A Fast Minimum Degree Algorithm and Matching Lower Bound

Robert Cummings[*] Matthew Fahrbach[†] Animesh Fatehpuria[‡]

Abstract

The minimum degree algorithm is one of the most widely-used heuristics for reducing the cost of solving large sparse systems of linear equations. It has been studied for nearly half a century and has a rich history of bridging techniques from data structures, graph algorithms, and scientific computing. In this paper, we present a simple but novel combinatorial algorithm for computing an exact minimum degree elimination ordering in $O(nm)$ time, which improves on the best known time complexity of $O(n^3)$ and offers practical improvements for sparse systems with small values of m. Our approach leverages a careful amortized analysis, which also allows us to derive output-sensitive bounds for the running time of $O(\min(m\sqrt{m^+}, \Delta m^+) \log n)$, where m^+ is the number of unique fill edges and original edges that the algorithm encounters and Δ is the maximum degree of the input graph.

Furthermore, we show there cannot exist an exact minimum degree algorithm that runs in $O(nm^{1-\varepsilon})$ time, for any $\varepsilon > 0$, assuming the strong exponential time hypothesis. This fine-grained reduction goes through the orthogonal vectors problem and uses a new low-degree graph construction called *U-fillers*, which act as pathological inputs and cause any minimum degree algorithm to exhibit nearly worst-case performance. With these two results, we nearly characterize the time complexity of computing an exact minimum degree ordering.

1 Introduction

The minimum degree algorithm is one of the most widely-used heuristics for reducing the cost of solving sparse systems of linear equations. This algorithm was first proposed by Markowitz [26] in the context of reordering equations that arise in asymmetric linear programming problems, and it has since been the impetus for using graph algorithms and data structures in scientific computing [30, 31, 17, 16, 19]. This line of work culminated in the approximate minimum degree algorithm (AMD) of Amestoy, Davis, and Duff [2], which has long been a workhorse in the sparse linear algebra libraries for Julia, MATLAB, Mathematica, and SciPy. Formally, the minimum degree algorithm is a preprocessing step that permutes the rows and columns of a sparse symmetric positive-definite matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, before applying Cholesky decomposition, in an attempt to minimize the number of nonzeros in the Cholesky factor. Without a judicious reordering, the decomposition typically becomes dense with *fill-in* (i.e., additional nonzeros). The goal of the minimum degree algorithm is to efficiently compute a permutation matrix $\mathbf{P}$ such that the Cholesky factor $\mathbf{L}$ in the reordered matrix $\mathbf{PAP}^\intercal = \mathbf{LL}^\intercal$ is close to being optimally sparse. Finding an optimal permutation, however, is NP-complete [35], so practical approaches such as minimum degree orderings, the Cuthill–McKee algorithm [11], and nested dissection [15] are used instead. We direct the reader to "The Sparse Cholesky Challenge" in [21, Chapter 11.1] to further motivate efficient reordering algorithms and for a comprehensive survey.

The minimum degree algorithm takes advantage of a separation between the symbolic and numerical properties of a matrix. To see this, start by viewing the nonzero structure of $\mathbf{A}$ as the adjacency matrix of an undirected graph G with $m = \text{nnz}(\mathbf{A} - \text{diag}(\mathbf{A}))/2$ edges. Permuting the matrix by $\mathbf{PAP}^\intercal$ does not change the underlying graph. In each iteration, the algorithm (1) selects the vertex u with minimum degree, (2) adds edges between all pairs of neighbors of u to form a clique, and (3) deletes u from the graph. Through the lens of matrix decomposition, each greedy step corresponds to performing row and column permutations that minimize the number of off-diagonal nonzeros in the pivot row and column. A clique is induced on the neighbors of u in the subsequent graph because of the widely-used *no numerical cancellation assumption* (i.e., nonzero entries remain nonzero). Motivated by the success and ubiquity of reordering algorithms in sparse linear algebra packages, and also by recent developments in the hardness of computing minimum degree orderings of Fahrbach et al. [13], we investigate the fine-grained time complexity of the minimum degree ordering problem.

[*]School of Computer Science, University of Waterloo. Email: robert.cummings@uwaterloo.ca.

[†]Google Research. Email: fahrbach@google.com. Part of this work was done at Georgia Tech while supported by an NSF Graduate Research Fellowship under grant DGE-1650044.

[‡]Nuro. Email: afatehpuria@nuro.ai.

1.1 Results and Techniques

Our main results complement each other and nearly characterize the time complexity to compute a minimum degree ordering. Our first result is a new combinatorial algorithm that outputs an exact minimum degree ordering in $O(nm)$ time. This improves upon the best known result of $O(n^3)$ achieved by the naive algorithm. We maintain two different data structures for the fill graph to efficiently detect and avoid the redundant work encountered by the naive algorithm. Using careful amortized analysis, we prove the following theorems.

Theorem 1.1. *The* FastMinDegree *algorithm outputs an exact minimum degree ordering in $O(nm)$ time.*

Our analysis also allows us to derive output-sensitive bounds. The fill produced by the minimum degree heuristic is typically small in practice, so the performance of the algorithm is often faster than $O(nm)$. The algorithm also allows for further practical implementations, such as using hash table-based adjacency lists.

Theorem 1.2. *The* FastMinDegree *algorithm can be implemented to run in $O(\min(m\sqrt{m^+}, \Delta m^+) \log n)$ time and use $O(m^+)$ space, where m^+ is the number of unique fill edges and original edges that the algorithm encounters and Δ is the maximum degree of the original graph.*

Our second main result improves upon a recent conditional hardness theorem of $O(m^{4/3-\varepsilon})$ for computing exact minimum degree elimination orderings assuming the strong exponential time hypothesis [13].

Theorem 1.3. *Assuming the strong exponential time hypothesis, there does not exist an $O(m^{2-\varepsilon}\Delta^k)$ algorithm for computing a minimum degree elimination ordering, where Δ is the maximum degree of the original graph, for any $\varepsilon > 0$ and $k \geq 0$.*

This result is given in its full generality above, and it implies an answer to $O(nm^{1-\varepsilon})$-hardness conjecture posed in [13]. Specifically, we have the following matching lower bound for our main algorithm.

