2018 IEEE 59th Annual Symposium on Foundations of Computer Science

Graph Sketching Against Adaptive Adversaries Applied to the Minimum Degree Algorithm

Matthew Fahrbach*, Gary L. Miller†, Richard Peng*, Saurabh Sawlani*. Junxing Wang† and Shen Chen Xu‡
*School of Computer Science, Georgia Institute of Technology, Atlanta, USA
Email: matthew.fahrbach@gatech.edu, rpeng@cc.gatech.edu, sawlani@gatech.edu
†School of Computer Science, Carnegie Mellon University. Pittsburgh, USA
Email: glmiller@cs.cmu.edu, junxingw@cs.cmu.edu
‡Facebook, Menlo Park, USA
Email: shenchex@cs.cmu.edu

***Abstract*—Motivated by the study of matrix elimination orderings in combinatorial scientific computing, we utilize graph sketching and local sampling to give a data structure that provides access to approximate fill degrees of a matrix undergoing elimination in polylogarithmic time per elimination and query. We then study the problem of using this data structure in the minimum degree algorithm, which is a widely-used heuristic for producing elimination orderings for sparse matrices by repeatedly eliminating the vertex with (approximate) minimum fill degree. This leads to a nearly-linear time algorithm for generating approximate greedy minimum degree orderings. Despite extensive studies of algorithms for elimination orderings in combinatorial scientific computing, our result is the first rigorous incorporation of randomized tools in this setting, as well as the first nearly-linear time algorithm for producing elimination orderings with provable approximation guarantees.**

While our sketching data structure readily works in the oblivious adversary model, by repeatedly querying and greedily updating itself, it enters the adaptive adversarial model where the underlying sketches become prone to failure due to dependency issues with their internal randomness. We show how to use an additional sampling procedure to circumvent this problem and to create an independent access sequence. Our technique for decorrelating interleaved queries and updates to this randomized data structure may be of independent interest.

I. Introduction

Randomization has played an increasingly fundamental role in the design of modern data structures. The current best algorithms for fully-dynamic graph connectivity [28], [29], shortest paths [1], graph spanners [4], maximal matchings [32], and the dimensionality-reductions of large matrices [24], [26] all critically rely on randomization. An increasing majority of these data structures operate under the *oblivious adversary model*, which assumes that updates are generated independently of the internal randomness used in the data structure. In contrast, many applications of data structures are *adaptive*—meaning that subsequent updates may depend on the output of previous queries. A classical example of this paradigm is the combination of greedy algorithms with data structures, including Dijkstra's algorithm for computing shortest paths and Kruskal's algorithm for finding minimum spanning trees. The limitations imposed by adaptive adversaries are beginning to receive attention in the dynamic connectivity [29] and spanner [5] literature, but even for these problems there remains a substantial gap between algorithms that work in the adaptive adversary model and those that work only against oblivious adversaries [25].

This document is an extended abstract which has neither full proofs nor our full set of results. Readers are strongly encouraged to read the full version of the paper [12] instead, which can be found at https://arxiv.org/abs/1804.04239.

Motivated by a practically important example of adaptive invocations to data structures for greedy algorithms, we study the minimum degree algorithm for sparse matrix factorization and linear system solving [10]. This heuristic for precomputing an efficient pivot ordering is ubiquitous in numerical linear algebra libraries that handle large sparse matrices, and relies on a graph-theoretic interpretation of Gaussian elimination. In particular, the variables and nonzeros in a linear system correspond to vertices and edges in a graph, respectively. When the variable associated with vertex u is eliminated, a clique is induced on the neighborhood of u, and then u is deleted from the graph. This heuristic repeatedly eliminates the vertex of minimum degree in this graph, which corresponds to the variable with the fewest nonzeros in its row and column.

Computing elimination orderings that minimize the number of additional nonzeros, known as *fill*, has been shown to be computationally hard [30], [34], even in parameterized settings [7], [23]. However, the practical performance of direct methods has greatly benefited from more efficient algorithms for analyzing elimination orderings [3], [9]. Tools such as elimination trees [17] can implicitly represent fill in time that is nearly-linear in the number of *original* nonzeros, which allows for efficient prediction and reorganization of future computation and, more importantly, memory bandwidth. In contrast to the abundance of algorithms built on examining elimination orderings via implicit representation [21], [31], surprisingly little attention has been given to producing elimination orderings implicitly. In the survey by Heggernes et al. [20], the authors give an $O(n^2 m)$ algorithm for computing a minimum degree ordering, which is more than the cost of Gaussian elimination itself and significantly more than the nearly-linear time algorithms for analyzing such orderings [17].

Main Results: We begin our study by combining implicit representations of fill with graph sketching. The nonzero entries of a partially eliminated matrix can be represented as the set of vertices reachable within two hops in a graph that undergoes edge contractions [17]. This allows us to incorporate ℓ_0-sketches [8], which were originally developed to estimate the cardinality of reachable sets of vertices in directed graphs. By augmenting ℓ_0-sketches with suitable data structures, we obtain the following result for dynamically maintaining fill structure.

