kashiwabara, and T. Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Transactions on Computers*, 39(1):124–127, January 1990. doi:10.1109/12.46286.

32 Claire Mathieu and Hang Zhou. A simple algorithm for graph reconstruction. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 68:1–68:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.ESA.2021.68.

33 Andrzej Proskurowski and Jan Arne Telle. Classes of graphs with restricted interval models. *Discrete Mathematics & Theoretical Computer Science*, 3(4):167–176, 1999. doi:10.46298/dmtcs.263.

34 Ilya Safro, Dorit Ron, and Achi Brandt. Multilevel algorithms for linear ordering problems. *ACM J. Exp. Algorithmics*, 13:Article 1.4, 2009. doi:10.1145/1412228.1412232.

35 Jonathan S. Turner. On the probable performance of heuristics for bandwidth minimization. *SIAM Journal on Computing*, 15(2):561–580, May 1986. doi:10.1137/0215041.

36 M. Wattenberg. Arc diagrams: visualizing structure in strings. In *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.*, pages 110–116, October 2002. ISSN: 1522-404X. doi:10.1109/INFVIS.2002.1173155.