Corollary 1.1. *Assuming the strong exponential time hypothesis, there does not exist an $O(nm^{1-\varepsilon})$ time algorithm for computing a minimum degree elimination ordering, for any $\varepsilon > 0$.*

The hardness in Theorem 1.3 also rules out the existence of an $O(\sum_{v\in V} \deg(v)^2)$ time algorithm. We prove our hardness results by reducing the orthogonal vectors problem [32] to computing a minimum degree ordering of a special graph constructed using building blocks called *U-fillers*. One of our main contributions is a simple recursive algorithm for constructing U-filler graphs that satisfy the challenging sparsity and degree requirements necessary for the fine-grained reduction. In particular, these U-fillers correspond to pathological sparsity patterns, and hence adversarial linear systems, that cause any minimum degree algorithm to exhibit worst-case performance (i.e., to output Cholesky factors with $\tilde{\Omega}(n^2)$ nonzeros). These graphs are of independent interest and could be useful in lower bounds for other greedy algorithms.

1.2 Related Works

Computing an elimination ordering that minimizes fill-in is an NP-complete problem closely related to chordal graph completion [35]. Agrawal, Klein, and Ravi [1] gave the first approximation algorithm for the minimum fill-in problem, building on earlier heuristics by George [15] and by Lipton, Rose, and Tarjan [24]. Natanzon, Shamir, and Sharan [28] later developed the first algorithm to approximate the minimum possible fill-in to within a polynomial factor. There has since been a wealth of recent results on fixed-parameter tractable algorithms [23, 14, 7, 6] and the conditional hardness of minimizing fill-in [34, 5, 8, 4].

Due to this computational complexity, we rely on the practicality and efficiency of greedy heuristics. In particular, the multiple minimum degree algorithm (MMD) of Liu [25] and the approximate minimum degree algorithm (AMD) of Amestoy, Davis, and Duff [2] have been the mainstays for solving sparse linear systems of equations. These algorithms, however, have some drawbacks. MMD eliminates a maximal independent set of minimum degree vertices in each step, but it runs in $O(n^2m)$ time and this is known to be tight [22]. On the other hand, AMD is a single elimination algorithm that runs in $O(nm)$ time, but achieves its speedup by using an easy-to-compute upper bound as a proxy to the true vertex degrees. While many variants of the minimum degree algorithm exist, an algorithm that computes an exact minimum degree ordering with time complexity better than $O(n^3)$ has never been proven. Therefore, our contributions are a significant step forward in the theory of minimum degree algorithms.

There have also been other major advancements in the theory of minimum degree algorithms recently. Fahrbach et al. [13] designed an algorithm that computes a $(1+\varepsilon)$-approximate greedy minimum degree elimination ordering in $O(m \log^5(n)\varepsilon^{-2})$ time. Although this result is a significant theoretical milestone, it is currently quite far from being practical. Ost, Schulz, and Strash [29] recently gave a comprehensive set of vertex elimination rules that are to be used before applying a greedy reordering algorithm and never degrade the quality of the output. The minimum degree heuristic has also appeared in algorithms for graph compression and coarsening [3, 9, 12].

2 Preliminaries

2.1 Fill Graphs and Minimum Degree Orderings

For an undirected, unweighted graph $G = (V, E)$, let $N(u) = \{v \in V : \{u, v\} \in E\}$ denote the neighborhood of vertex u and $\deg(u) = |N(u)|$ denote its degree. We overload the notation $N(U) = \bigcup_{u \in U} N(u)$ to be the neighborhood of a set of vertices. For two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, define their union to be $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. For a given set of vertices U, let K_U be the complete graph with vertex set U.

Now we introduce the idea of fill graphs. Our notation extends that of Gilbert, Ng and Peyton [20]. We use the shorthand $[n] = \{1, 2, \dots, n\}$ throughout the paper.

DEFINITION 2.1. *For any undirected graph $G = (V, E)$ and subset $U \subseteq V$, the* fill graph *$G^+_U = (V^+_U, E^+_U)$ is the graph resulting from eliminating the vertices in U. Its vertex set is $V^+_U = V \setminus U$, and an edge $\{u, v\} \in E^+_U$ if and only if there exists a path $(u, x_1, \dots, x_t, v)$ in G such that $x_i \in U$ for all $i \in [t]$.*

Characterizing fill-in by uv-paths through eliminated vertices allows us to compute the *fill degree* of a vertex in any partially eliminated state U without explicitly computing the eliminated matrix. For a fill graph G^+_U, we avoid double subscripts and use the analogous notation $\deg^+_U(v) = \deg_{G^+_U}(v)$ and $N^+_U(v) = N_{G^+_U}(v)$ to denote the degree and neighborhood of a vertex $v \in V^+_U$. Alternatively, we can use tools from linear algebra and view G^+_U as the nonzero structure of the Schur complement of the adjacency matrix $\mathbf{A}(G)/U$.

An elimination ordering $p = (v_1, v_2, \dots, v_n)$ naturally induces a sequence of fill graphs $(G^+_0, G^+_1, \dots, G^+_n)$, where $G^+_0 = G$ and G^+_n is the empty graph. Let $\deg^+_i(v)$ and $N^+_i(v)$ denote the degree and neighborhood of vertex $v \in V^+_i$ in the i-th fill graph of this sequence. This allows us to define a minimum degree elimination ordering.

DEFINITION 2.2. *A* minimum degree elimination ordering *is a permutation of the vertices $(v_1, v_2, \dots, v_n)$ such that $v_i \in \arg\min_{v \in V^+_{i-1}} \deg^+_{i-1}(v)$ for all $i \in [n]$.*

2.2 SETH-Hardness for Computing Minimum Degree Orderings

Our lower bound for the time complexity of computing a minimum degree elimination ordering is based on the *strong exponential time hypothesis* (SETH), which asserts that for every $\varepsilon > 0$, there exists an integer k such that k-SAT cannot be solved in $O(2^{(1-\varepsilon)n})$ time. SETH has been tremendously useful in establishing tight conditional lower bounds for a diverse set of problems [33]. Many of these results rely on a fine-grained reduction to the *orthogonal vectors* problem and make use of the following theorem.

THEOREM 2.1. ([32]) *Assuming SETH, for any $\varepsilon > 0$, there does not exist an $O(n^{2-\varepsilon})$ time algorithm that takes n binary vectors with $\Theta(\log^2 n)$ bits and decides if there is an orthogonal pair.*

3 A Fast Minimum Degree Algorithm

We present a new combinatorial algorithm called FASTMINDEGREE for computing minimum degree orderings. Its key feature is that it maintains the fill graph using an implicit representation of fill-in together with an explicit graph representation. This combination allows us to reduce the number of redundant edge insertions to the fill graph. Using a specialized amortized analysis, we prove the following theorems.