Theorem 1. *Against an oblivious adversary, we can maintain $(1 \pm \epsilon)$-approximations to the degrees of the graph representation of a matrix undergoing elimination in $O(\log^3 n\epsilon^{-2})$ per operation.*

We also give an exact version of this data structure for cases where the minimum degree is always small (e.g., empirical performance of Gaussian elimination on grid graphs [6]). Ignoring issues of potential dependent randomness, the approximation guarantees of this data structure provide us with an ordering that we call an *approximate greedy minimum degree ordering*, where at each step a vertex whose degree is close to the minimum is pivoted. It is unclear if such an ordering approximates a true minimum degree ordering, but such guarantees are more quantifiable than previous heuristics for approximating minimum degree orderings [2], [20].

However, using this randomized data structure in a greedy manner exposes the severe limitations of data structures that only work in the oblivious adversary model. The updates (i.e. the vertices we eliminate) depend on the output to previous minimum-degree queries, and hence its own internal randomness. The main result in this paper is an algorithm that uses dynamic sketching, as well as an additional routine for estimating degrees via local sampling, to generate an approximate greedy minimum degree sequence in nearly-linear time against adaptive adversaries.

Theorem 2. *Given an $n \times n$ matrix A with nonzero graph structure G containing m nonzeros, we can produce a $(1 + \epsilon)$-approximate greedy minimum degree ordering in $O(m \log^5 n\epsilon^{-2})$ time.*

Techniques: Several components of our algorithm are highly tailored to the minimum degree algorithm. For example, our dynamic sketches and local degree estimation routine depend on the implicit representation of intermediate states of Gaussian elimination [17]. That said, our underlying randomized techniques (e.g., ℓ_0-sketches [8] and wedge sampling [11]) are new additions to combinatorial scientific computing.

The primary focus of this paper is modifying the guarantees in the oblivious adversary model from Theorem 1 to work within a greedy loop (i.e. an adaptive adversary) to give Theorem 2. However, we do not accomplish this by making the queries deterministic or worst-case as in [5], [28], [29]. Instead, we use an external randomized routine for estimating fill degrees to create a fixed sequence of updates. The randomness within the sketching data structure then becomes independent to the update sequence, but its internal state is still highly useful for determining which vertices could have approximate minimum degree. We then efficiently construct the update sequence using recent developments for randomized graph algorithms that use exponential random variables [27]. Our use of sketching can also be viewed as a pseudodeterminstic algorithm whose goal is to efficiently recover a particular sequence of vertices [13], [18]. We believe that both of these views are valuable to the study of randomness and for better understanding the relationship between oblivious and adaptive adversaries.

Organization: In Section II we formalize the implicit representation of fill and variants of minimum degree orderings. In Section III we give an overview of our results, along with a brief description of the algorithms and techniques we employ. The use of sketching and sampling to obtain our exact and approximate algorithms are given in Section IV and Section V, respectively. We also detail our derandomization routine in Section V, which is crucial for using our randomized data structure against an adaptive adversary. In Section VI we demonstrate how to estimate fill degrees via local sampling, and in Section VII we show how to maintain sketches as vertices are pivoted.

II. Preliminaries

A. Gaussian Elimination and Fill Graphs

Gaussian elimination is the process of repeatedly eliminating variables from a system of linear equations, while maintaining an equivalent system on the remaining variables. Algebraically, this involves taking an equation involving a target variable and subtracting (a scaled version of) this equation from all others involving the target variable. We assume throughout the paper that the systems are symmetric positive definite (SPD) and thus the diagonal will remain positive, allowing for any pivot order. This further implies that we can apply elimination operations to columns in order to isolate the target variable, resulting in the Schur complement.

A particularly interesting fact about Gaussian elimination is that the numerical Schur complement is unique irrespective of the pivoting order. Under the now standard assumption that nonzero elements do not cancel each other out [15], this commutative property also holds for the combinatorial nonzero structure. By interpreting the nonzero structure of a symmetric matrix A as an adjacency matrix for a graph G, we can define the change to the nonzero structure of A as a graph-theoretic operation on G analogous to the Schur complement.

Our notation extends that of Gilbert, Ng, and Peyton [17], who worked with known elimination orderings and treated the entire fill pattern (i.e. additional nonzeros entries) statically. Because we work with partially eliminated states, we will need to distinguish between the *eliminated* and *remaining* vertices in G. We implicitly address this by letting x and y denote eliminated vertices and by letting u, v, and w denote remaining vertices. The following definition of a fill graph allows us to determine the nonzero structure on the remaining variables of a partially eliminated system.

Definition 1. The *fill graph* $G^+ = (V^+, E^+)$ is a graph on the remaining vertices such that the edge $(u, v) \in E^+$ if u and v are connected by a (possibly empty) path of eliminated vertices.