THEOREM 3.1. *The* FASTMINDEGREE *algorithm outputs an exact minimum degree ordering in $O(nm)$ time.*

THEOREM 3.2. *The* FASTMINDEGREE *algorithm can be implemented to run in $O(\min(m\sqrt{m^+}, \Delta m^+) \log n)$ time and use $O(m^+)$ space, where m^+ is the number of unique fill edges and original edges that the algorithm encounters and Δ is the maximum degree of the original graph.*

3.1 The Algorithm

We begin by describing an alternative approach for representing the fill graphs G^+_i as vertices are eliminated. This *hypergraph representation* stores *hyperedges* $U_1, U_2, \dots, U_k \subseteq V^+_i$ such that $G^+_i = K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_k}$ at the end of each iteration. Variants of this have frequently been used in previous literature on the min-degree algorithm. Our presentation closely follows that of George and Liu [19], albeit with different terminology.

Given a graph G, we construct the initial hypergraph representation consisting of all hyperedges $\{u, v\}$, for every edge $\{u, v\} \in E(G)$. Next, we consider how to update the hypergraph representation as vertices are eliminated. Suppose we wish to eliminate a vertex v. Let $U_1, U_2, \dots, U_t$ be precisely the hyperedges that contain v. Construct $W = (U_1 \cup U_2 \cup \cdots \cup U_t) \setminus \{v\}$. Let G' be the graph represented by the current hypergraph representation. Then W is precisely the neighborhood of v in G'. It follows that the fill graph obtained by eliminating v is represented by removing the hyperedges $U_1, U_2, \dots, U_t$ and adding the hyperedge W.

Our algorithm to find a minimum degree ordering can be summarized as maintaining both a hypergraph representation and an adjacency matrix of the fill graph. The adjacency matrix is used to efficiently compute the minimum degree vertex in each step, and the hypergraph representation is used to reduce the number of redundant updates to the adjacency matrix.

When eliminating the vertex v in the current fill graph G_i^+, we find the hyperedges $U_1, U_2, \dots, U_t \subseteq V_i^+$ containing v, as described above. We also compute $W = (U_1 \cup U_2 \cup \cdots \cup U_t) \setminus \{v\} = N_i^+(v)$. We must add edges to the fill graph so that it contains the clique K_W. Therefore, it is enough for our algorithm to try adding the edges in K_W that are not in $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_t}$, since these are already guaranteed to be in G_i^+.

Below we give high-level pseudocode to describe this algorithm. Although the implementation details of the algorithm are very important to its efficiency, we defer these discussions to Section 3.2.

Algorithm 1 A fast minimum degree algorithm for producing exact elimination orderings.

```
1: function FastMinDegree(adjacency list for undirected graph $G = (V, E)$ with $|V| = n$)
2:     Initialize adjacency structure fill_graph to $G$
3:     for each edge $\{u, v\} \in E$ do
4:         Add $\{u, v\}$ to the list of hyperedges
5:     Mark all vertices as active
6:     Initialize array elimination_ordering of size $n$
7:     for $i = 1$ to $n$ do
8:         Let $a \leftarrow$ active vertex with minimum degree in fill_graph
9:         Deactivate $a$
10:        Set elimination_ordering[$i$] $\leftarrow a$
11:        Initialize $W \leftarrow \emptyset$
12:        for each hyperedge $U$ containing $a$ do
13:            Remove $U$ from the list of hyperedges
14:            Set $U \leftarrow U \setminus \{a\}$
15:            Let $X \leftarrow W \setminus U$ and $Y \leftarrow U \setminus W$
16:            for each pair $(x, y)$ in $X \times Y$ do
17:                Add edge $\{x, y\}$ to fill_graph if not present
18:            for each vertex $b \in Y$ do
19:                Remove edge $\{a, b\}$ from fill_graph
20:            Update $W \leftarrow W \cup Y$
21:        Add $W$ to the list of hyperedges
22:    return elimination_ordering
```

We claim that the algorithm is correct and maintains the desired state at the end of each iteration.

Lemma 3.1. FastMinDegree *produces an exact minimum degree ordering* `elimination_ordering`*. Furthermore, suppose $(G_0^+, G_1^+, \dots, G_n^+)$ is the sequence of fill graphs induced by* `elimination_ordering`*. Then, at the end of each iteration $i \in [n]$ of the* FastMinDegree *algorithm:*

1. *The state of* `fill_graph` *corresponds to the fill graph G_i^+.*
2. *The list of hyperedges $U_1, U_2, \dots, U_k$ satisfies $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_k} = G_i^+$ (i.e., the hypergraph representation is maintained).*

Proof. We proceed by induction and show that these invariants hold at the end of each iteration. Before any vertices are eliminated (line 6) both properties are true since $G_0^+ = G$. Assume the claim as the induction hypothesis and suppose that the algorithm begins iteration $i \in [n]$. Clearly FastMinDegree selects a minimum degree vertex in G_{i-1}^+ and updates `elimination_ordering` correctly.

Now we consider what happens when vertex a is eliminated. Let $U_1, U_2, \dots, U_t \subseteq V$ be all the hyperedges containing a at the start of iteration i. When the algorithm reaches line 21, we have $W = (U_1 \cup U_2 \cup \cdots \cup U_t) \setminus \{a\}$. All hyperedges $U_1, U_2, \dots, U_t$ are removed from the hypergraph representation by the end of this iteration, and W is added in their place. Since $W = N_{i-1}^+(a)$ by the induction hypothesis, it follows that the new list of hyperedges at the end of iteration i corresponds to G_i^+ and satisfies the second property.

Ensuring that the state of `fill_graph` is updated correctly requires a little more work. By the induction hypothesis, `fill_graph` is equal to G_{i-1}^+ at the beginning of iteration i. The hypergraph representation also corresponds to G_{i-1}^+ at the start of this iteration, so it follows that for each hyperedge U that contains a, the edges in K_U are present in `fill_graph`. Lines 9–18 attempt to insert edges from K_W into `fill_graph` that may be missing. In particular, clique K_W is constructed one hyperedge at a time to reduce redundant work. When processing the j-th hyperedge U_j, we have $X = (U_1 \cup U_2 \cup \dots U_{j-1}) \setminus U_j$ and $Y = U_j \setminus (U_1 \cup U_2 \cup \cdots \cup U_{j-1})$ on line 15. Only edges $\{x, y\} \in X \times Y$ spanning the symmetric difference can be missing from `fill_graph`, so at the end of iteration i, all edges in K_W have been considered. Finally, all edges adjacent to a in G_{i-1}^+ are removed in lines 18–19. Therefore, it follows that the state of `fill_graph` corresponds to G_i^+ at the end of the iteration. This proves the first property and completes the proof by induction. ◻

3.2 Complexity Analysis

To thoroughly investigate the time complexity of minimum degree algorithms, we first relax any rigid space requirements [18, 22] (e.g., using $O(m)$ memory). This allows us to more conveniently present an algorithm whose running time matches the SETH-based lower bound in Corollary 1.1.

Our goal is to bound the running time of FAST-MINDEGREE not just by $O(nm)$, but by more accurate bounds in terms of the total fill produced by the returned minimum degree ordering. Since the fill produced by the minimum degree heuristic is typically small in practice, our analysis shows that the performance of the algorithm will often be better than $O(nm)$. Below are the main quantities used to describe the total fill.

DEFINITION 3.1. *Let $E^+ = \bigcup_{i=0}^n E_i^+$ be the set of all edges in the fill graph at some iteration. Let $m^+ = |E^+|$.*

We have that E^+ is precisely the edge set of the input graph, together with all edges inserted throughout the course of the algorithm. Clearly (V, E^+) is a simple graph, and so $m^+ \le n^2$. These edges are important because they correspond to nonzero entries in the corresponding Cholesky factor $\mathbf{L}$. Intuitively, m^+ is the size of the "output" of the reordering procedure, and is the quantity that the minimum degree heuristic is designed to minimize.

First, we claim that any adjacency structure for the evolving fill graph can easily be extended to handle fast minimum degree queries. This technique uses a bucket queue and is inspired by ideas of Matula and Beck in [27].

LEMMA 3.2. *The `fill_graph` adjacency structure can be augmented to support the selection of the minimum degree vertex, increasing the total running time by at most $O(m^+)$.*

Proof. We maintain the data structure of n doubly linked lists that are "buckets" corresponding to degrees. Each vertex in the graph has a node in exactly one bucket. Every time we update the degree of a vertex, we delete it from one bucket and add it to another. This operation can be performed in $O(1)$ time.

The total number of degree updates is also $O(m^+)$. For each entry of the adjacency matrix, there are at most two degree updates: one when the edge is added (line 17), and one when it is deleted (line 19). These operations are only applied to precisely the $O(m^+)$ entries corresponding to the edges in E^+.

Selecting the minimum degree vertex v is done by checking the buckets in order of increasing degree, until a nonempty bucket is found. The running time for the i-th step is linear in $\deg_i^+(v)$, which is at most the degree of v in the graph (V, E^+). So in total, the running time is $O(m^+)$. $\square$

The next lemma bounds the running time of the algorithm by the total fill-in, the number of edge insertion attempts, and the cost of edge updates. The proof uses a simple credit analysis to show that computing all of the sets X, Y, and W over the course of the algorithm is not too expensive.

LEMMA 3.3. *Let k be the total number of edge insertion attempts, and let c be the cost of querying, inserting, or deleting an edge. Then the algorithm can be implemented to run in $O(c(m^+ + k))$ time.*

Proof. Let $(v_1, v_2, \dots, v_n)$ be the ordering found by the algorithm. The number of edge removals performed in iteration i is $\deg_{i-1}^+(v_i)$. Summed across all iterations, the total number of edge removals is $O(m^+)$. The other queries and insertions to the adjacency structure happen at most k times. Therefore, the total running time of adjacency structure operations is $O(c(m^+ + k))$.

We claim that the rest of the algorithm can be implemented to run in a total of $O(m^+ + k)$ time. The first step of this is an amortized analysis that assigns $|U|$ credits to each hyperedge U when it is created, so that when the hyperedge U is removed, the algorithm can afford an extra running time of $O(|U|)$. The initial hypergraph requires a total of $O(m)$ credits.

Consider iteration i of the algorithm, and let $U_1, U_2, \dots, U_t$ be the hyperedges containing v_i. The set W produced by the algorithm is, by the end of the iteration, equal to $(U_1 \cup U_2 \cup \cdots \cup U_t) \setminus \{v_i\} = N_i^+(v_i)$. So, the hyperedge added in iteration i has size $\deg_i^+(v_i)$, which summed over all iterations, requires $O(m^+)$ credits. This amortized analysis assigns $\sum_{j=1}^t |U_j|$ credits to the iteration where $U_1, U_2, \dots, U_t$ are the hyperedges being manipulated and removed.

Therefore, it suffices to show that iteration i runs in $O(\sum_{j=1}^t |U_j| + k_i)$ time, where k_i is the number of edge insertion attempts in iteration i. First, consider the loop over the hyperedges $U_1, U_2, \dots, U_t$. For $j \in [t]$, let W_j be the set W at the beginning of the iteration that considers U_j. Most operations in the loop clearly run in $O(|U_j|)$ time.

The exception is the computation of $X_j = W_j \setminus U_j$ and $Y_j = U_j \setminus W_j$. In general, finding $A \setminus B$ when A and B have values in $[n]$ is done by having a global array of size n modified to represent the contents of B. Then $A \setminus B$ can be found in $O(|A|)$ time. This also requires $O(|B|)$ time to modify and reset the relevant array entries. Our implementation maintains such an array that represents W as it changes over all t iterations, in a total of $|N_i^+(v_i)|$ time. This allows Y_j to be computed in $O(|U_i|)$ time.

If Y_j is empty, our algorithm can end iteration j of the inner loop early. Otherwise, the number of edge insertion attempts in this iteration is at least $|X_j \times Y_j| \ge |X_j|$. In this case, the set X_j is found in $O(|W_j| + |U_j|)$ time. Note that

$$\begin{aligned} |W_j| + |U_j| &\leq (|W_j \setminus U_j| + |U_j|) + |U_j| \\ &= O(|X_j| + |U_j|). \end{aligned}$$

So, the running time of these operations summed over all iterations $j \in [t]$ is $O(\sum_{j=1}^{t} |U_j| + k_i)$.

Finally, we consider the hypergraph data structure used to efficiently determine the hyperedges that contain a vertex a, and how this data structure is updated throughout the algorithm. In the implementation, each hyperedge ever created has a marker that determines whether it is valid or invalid (i.e., whether the hyperedge has since been removed). There is also a array of lists that determines, for each vertex a, the list of all hyperedges (valid or invalid) that contain it. This data structure is efficient to use and maintain. When a hyperedge U is inserted, we iterate through all $b \in U$ and add a pointer to U to the list of vertex b. $\square$

The bottleneck of the algorithm is the total number of attempted edge insertions across all vertex pairs $(x, y) \in X \times Y$. A trivial bound on the total running time of this step is $O(n^3)$. Improving on this requires a careful amortized analysis. Note that the algorithm may attempt to insert an edge between the same pair of vertices multiple times. Our approach is to investigate and bound precisely how many times an edge in E^+ is attempted to be inserted. The lemma below is the key technical result in the analysis of the algorithm.