This characterization of fill means that we can readily compute the *fill degree* of a vertex v, denoted by $\deg^+(v) = |N^+(v)|$, in a partially eliminated state without explicitly constructing the matrix. We can also iteratively form G^+ from the original graph G by repeatedly removing an eliminated vertex x along with its incident edges, and then adding edges between all of the neighbors of x to form a clique. This operation gives the nonzero structure of the Schur complement.

Lemma 1. *For any graph $G = (V, E)$ and vertex $v \in V$, given an elimination ordering S we can compute $\deg^+(v)$ at the step when v is eliminated in $O(m)$ time.*

This kind of path finding among eliminated vertices adds an additional layer of complexity to our data structures. To overcome this, we contract eliminated vertices into their connected components (with respect to their induced subgraph in G), which leads to the component graph.

Definition 2. We use $G^\circ = (V^\circ_{\text{comp}}, V^\circ_{\text{rem}}, E^\circ)$ to denote the *component graph*. The set of vertices in V°_{comp} is formed by contracting edges between eliminated vertices, and the set of vertices that have not been eliminated is V°_{rem}. The set of edges E° is implicitly given by the contractions.

Note that G° is quasi-bipartite, as the contraction rule implies there are no edges between vertices in V°_{comp}. It will be useful to refer to two different kinds of neighborhoods in a component graph. For any vertex v in G°, let $N^\circ_{\text{rem}}(v)$ be the set of neighbors of v are in V°_{rem}, and let $N^\circ_{\text{comp}}(v)$ denote the neighbors of v that are in V°_{comp}. We also use the notation $\deg^\circ_{\text{rem}}(v) = |N^\circ_{\text{rem}}(v)|$ and $\deg^\circ_{\text{comp}}(v) = |N^\circ_{\text{comp}}(v)|$.

B. Minimum Degree Orderings

The minimum degree algorithm is a greedy heuristic for reducing the cost of solving sparse linear systems that repeatedly eliminates the variable involved in the fewest number of equations [15]. Although there are many situations where this is suboptimal, it is remarkably effective and widely used in practice. For example, the approximate minimum degree algorithm (AMD) [2] is a heuristic for generating minimum degree orderings that plays an integral role in the sparse linear algebra packages in MATLAB, Mathematica, and Julia.

For any elimination ordering $(u_1, u_2, \ldots, u_n)$, we let G_i be the graph with vertices $u_1, u_2, \ldots, u_i$ marked as eliminated and $u_{i+1}, u_{i+2}, \ldots, u_n$ marked as remaining. We denote the corresponding sequence of fill graphs by $(G^+_0, G^+_1, \ldots, G^+_n)$, where $G^+_0 = G$ and G^+_n is the empty graph. Throughout the paper, we frequently use the notation $[n] = \{1, 2, \ldots, n\}$ when iterating over sets.

Definition 3. A *minimum degree ordering* is an elimination ordering such that for all $i \in [n]$, the vertex u_i has minimum fill degree in G^+_{i-1}. Concretely, this means $\deg^+_{i-1}(u_i) = \min_{v \in V^+_{i-1}} \deg^+_{i-1}(v)$.

The data structures we use for finding the vertices with minimum fill degree are randomized, so we need to be careful to not introduce dependencies between different steps of the algorithm when several vertices are of minimum degree. To avoid this problem, we require that the lexicographically-least vertex be eliminated in the event of a tie.

Our notion for approximating a minimum degree ordering is based on finding a vertex at each step whose degree is close to the minimum in G^+_t, which is the goal of the AMD algorithm. This decision process has no lookahead, and thus does not in any way approximate the minimum possible total fill incurred during Gaussian elimination, which is known to be NP-complete [34].

Definition 4. A $(1+\epsilon)$*-approximate greedy minimum degree ordering* is an elimination ordering such that at each step $i \in [n]$, we have $\deg^+_{i-1}(u_i) \le (1 + \epsilon) \min_{v \in V^+_{i-1}} \deg^+_{i-1}(v)$.

III. Overview

We discuss the main components of our algorithms in three parts: sketching fill graphs, dealing with adaptive adversaries using decorrelation, and local estimation of fill degrees. Lastly, we explain the implications of our results to the study of algorithms for computing elimination orderings.

A. Dynamically Sketching Fill Graphs

The core problem of estimating fill degrees can be viewed as estimating the cardinality of sets undergoing unions and deletion of elements. To see this connection, assume for simplicity that no edges exist between the remaining vertices in the component graph G°. Split each remaining vertex u into two vertices u_1 and u_2, and replace every edge (u, x) to a component vertex x by the directed edges (u_1, x) and (x, u_2). The fill degree of u is the number of remaining vertices v_2 reachable from u_1 (not including u_1). Cohen [8] developed a nearly-linear time size-estimation framework for reachability problems using sketching and ℓ_0-estimators. Adapting this framework to our setting for fill graphs leads to the following kind of ℓ_0-sketch data structure. We refer to the set $N(u) \cup \{u\}$ as the 1-*neighborhood* of u, and we call its cardinality $\deg(u) + 1$ the 1-*degree* of u.

Definition 5. A *1-neighborhood ℓ_0-sketch* of a graph G is constructed as follows:

1) Each vertex $u \in V$ independently generates a random key $R(u)$ uniformly from $[0, 1)$.
2) Then each vertex determines which of its neighbors (including itself) has the smallest key. We denote this by $\textsc{Minimizer}(u) \overset{\text{def}}{=} \arg\min_{v \in N(u) \cup \{u\}} R(v)$.

To give some intuition for how sketching is used to estimate cardinality, observe that choosing keys independently and uniformly at random essentially assigns a random vertex $N(u) \cup \{u\}$ to be $\textsc{Minimizer}(u)$. Therefore, the key value $R(\textsc{Minimizer}(u))$ is correlated with $\deg(u)+1$. This correlation is the cornerstone of sketching. If we construct $k = \Omega(\log n\epsilon^{-2})$ independent sketches, then by concentration we can use an order statistic of $R_i(\textsc{Minimizer}(u))$ over all k sketches to give an ϵ-approximation of $\deg(u)+1$ with high probability.

To maintain sketches of the fill graph as it undergoes vertex eliminations, we first need to implicitly maintain the component graph G° (Lemma 10). We demonstrate how to efficiently propagate key values in a sketch as vertices are pivoted in Section VII. For now, it is sufficient to know that each vertex in a sketch has an associated min-heap that it uses to report and update its minimizer. Because eliminating vertices leads to edge contractions in the component graph, there is an additional layer of intricacies that we need to resolve using amortized analysis.

Suppose v is the vertex eliminated as we transition from G_t° to G_{t+1}°. The sketch propagates this information to relevant vertices in the graph using a two-level notification mechanism. The neighbors of v are informed first, and then they notify their neighbors about the change, all the while updating the key values in their heaps. While this algorithm is relatively simple, bounding its running time is nontrivial and requires a careful amortized analysis to show that the bottleneck operation is the merging of component vertices. We demonstrate the working of this two-level notification algorithm for pivoting vertices along with its performance bounds in Section VII.

B. Correlation and Decorrelation

We now discuss how we use the randomized sketching data structure within a greedy algorithm. We start with a simple concrete example to illustrate a problem that an adaptive adversary can cause. Consider a data structure that uses sketching to estimate the cardinality of a subset $S \subseteq [n]$ under the insertion and deletion of elements. This data structure randomly generates a subset of keys $T \subseteq [n]$ such that $|T| = \Theta(\log n\epsilon^{-2})$, and it returns as its estimate the scaled intersection $n \cdot |S \cap T| / |T|$, which is guaranteed to be within an ϵn-additive error of the true value $|S|$ by Chernoff bounds, assuming that T is generated independently of S.

Clearly this cardinality-estimation algorithm works in the oblivious adversary model. However, an adaptive adversary can use answers to previous queries to infer the set of secret keys T in $O(n)$ updates and queries. Consider the following scheme in Figure 1 that returns $S = T$.

```
1) Initialize $S = [n]$.
2) For each $i = 1$ to $n$:
    a) Delete $i$ from $S$. If the estimated size of $S$
       changed, reinsert $i$ into $S$.
3) Return $S$.
```

Figure 1: An adaptive routine that amplifies the error of a cardinality-estimation scheme using a fixed sketch.

While the updates performed by a greedy algorithm are less extreme than this, in the setting where we maintain the cardinality of the smallest of k dynamic sets, having access to elements in the minimizer does allow for this kind of sketch deduction. Any accounting of correlation (in the standard sense) also allows for worst-case kinds of adaptive behavior, similar to the scheme above.

To remove potential correlation, we use an external routine that is analogous to the local degree-estimation algorithm used in the approximate minimum degree algorithm and runs in time close to the degree it estimates. In this simplified example above, suppose for each cardinality query that the data structure first regenerates T. Then the probability that i belongs to S is $\Theta(\log n\epsilon^2/n)$. Stepping through all $i \in [n]$, it follows that the expected number of deletions is $\Theta(\log n\epsilon^{-2})$, and hence S remains close to size n with high probability.

Reinjecting randomness is a standard method for decorrelating a data structure across steps. However, if we extend this example to the setting where we maintain the cardinality of k sets (similar to our minimum degree algorithm), then the previous idea requires that we reestimate the size of every set to determine the one with minimum cardinality. As a result, this approach is prohibitively expensive. However, these kinds of cardinality estimations are actually local—meaning that it is sufficient to instead work with a small and accurate subset of candidates sets. If we compute the set with minimum cardinality among the candidates using an external estimation scheme, then this decision is independent of the random choice of T in the sketching data structure, allowing us to use the sketching data structure to generate the list candidates.