Lemma 3.4. *The number of edge insertion attempts is at most* $\sum_{\{u,v\} \in E^+} \min(\deg(u), \deg(v))$, *where* $\deg(v)$ *is the degree of* v *in the original graph.*

Proof. Each $\{u, v\}$ not in E^+ is never inserted by the algorithm. So, it suffices to show that for all $\{u, v\} \in E^+$, the insertion of $\{u, v\}$ is attempted at most $\min(\deg(u), \deg(v))$ times. Suppose $\{u, v\} \in E^+$, and assume without loss of generality that $\deg(u) = \min(\deg(u), \deg(v))$. Let $f(u, v)$ be the number of hyperedges that contain u but not v. The quantity changes over time as the algorithm progresses.

First, we claim that until u or v is eliminated, $f(u, v)$ only decreases over time. This is because the only operations done to hyperedges is to merge them and to delete eliminated vertices. Merging hyperedges does not increase the number that contain u but not v, so until u or v is eliminated, $f(u, v)$ cannot increase.

Next, we claim that for each time the algorithm attempts to insert $\{u, v\}$, $f(u, v)$ strictly decreases. This is because an insertion attempt only happens when a set containing u but not v is merged with a set that contains v. So, after the merge operation, $f(u, v)$ has decreased by at least one.

Finally, note that when all of the hyperedges are first initialized, $f(u, v)$ is at most $\deg(u)$. All insertion attempts for $\{u, v\}$ must happen before u or v is eliminated, and during this time, $f(u, v) \geq 0$. Thus, there are at most $\deg(u) = \min(\deg(u), \deg(v))$ insertion attempts for the edge $\{u, v\} \in E^+$. $\square$

Corollary 3.1. *There are* $O(\Delta m^+)$ *edge insertion attempts, where* Δ *is the maximum degree of the input graph.*

We note that the approximate minimum degree algorithm (AMD) of Amestoy, Davis, and Duff [2] also emits a tighter time complexity of $O(m^+)$ on bounded-degree graphs [22]. This is one of the key reasons why it is exceptionally useful for reordering linear systems of equations on grid graphs.

Now we present a useful folklore result that allows us to make the sum over minimum degrees expression in Lemma 3.4 more comprehensible. This inequality leverages the existence of a structured edge orientation of its input graph and holds for any nonnegative assignment to the vertices, including the degree function.

Fact 3.1. *Let* $G = (V, E)$ *be a simple graph and* $m = |E|$. *For any vertex function* $f : V \to \mathbb{R}_{\geq 0}$, *we have*

$$\sum_{\{u,v\} \in E} \min(f(u), f(v)) \leq \sqrt{2m} \sum_{v \in V} f(v).$$

Proof. We first claim there is an orientation of E such that each vertex has out-degree at most $\sqrt{2m}$. Orient the edge $\{u, v\}$ such that if $u \to v$, then $\deg(u) \leq \deg(v)$. Let $\deg_{\text{out}}(u)$ denote the out-degree of u. Then

$$2m \geq \sum_{v \in N_{\text{out}}(u)} \deg(v) \geq \sum_{v \in N_{\text{out}}(u)} \deg(u) \geq \deg_{\text{out}}(u)^2.$$

Therefore, every vertex has out-degree at most $\sqrt{2m}$. The result then follows from the inequality

$$\begin{aligned} \sum_{\{u,v\} \in E} \min(f(u), f(v)) &= \sum_{u \in V} \sum_{v \in N_{\text{out}}(u)} \min(f(u), f(v)) \\ &\leq \sum_{u \in V} \sum_{v \in N_{\text{out}}(u)} f(u) \\ &\leq \sqrt{2m} \sum_{u \in V} f(u), \end{aligned}$$

which completes the proof. $\square$

Corollary 3.2. *The number of edge insertion attempts is* $O(m\sqrt{m^+})$. *In particular, it is at most* $O(nm)$.

Proof. First, we apply Fact 3.1 to the graph (V, E^+) using the degree function of the input graph $G = (V, E)$. The $O(m\sqrt{m^+})$ bound then follows from Lemma 3.4 and the fact that $\sum_{v \in V} \deg(v) = 2m$. The $O(nm)$ bound is a consequence of the inequality $m^+ \leq n^2$. ∎

We are now prepared to prove our main theorems about the FastMinDegree algorithm. Our first result states that the algorithm runs in $O(nm)$ time, matching the conditional hardness result in Corollary 1.1. Our second result is an output-sensitive bound on the running time that demonstrates the nuances in the speed of the algorithm across various inputs. The two proofs are analogous to each other, but use different adjacency structures for the fill graph in order to achieve a space-time tradeoff that is beneficial in practice.

Proof of Theorem 1.1. We use an adjacency matrix to represent `fill_graph`. Thus, initializing `fill_graph` requires $O(n^2)$ time and space. Note that we may assume G is connected, so $m \geq n - 1$. Using Lemma 3.3, the algorithm runs in $O(m^+ + k)$ time since adjacency matrices support constant-time edge queries, insertions, and deletions. Here k is the total number of edge insertion attempts. By Corollary 3.2, we have $k = O(nm)$. Therefore, the overall running time is bounded by $O(n^2 + m^+ + k) = O(n^2 + n^2 + nm) = O(nm)$, as desired. ∎

Proof of Theorem 1.2. We represent `fill_graph` as an adjacency list where the neighborhood of every node is stored in a balanced binary search tree [10, Chapter 13]. Each edge query and update costs $O(\log n)$. Initializing G requires $O(m \log n)$ time and $O(m)$ space. The algorithm then runs in $O((m^+ + k) \log n)$ time by Lemma 3.3, where k is the number of edge insertion attempts. The output-sensitive bounds in Corollary 3.1 and Corollary 3.2 imply that $k = O(\min(m\sqrt{m^+}, \Delta m^+))$. Since we have $m^+ = \sqrt{m^+}\sqrt{m^+} \leq n\sqrt{m^+} = O(m\sqrt{m^+})$, the time complexity follows. Further, the total space used is $O(m^+)$ because of the binary search trees. ∎

4 Improved SETH-Based Hardness

Now we affirmatively answer a conjecture of Fahrbach et al. in [13] about the conditional hardness of finding a minimum degree ordering. In particular, we prove the following stronger result.

Theorem 4.1. *Assuming the strong exponential time hypothesis, there does not exist an $O(m^{2-\varepsilon}\Delta^k)$ algorithm for computing a minimum degree elimination ordering, where Δ is the maximum degree of the original graph, for any $\varepsilon > 0$ and $k \geq 0$.*

The previous best SETH-based lower bound in [13] ruled out the existence of a nearly linear time algorithm for computing an exact minimum degree ordering by showing that a $O(m^{4/3-\varepsilon})$ time algorithm could be used to solve any instance of the orthogonal vectors problem in subquadratic time, for any $\varepsilon > 0$. Our approach has several similarities to that of Fahrbach et. al, and a consequence of our main hardness result gives a nearly matching lower bound for the running time of the FastMinDegree algorithm.