Our algorithm for generating an approximate greedy minimum degree ordering relies on a similar external routine called EstimateFill1Degree(u, ϵ), which locally estimates the fill 1-degree of u at any step of the algorithm in time proportional to $\deg(u)$ in the original graph. We further describe this estimator in Section III-C and present the full sampling algorithm in Section VI. In Section V we show that to generate an approximate greedy minimum degree sequence, it is instead sufficient to pivot the remaining vertex

$$\arg\min_{u \in V^+} \left(1 - \frac{\epsilon \cdot \text{Exp}(1)}{\log n}\right) \cdot \text{EstimateFill1Degree}\,(u)$$

at each step, where $\text{Exp}(1)$ is drawn from an exponential distribution. We call this the *ϵ-decayed minimum* over all external estimates.

Analogous to our example of set cardinality estimation above, evaluating the degrees of every remaining vertex using EstimateFill1Degree at each step is expensive and leads to a total cost of $\Omega(nm)$. However, we can use the sketching data structure and the following observations about the perturbation coefficient to sample a small number of candidate vertices that contains the ϵ-decayed minimum.

- For a set of vertices whose degrees are within $1 \pm \epsilon / \log n$ of each other, it suffices to consider $O(1)$ of them by generating the highest order statistics of exponential random variables in decreasing order.
- By the memoryless property of the exponential distribution, if we call EstimateFill1Degree, then with constant probability it will be for the vertex we pivot. Thus, we can charge the cost of these evaluations to the original edges and retain a nearly-linear running time.

Invoking EstimateFill1Degree only on the candidate vertices allows us to efficiently find the ϵ-decayed minimizer in each step, which leads to the nearly-linear runtime as stated in Theorem 2. The key idea is that any dependence on the ℓ_0-sketches stops after the candidates are generated, since their degrees only depend on the randomness of an external cardinality-estimation routine.

C. Local Estimation of Fill Degrees

A critical part of the approximate min-degree algorithm is the EstimateFill1Degree function, which estimates the fill 1-degree of a vertex $u \in V^+$ using fresh randomness and $O(\deg(u) \log^2 n\epsilon^{-2})$ oracle queries to the component graph G°. At the beginning of [12, Section VI] we show how to construct a $(0, 1)$-matrix A where each row corresponds to a remaining neighborhood of a component neighbor of u. The number of nonzero columns in A is equal to $\deg^+(u)$. Using only the following matrix operations (corresponding to component graph oracle queries), we analyze the more general problem of counting the number of nonzero columns in a matrix.

- RowSize(A, i): Returns the number of nonzero elements in row i of A.
- SampleFromRow(A, i): Returns a random column index j from the nonzero entries of row i of A.
- QueryValue(A, i, j): Returns the value of $A(i, j)$.

D. Significance to Combinatorial Scientific Computing

Despite the unlikelihood of theoretical gains for solving linear systems by improved direct methods for sparse Gaussian elimination, we believe our study could influence combinatorial scientific computing in several ways. First, we provide evidence for the nonexistence of nearly-linear time algorithms for finding exact minimum degree orderings by proving conditional hardness results. Our reduction uses the observation that determining if a graph can be covered by a particular union of cliques (or equivalently, that the fill graph is a clique after eliminating certain vertices) is equivalent to the orthogonal vectors problem [33]. Assuming the strong exponential time hypothesis, this leads to a conditional hardness of $\Omega(m^{4/3-\theta})$ for computing a minimum degree ordering. However, we believe that this result is suboptimal and that a more careful construction could lead to $\Omega(nm^{1-\theta})$-hardness.

On the other hand, advances in minimum degree algorithms cannot be justified in practice solely by worst-case asymptotic arguments. In general, nested dissection orderings are asymptotically superior in quality to minimum degree orderings [22]. Furthermore, methods based on Krylov spaces, multiscale analysis, and iterative methods [14], [19] are becoming increasingly popular as they continue to improve state-of-the-art solvers for large sparse systems. Such advancements are also starting to be reflected in theoretical works. As a result, from both a theoretical and practical perspective, we believe that the most interesting question related to minimum degree algorithms is whether or not such sequences lead to computational gains for problems of moderate size.

IV. Sketching to Compute Degrees

In this section we show that if an ℓ_0-sketch can efficiently be maintained for a dynamic graph, then we can use the same set of sketches at each step to find the vertex with minimum fill degree and eliminate it. We explore the dynamic ℓ_0-sketch data structure for efficiently propagating key values under pivots in detail in Section VII (and for now we interface it via Theorem 5). This technique leads to improved algorithms for computing the minimum degree ordering of a graph, which we analyze in two different settings.

First, we consider the case where the minimum degree at each step is bounded. In this case we choose a fixed number of ℓ_0-sketches and keep track of every minimizer of a vertex over all of the sketch copies. Note that we can always use n as an upper bound on the minimum fill degree.