Corollary 1.1. *Assuming the strong exponential time hypothesis, there does not exist an $O(nm^{1-\varepsilon})$ time algorithm for computing a minimum degree elimination ordering, for any $\varepsilon > 0$.*

The key to our reduction is a recursive algorithm for constructing a graph with $O(n \log n)$ vertices and edges such that after any minimum degree ordering eliminates all but n vertices, the resulting graph is K_n. Although this construction was originally motivated by connections to SETH-based hardness, it has several interesting standalone properties. In particular, it demonstrates a case where the minimum degree heuristic has extremely poor performance (i.e., it is an input of size $O(n \log n)$ that always results in $\Omega(n^2)$ fill edges).

4.1 Constructing Sparse Min-Degree U-Fillers

Our goal in this subsection is to construct sparse graphs that contain the vertex set U and have the additional property that by repeatedly eliminating a minimum degree vertex, the resulting fill graph is eventually K_U. We begin by defining several specific properties that are helpful for presenting our construction.

Definition 4.1. *A U-*filler *is a graph G with vertex set $U \cup W$, where $U \cap W = \emptyset$, such that after eliminating all of the vertices in W, the resulting fill graph is $G_W^+ = K_U$. We call W the set of* extra vertices.

Many of the graphs we construct have a subset of extra vertices similar to the one in this definition. When taking the union of graphs with extra vertices, we always assume the sets of extra vertices are disjoint.

A simple example of a U-filler is the star graph with vertices $U \cup \{w\}$, where w is the center vertex with degree $|U|$. Although stars are sparse, their maximum degree can be arbitrarily large. This is the main challenge in designing adversarial inputs for minimum degree algorithms. Note that K_U is itself a U-filler.

Definition 4.2. *A U-filler is* min-degree *if after eliminating any proper subset of extra vertices $X \subset W$, all of the minimum-degree vertices in G_X^+ are extra vertices.*

It immediately follows from Definition 4.2 that every minimum degree elimination ordering of a min-degree

U-filler eliminates all of the extra vertices in W before any of the vertices in U.

Another key property we use in our construction is the following notion of degree bound.

Definition 4.3. *A U-filler is d*-bounded *if after eliminating any subset of extra vertices $X \subseteq W$, all remaining extra vertices have degree at most d in G_X^+.*

The first graph construction we use is called a *U-comb*. We define a U-comb to consist of a path of $|U|$ extra vertices together with $|U|$ edges that form a matching between the path vertices and those in U.

Lemma 4.1. *A U-comb is a $|U|$-bounded U-filler.*

Proof. The extra vertices of the U-comb form a connected component, so eliminating them results in K_U. Now suppose some subset of extra vertices is eliminated, and let v be one of the remaining extra vertices. Consider the neighbors of v in the fill graph. Let $w \neq v$ be another extra vertex of the U-comb. If w has not been eliminated, v cannot be adjacent to w's neighbor in U. So for any of the $|U| - 1$ other extra vertices, v has at most one neighbor in the fill graph. Since v is adjacent to a single other vertex in U, it follows that the U-comb is $|U|$-bounded. □

There is also a straightforward way to combine U-combs to obtain d-bounded U-fillers for any $d \geq 2$, although the size of the solution depends on the ratio $|U|/d$.

Lemma 4.2. *Let $d \geq 2$ and suppose $|U|/d \leq c$. Then we can construct a d-bounded U-filler with $O(|U|c)$ edges and maximum degree $O(c)$.*

Proof. We partition U into $O(c)$ parts $U_1, U_2, \ldots, U_k$ each of size at most $d/2$. We then let G be the union of $(U_i \cup U_j)$-combs over all unordered pairs $\{i, j\} \in \binom{[k]}{2}$. The result of eliminating all extra vertices of G is the union of $K_{U_i \cup U_j}$, which is exactly K_U. Since $|U_i \cup U_j| \leq d/2 + d/2 = d$, the combs are all d-bounded by Lemma 4.1. Therefore, G is a d-bounded U-filler. We constructed G from $O(c^2)$ combs each with $O(d)$ edges, so G has $O(c^2 d) = O(|U|c)$ edges. The extra vertices of the combs have maximum degree 3, but each vertex in U has degree exactly $k - 1$, so the maximum degree of G is $O(c)$. □

We now use Lemma 4.2 to construct min-degree U-fillers. The main idea of our approach is to (1) recursively construct min-degree fillers for two halves of U, and (2) connect the halves using a new U-filler with $O(|U|)$ edges whose extra vertices are guaranteed to be eliminated before any vertices in U. Combining this idea with divide-and-conquer, we show that the solution is of size $O(|U| \log|U|)$. Before proceeding to the proof, we reiterate that all extra vertices introduced in this construction are unique.

Theorem 4.2. *We can construct a $(|U| - 3)$-bounded min-degree U-filler with $O(|U| \log|U|)$ vertices and edges, and maximum degree $O(\log|U|)$.*

Proof. We prove this theorem by describing a recursive divide-and-conquer algorithm. In the base case when $|U| \leq 7$, simply return K_U. Since K_U has no extra vertices, it is a $(|U| - 3)$-bounded min-degree U-filler.

Now suppose $|U| \geq 8$, and partition U into U_1 and U_2 with sizes $\lfloor |U|/2 \rfloor$ and $\lceil |U|/2 \rceil$, respectively. We then recursively apply the theorem to construct min-degree fillers G_1 and G_2 for U_1 and U_2. Finally, we apply Lemma 4.2 to construct a $(\lfloor |U|/2 \rfloor - 2)$-bounded U-filler G_3. (We can do this because $\lfloor |U|/2 \rfloor - 2 \geq 8/2 - 2 = 2$.) Then we return the union $G_1 \cup G_2 \cup G_3$.

Since $\lceil |U|/2 \rceil - 3 \leq \lfloor |U|/2 \rfloor - 2$ we have that G_1, G_2, and G_3 are $(\lfloor |U|/2 \rfloor - 2)$-bounded. Since G_3 is a U-filler, so is $G_1 \cup G_2 \cup G_3$. Therefore, $G_1 \cup G_2 \cup G_3$ is a $(\lfloor |U|/2 \rfloor - 2)$-bounded U-filler. Since $|U| \geq 8$, this implies in particular that $G_1 \cup G_2 \cup G_3$ is $(|U| - 3)$-bounded.