Theorem 3. *There is an algorithm* DeltaCappedMinDegree *that, when given a graph with a lexicographically-first min-degree ordering whose minimum degree is always* *bounded by Δ, outputs this ordering with high probability in expected time $O(m\Delta \log^3 n)$ and uses space $O(m\Delta \log n)$.*

Second, we modify the algorithm to compute an approximate minimum degree vertex at each step. By maintaining $\Theta(\log n\epsilon^{-2})$ copies of the ℓ_0-sketch data structure, we are able to accurately approximate the 1-degree of a vertex using the $(1-1/e)$-thorder statistic of the key values of its minimizers. We use the following degree data structure to abstract this idea, which when given an elimination ordering directly leads to a nearly-linear time algorithm.

Theorem 4. *There is a data structure* `ApproxDegreeDS` *that supports the following methods:*

- ApproxDegreeDS_Pivot(u), *which pivots a remaining vertex u.*
- ApproxDegreeDS_Report(), *which provides balanced binary search tree (BST) containers $V_1, V_2, \ldots, V_B$ such that all the vertices in the bucket V_i have 1-degree in the range $[(1+\epsilon)^{i-2}, (1+\epsilon)^{i+2}]$.*

The memory usage of this data structure is $O(m \log n\epsilon^{-2})$. Furthermore, if the pivots are picked independently from the randomness used in this data structure (i.e., we work under the oblivious adversary model) then:

- *The cost of all calls to* ApproxDegreeDS_Pivot *is bounded by $O(m \log^3 n\epsilon^{-2})$.*
- *The cost of each call to* ApproxDegreeDS_Report *is bounded by $O(\log^2 n\epsilon^{-1})$.*

A. Computing the Exact Minimum Degree Ordering

We first consider the case where the minimum degree in each of the fill graphs G_t^+ is at most Δ. In this case, we maintain $k = O(\Delta \log n)$ copies of the ℓ_0-sketch data structure. By a coupon collector argument, any vertex with degree at most Δ contains all of its neighbors in its list of minimizers with high probability. This implies that for each $t \in [n]$, we can obtain the exact minimum degree in G_t^+ with high probability. Figure 2 briefly describes the data structures we will maintain for this version of the algorithm.

```
Global Variables: graph $G$, degree cap $\Delta$.
  1) $k$, the number of sketches set to $10(\Delta+1)\lceil \log n \rceil$.
  2) $k$ independent $\ell_0$-sketch data structures
     dynamic_sketch[1], ..., dynamic_sketch[$k$].
  3) For each vertex $u$, a balanced binary search tree
     minimizers[$u$] that stores Minimizer$_i(u)$ across
     all $i \in [k]$ $\ell_0$-sketches.
  4) A balanced binary tree size_of_minimizers
     on all vertices $u$ with the key of $u$ set to the number
     of different elements in minimizers[$u$].
```

Figure 2: Global variables for the Δ-capped min-degree algorithm DeltaCappedMinDegree.

Note that if we can efficiently maintain the data structures in Figure 2, then querying the minimum element in `size_of_minimizers` returns the (lexicographically-least) vertex with minimum degree. Theorem 5 demonstrates that we can maintain the ℓ_0-sketch data structures efficiently.

```
DeltaCappedMinDegree($G, \Delta$)
Input: graph $G = (V, E)$, threshold $\Delta$.
Output: exact lexicographically-first min-degree ordering.
  1) For each step $t = 1$ to $n$:
       a) Set $u_t \leftarrow \min($size_of_minimizers$)$.
       b) DeltaCappedMinDegree_Pivot($u_t$).
  2) Return $(u_1, u_2, \ldots, u_n)$.
```

```
DeltaCappedMinDegree_Pivot($u$)
Input: vertex to be pivoted $u$.
Output: updated global state.
  1) For each sketch $i = 1$ to $k$:
       a) $(v_1, v_2, \ldots, v_\ell) \leftarrow$
          dynamic_sketch[$i$].PivotVertex($u$),
          the set of vertices in the $i$-th sketch whose
          minimizers changed after pivoting out $u$.
       b) For each $j = 1$ to $\ell$:
            i) Update the values corresponding to sketch $i$
               in minimizers[$v_j$].
           ii) Update size_of_minimizers entry for
               $v_j$ with the size of minimizers[$v_j$].
```

Figure 3: Pseudocode for the exact Δ-capped min-degree algorithm, which utilizes the global data structures for DeltaCappedMinDegree defined in Figure 2.