Suppose for contradiction that $G_1 \cup G_2 \cup G_3$ is not min-degree, i.e., after eliminating some proper subset of extra vertices, there is a vertex $u \in U$ of minimum degree. Let U_i be the part of U that contains u. Note that the degree of u in $G_1 \cup G_2 \cup G_3$ after elimination is at least that of u in G_i after eliminating the corresponding extra vertices of G_i. So for u to have minimum degree, the min-degree property of G_i implies all extra vertices of G_i are eliminated. Then, since G_i is a U_i-filler, the degree of u is at least $|U_i| - 1 \geq \lfloor |U|/2 \rfloor - 1$. But since $G_1 \cup G_2 \cup G_3$ is $(\lfloor |U|/2 \rfloor - 2)$-bounded, this implies that all extra vertices were eliminated, which is impossible. Therefore, $G_1 \cup G_2 \cup G_3$ is min-degree.

Finally, we claim that $G_1 \cup G_2 \cup G_3$ has $O(|U| \log|U|)$ edges and maximum degree $O(\log|U|)$. Since

$$\begin{aligned} |U|/(\lfloor |U|/2 \rfloor - 2) &\leq |U|/(|U|/2 - 3) \\ &= 2/(1 - 6/|U|) \\ &\leq 8, \end{aligned}$$

G_3 has $O(|U|)$ edges and maximum degree $O(1)$ by Lemma 4.2. Now consider the divide-and-conquer nature of the construction. The number of edges when $|U| = n$ follows a recurrence of the form $f(n) = 2f(n/2) + O(n)$, which has the solution $f(n) = O(n \log n)$. So, the resulting graph has $O(|U| \log|U|)$ vertices and edges. The degrees of all extra vertices are $O(1)$, and any vertex in U is adjacent to at most $O(\log|U|)$ fillers across all recursive levels. Therefore, the overall maximum degree is $O(\log|U|)$. □

4.2 Reduction from Orthogonal Vectors

We now use our min-degree U-filler construction to demonstrate the hardness of finding an exact minimum degree ordering, assuming SETH. The connection to SETH is through the following problem via a reduction from orthogonal vectors [32].

DEFINITION 4.4. *The* clique union problem *takes as input a set of vertices V with $|V| = n$ and subsets of the vertices $U_1, U_2, \ldots, U_d \subseteq V$, where $d = \Theta(\log^2 n)$, and asks whether $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d} = K_V$.*

LEMMA 4.3. *Assuming the strong exponential time hypothesis, for any $\varepsilon > 0$, there is no $O(n^{2-\varepsilon})$ time algorithm for the clique union problem.*

Proof. Let each set U_i correspond to the set of vectors with a nonzero entry in the i-th dimension. There is a pair of orthogonal vectors if and only if the union $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d}$ is not the complete graph K_V. Thus, the result follows from Theorem 2.1 ([32]). □

Our approach to prove Theorem 1.3 using Lemma 4.3 is outlined in Algorithm 2 below. It is essentially a reduction from the clique union problem to the problem of finding an exact minimum degree ordering. The graph G on which we call the minimum degree ordering subroutine is built as a union over U_i-fillers produced by Theorem 4.2. As we prove below, G has special properties that, given any minimum degree ordering, allow us to efficiently determine the answer to the clique union instance.

Algorithm 2 Decides if $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d} = K_V$.

```
1:  function CliqueUnion($V$ and $U_1, \ldots, U_d \subseteq V$)
2:      for $i = 1$ to $d$ do
3:          Let $G_i$ be a min-degree $U_i$-filler constructed
            using Theorem 4.2
4:      Let $G \leftarrow G_1 \cup G_2 \cup \cdots \cup G_d$
5:      Let $W$ be the set of extra vertices of $G$
6:      Set elimination_ordering $\leftarrow$ MinDegree($G$)
7:      for $i = 1$ to $|W|$ do
8:          if elimination_ordering[$i$] $\notin W$ then
9:              return false
10:     Set $v \leftarrow$ elimination_ordering[$|W| + 1$]
11:     Determine which vertices in $G$ are reachable from
        $v$ via paths whose internal vertices are in $W$
12:     Let $k$ be the number of vertices in $V$ that are
        reachable from $v$, including $v$
13:     return $k = |V|$
```

LEMMA 4.4. *If* MinDegree *returns a minimum degree ordering, then* CliqueUnion *correctly decides if $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d} = K_V$.*

Proof. First, consider the case where CliqueUnion terminates early by returning `false` on line 9. Then for some proper subset $X \subset W$, a minimum degree vertex v of G_X^+ is not in W. Suppose i is an index such that $v \in U_i$. By Theorem 4.2, G_i is a min-degree U_i-filler. By considering the fill graph of G_i after eliminating all vertices in $X \cap V(G_i)$, it follows that all the extra vertices of G_i have been eliminated. Therefore, the degree of v in G_X^+ equals that of v in $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d}$. But Theorem 4.2 guarantees that for all $i \in [d]$, G_i is $(|U_i|-3)$-bounded, and thus $(|V|-2)$-bounded. So since $X \neq W$, the degree of v in $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d}$ is at most $|V| - 2$. Therefore, the algorithm correctly decides that $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d} \neq K_V$.

Now assume that the algorithm does not terminate early. Then the vertex v that is chosen on line 10 is the minimum degree vertex of G_W^+. Since G_i is a U_i-filler for all $i \in [d]$, $G_W^+ = K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d}$. Recall that an edge $\{u, v\}$ is in G_W^+ if and only if there is a path from u to v in G with internal vertices in W. It follows that the value of k found by the algorithm is one plus the degree of v in G_W^+. Thus, $k = |V|$ if and only if $G^+ = K_V$, so the algorithm returns the correct decision. □

LEMMA 4.5. *If* MinDegree *runs in $O(m^{2-\varepsilon}\Delta^k)$ time (for some $\varepsilon > 0$ and $k \geq 0$) on input G with m edges and max degree Δ, then* CliqueUnion *runs in $O(n^{2-\varepsilon'} d^{k+2})$ time for some $\varepsilon' > 0$, where $n = |V|$.*

Proof. For all $i \in [d]$, the graph G_i is constructed in $O(n \log n)$ time, has $O(n \log n)$ vertices and edges, and has maximum degree $O(\log n)$ by Theorem 4.2. Therefore, G has $O(nd \log n)$ vertices and edges and maximum degree $O(d \log n)$. We can compute the union of two graphs of size $O(m)$ in $O(m \log m)$ time, so we can construct G in $O(nd \log^2 n)$ time. The next step of the algorithm is to run MinDegree(G), and the running time of this step is

$$\begin{aligned} O\left(m^{2-\varepsilon}\Delta^k\right) &= O\left((nd \log n)^{2-\varepsilon}(d \log n)^k\right) \\ &= O\left(n^{2-\varepsilon} d^{k+2} \log^{k+2} n\right) \\ &= O\left(n^{2-\varepsilon'} d^{k+2}\right), \end{aligned}$$

for some $\varepsilon' > 0$. To determine reachability at line 11 of the algorithm, it suffices to use a breadth-first search that runs in $O(|E(G)|) = O(nd \log n)$ time. Therefore, the algorithm runs in $O(n^{2-\varepsilon'} d^{k+2})$ time. □

We conclude with the proof of our improved conditional hardness result for computing exact minimum degree elimination orderings. The complementary lower bound in Corollary 1.1 immediately follows.