Theorem 5. *Given i.i.d. random variables $R(v)$ associated with each vertex $v \in V_t^+$, there is a data structure* `DynamicSketch` *that, for each vertex u, maintains the vertex with minimum $R(v)$ among itself and its neighbors in G_t^+. This data structure supports the following methods:*

- QueryMin(u), *which returns* Minimizer(u) *for a remaining vertex u in $O(1)$ time.*
- PivotVertex(u), *which pivots a remaining vertex u and returns the list of remaining vertices v whose value of* Minimizer(v) *changed immediately after this pivot.*

The memory usage of this data structure is $O(m)$. Moreover, for any choice of key values $R(v)$:

- *The total cost of all the pivots is $O(m \log^2 n)$.*
- *The total size of all lists returned by* PivotVertex *over all steps is $O(m \log n)$.*

This theorem relies on intermediate data structures described in Section VII, so we defer the proof until the end of that section. Note that this `DynamicSketch` data structure will be essential to all of our min-degree algorithms.

Now consider a sketch of G^+ and a vertex u with degree $\deg^+(u) \le \Delta$. By symmetry of the $R(v)$ values, each vertex in $N^+(u) \cup \{u\}$ is the minimizer of u with probability $1/(\deg^+(u)+1)$. Therefore, if we maintain $O(\Delta \log n)$

independent ℓ_0-sketches, we ensure that we have an accurate estimation of the minimum fill degree with high probability. The pseudocode for this routine is given in Figure 3. We formalize the probability guarantees in Lemma 2 and Lemma 3, which are a restatement of [8, Theorem 2.1].

Lemma 2. *For all vertices u such that $\deg^+(u) \leq 2\Delta$, we have* `size_of_minimizers`$[u] = \deg^+(u)+1$ *with high probability.*

Lemma 3. *For all vertices u with $\deg^+(u) > 2\Delta$, we have* `size_of_minimizers`$[u] > \Delta+1$ *with high probability.*

B. Computing an Approximate Minimum Degree

To avoid bounding the minimum fill degree over all steps, we modify the previous algorithm to obtain an approximate min-degree vertex at each step. We reduce the number of ℓ_0-sketches and use the reciprocal of the $(1 - 1/e)$-th order statistic to approximate the cardinality `size_of_minimizers`$[u]$ (and hence the 1-degree of u) to obtain a nearly-linear time approximation algorithm.

There is, however, a subtle issue with the randomness involved with this algorithm. A necessary condition for the algorithm to succeed as intended is that the sketches at each step are independent of the past decisions of the algorithm. Therefore, we must remove all dependencies between previous and current queries. In Section III-B we demonstrate how correlations between steps can amplify. To avoid this problem, we must decorrelate the current state of the sketches from earlier pivoting updates to the data structures. We carefully address this issue in Section V. Instead of simply selecting a vertex with an approximate min-degree, this algorithm instead requires access to all vertices whose estimated degree is within a certain range of values. Therefore, this approximation algorithm uses a bucketing data structure, as opposed to the previous version that outputs the vertex to be pivoted. Figure 4 describes the global data structures for this version of the algorithm.

```
Global Variables: graph $G$, error tolerance $\epsilon > 0$.
  1) $k$, the number of sketches set to $50\lceil \log n\epsilon^{-2} \rceil$.
  2) $k$ independent $\ell_0$-sketch data structures:
     dynamic_sketch[1], ..., dynamic_sketch[$k$].
  3) For each vertex $u$, a balanced binary search tree
     minimizers[$u$] that stores MINIMIZER$_i(u)$ across
     all $i \in [k]$ $\ell_0$-sketches, and maintains the element in
     minimizers[$u$] with rank $\lfloor k(1-1/e) \rfloor$.
  4) A balanced binary tree quantile over all vertices
     $u$ whose key is the $\lfloor k(1-1/e) \rfloor$-ranked element in
     minimizers[$u$].
```

Figure 4: Global variables and data structures for ApproxDegreeDS, which returns (implicit) partitions of vertices into buckets with ϵ-approximate degrees.

To successfully use fewer sketches, for a given vertex u we estimate the cardinality of the set of its minimizers via its order statistics instead of using the exact cardinality as we did before with the binary search tree `size_of_minimizers`$[u]$. Exploiting correlations in the order statistics of sketches is often the underlying idea behind efficient cardinality estimation. In particular, we make use of the following lemma, which is essentially a restatement of [8, Propositions 7.1 and 7.2].

Lemma 4. *Suppose that we have k copies of the ℓ_0-sketch data structure, for $k = 50\lceil \log n\epsilon^{-2} \rceil$. Let u be any vertex such that $\deg(u) + 1 > 2\epsilon^{-1}$, and let $Q(u)$ denote the $\lfloor k(1-1/e) \rfloor$-ranked key value in the list* `minimizers`$[u]$. *Then $\frac{1-\epsilon}{\deg(u)+1} \leq Q(u) \leq \frac{1+\epsilon}{\deg(u)+1}$ with high probability.*