Proof of Theorem 1.3. Assume for contradiction that an $O(m^{2-\varepsilon}\Delta^k)$ time algorithm exists for computing a minimum degree ordering. By Lemma 4.4 and Lemma 4.5, we can obtain an $O(n^{2-\varepsilon'}d^{k+2})$ time algorithm for deciding if $K_{U_1} \cup K_{U_2} \cup \cdots \cup K_{U_d} = K_V$, for some $\varepsilon' > 0$. For instances where $d = \Theta(\log^2 n)$, this algorithm runs in time $O(n^{2-\varepsilon'} \log^{2(k+2)} n) = O(n^{2-\varepsilon''})$, for some $\varepsilon'' > 0$. However, this contradicts SETH by Lemma 4.3, so the result follows. □

5 Conclusion

We have presented a new combinatorial algorithm for computing an exact minimum degree ordering with an $O(nm)$ worst-case running time. This is the first algorithm that improves on the naive $O(n^3)$ algorithm. We achieve this result using a careful amortized analysis, which also leads to strong output-sensitive bounds for the algorithm.

We also show a matching conditional hardness of $O(m^{2-\varepsilon}\Delta^k)$, for any $\varepsilon > 0$ and $k \geq 0$, which affirmatively answers a conjecture in [13] and implies there are no minimum degree algorithms with running time $O(nm^{1-\varepsilon})$ or $O(\sum_{v \in V} \deg(v)^2)$, assuming SETH. Together with the $O(nm)$ algorithm, this nearly characterizes the time complexity for computing an exact minimum degree ordering. Extending our U-filler graph construction to achieve fine-grained hardness results for other elimination-based greedy algorithms is of independent interest and an exciting future direction of this work.

Acknowledgments

We thank Richard Peng for many helpful discussions about the ideas in this paper.

References

[1] Ajit Agrawal, Philip Klein, and Ramamurthy Ravi. Cutting down on fill using nested dissection: Provably good elimination orderings. In *Graph Theory and Sparse Matrix Computation*, pages 31–55. Springer, 1993.

[2] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[3] Cleve Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM Journal on Scientific Computing*, 16(6):1404–1411, 1995.

[4] David Bergman, Carlos H. Cardonha, Andre A. Cire, and Arvind U. Raghunathan. On the minimum chordal completion polytope. *Operations Research*, 67(2):532–547, 2019.

[5] Ivan Bliznets, Marek Cygan, Pawel Komosa, Lukáš Mach, and Michał Pilipczuk. Lower bounds for the parameterized complexity of minimum fill-in and other completion problems. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1132–1151. SIAM, 2016.

[6] Ivan Bliznets, Fedor V. Fomin, Marcin Pilipczuk, and Michał Pilipczuk. Subexponential parameterized algorithm for interval completion. *ACM Transactions on Algorithms (TALG)*, 14(3):1–62, 2018.

[7] Yixin Cao and Dániel Marx. Chordal editing is fixed-parameter tractable. *Algorithmica*, 75(1):118–137, 2016.

[8] Yixin Cao and RB Sandeep. Minimum fill-in: Inapproximability and almost tight lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 875–880. SIAM, 2017.

[9] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 219–228, 2009.

[10] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 2009.

[11] Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, pages 157–172. ACM, 1969.

[12] Matthew Fahrbach, Gramoz Goranci, Richard Peng, Sushant Sachdeva, and Chi Wang. Faster graph embeddings via coarsening. *arXiv preprint arXiv:2007.02817*, 2020.

[13] Matthew Fahrbach, Gary L. Miller, Richard Peng, Saurabh Sawlani, Junxing Wang, and Shen Chen Xu. Graph sketching against adaptive adversaries applied to the minimum degree algorithm. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 101–112. IEEE, 2018.

[14] Fedor V. Fomin and Yngve Villanger. Subexponential parameterized algorithm for minimum fill-in. *SIAM Journal on Computing*, 42(6):2197–2216, 2013.

[15] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[16] Alan George and Joseph W. Liu. *Computer Solution of Large Sparse Positive Definite*. Prentice Hall Professional Technical Reference, 1981.

[17] Alan George and Joseph W. H. Liu. A quotient graph model for symmetric factorization. In *Sparse Matrix Proceedings*, pages 154–175, 1979.

[18] Alan George and Joseph W. H. Liu. A minimal storage implementation of the minimum degree algorithm. *SIAM Journal on Numerical Analysis*, 17(2):282–299, 1980.

[19] Alan George and Joseph W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[20] John R. Gilbert, Esmond G. Ng, and Barry W. Peyton. An efficient algorithm to compute row and column counts for sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 15(4):1075–1091, 1994.

[21] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, fourth edition, 2013.

[22] Pinar Heggernes, S. C. Eisestat, Gary Kumfert, and Alex Pothen. The computational complexity of the minimum degree algorithm. Technical report, Institute for Computer Applications in Science and Engineering, 2001.

[23] Haim Kaplan, Ron Shamir, and Robert E. Tarjan. Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. *SIAM Journal on Computing*, 28(5):1906–1922, 1999.

[24] Richard J. Lipton, Donald J. Rose, and Robert E. Tarjan. Generalized nested dissection. *SIAM Journal on Numerical Analysis*, 16(2):346–358, 1979.

[25] Joseph W. H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software (TOMS)*, 11(2):141–153, 1985.

[26] Harry M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.

[27] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983.

[28] Assaf Natanzon, Ron Shamir, and Roded Sharan. A polynomial approximation algorithm for the minimum fill-in problem. *SIAM Journal on Computing*, 30(4):1067–1079, 2000.

[29] Wolfgang Ost, Christian Schulz, and Darren Strash. Engineering data reduction for nested dissection. *arXiv preprint arXiv:2004.11315*, 2020.

[30] Donald J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In *Graph Theory and Computing*, pages 183–217. Elsevier, 1972.

[31] Robert E. Tarjan. Graph theory and gaussian elimination. In *Sparse Matrix Computations*, pages 3–22. Elsevier, 1976.

[32] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

[33] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018.

[34] Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth and related problems. *Journal of Artificial Intelligence Research*, 49:569–600, 2014.

[35] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.