This idea leads to a subroutine for providing implicit access to all vertices with approximately the same degree. This is critical for our nearly-linear time algorithm, and we explain its intricacies in Section V. The pseudocode for this subroutine is given in Figure 5.

```
ApproxDegreeDS_Pivot($u$)
Input: vertex to be pivoted, $u$.
Output: updated global state.
  1) For each sketch $i = 1$ to $k$:
     a) $(v_1, v_2, \ldots, v_\ell) \leftarrow$
        dynamic_sketch[$i$].PivotVertex($u$),
        the set of vertices in the $i$-th sketch whose
        minimizers changed after we pivot out $u$.
     b) For each $j = 1$ to $\ell$:
        i) Update the values corresponding to sketch $i$
           in minimizers[$v_j$], which in turn updates
           its $\lfloor k(1-1/e) \rfloor$-ranked quantile.
        ii) Update the entry for $v_j$ in quantile with
            the new value of the $\lfloor k(1-1/e) \rfloor$-ranked
            quantile of minimizers[$v_j$].
```

```
ApproxDegreeDS_Report()
Output: bucketing of the vertices by their fill 1-degrees.
  1) For each $i = 0$ to $B = O(\log n\epsilon^{-1})$:
     a) Set $V_i$ to be the split binary tree in
        quantile that contains all nodes with
        $\lfloor k(1-1/e) \rfloor$-ranked quantiles in the range
        $[(1+\epsilon)^{-(i+1)}, (1+\epsilon)^{-i}]$.
  2) Return $(V_1, V_2, \ldots, V_B)$.
```

Figure 5: Pseudocode for the data structure that returns pointers to binary trees containing partitions of the remaining vertices into sets with ϵ-approximate degrees.

Observe that because 1-degrees are bounded by n, if we call ApproxDegreeDS_Report then $B = O(\log n\epsilon^{-1})$ with high probability by Lemma 4. Therefore, this data structure can simply return pointers to the first element in each of the partitions $V_1, V_2, \ldots, V_B$. Note that there will be overlaps between the 1-degree intervals, so determining which bucket contains a given vertex is ambiguous if its order statistic is near the boundary of an interval.

An immediate corollary of Theorem 4 is that we can provide access to approximate min-degree vertices for a fixed sequence of updates by always returning an entry from the first nonempty bucket.

Corollary 1. *For a fixed ordering $(u_1, u_2, \ldots, u_n)$, we can find $(1+\epsilon)$-approximate minimum degree vertices in each of the intermediate states in $O(m \log^3 n\epsilon^{-2})$ time.*

V. Generating Decorrelated Sequences

In this section we present a nearly-linear time $(1+\epsilon)$-approximate marginal min-degree algorithm. This algorithm relies on degree approximation via sketching, as described in Theorem 4. In particular, it uses the randomized data structure `ApproxDegreeDS`, which provides access to buckets of vertices where the i-th bucket contains vertices with fill 1-degree in the range $[(1+\epsilon)^{i-2}, (1+\epsilon)^{i+2}]$.

Theorem 6. *The* ApproxMinDegreeSequence *algorithm produces a $(1+\epsilon)$-approximate marginal min-degree ordering in expected $O(m \log^5 n\epsilon^{-2})$ time with high probability.*

At each step of this algorithm, reporting any member of the first nonempty bucket gives an approximate minimum degree vertex to pivot. However, such a choice must not have any dependence on the randomness used to get to this step, and more importantly, it should not affect pivoting decisions in future steps. To address this, we introduce an additional layer of randomization that decorrelates the ℓ_0-sketches and the choice of vertices to pivot. Most of this section focuses on efficiently decorrelating such sequences.

The pseudocode for ApproxMinDegreeSequence is given in Figure 6. This algorithm makes use of the following global data structures and subroutines.

- `ApproxDegreeDS`: Returns buckets of vertices with approximately equal 1-degrees (Section IV-B).
- ExpDecayedCandidates: Takes a sequence of values that are within $1 \pm \epsilon$ of each other, randomly perturbs the elements, and returns the new (ϵ-decayed) sequence (Section V-A).
- EstimateFill1Degree: Gives an ϵ-approximation to the 1-degree of any vertex (Section VI).

Theorem 7. *There is a data structure that maintains a component graph G° under (adversarial) vertex pivots in a total of $O(m \log^2 n)$ time and supports the operation* EstimateFill1Degree(u, ϵ)*, which given a vertex u and error threshold $\epsilon > 0$, returns with high probability an ϵ-approximation to the fill 1-degree of u by making $O(\deg(u) \log^2 n\epsilon^{-2})$ oracle queries to G°.*

The most important part of this algorithm is arguably the use of exponential random variables to construct a list of candidates that is completely uncorrelated with the randomness used to generate the ℓ_0-sketches and the choice of previous vertex pivots.

```
ApproxMinDegreeSequence$(G, \epsilon)$
Input: graph $G$ with $n$ vertices, error $\epsilon$.
Output: $(1+\epsilon)$-approximate min-degree sequence.
  1) Set a smaller error $\hat{\epsilon} \leftarrow \epsilon / \Theta(\log n)$.
  2) Initialize the approximate degree reporting data
     structure ApproxDegreeDS$(G, \hat{\epsilon})$.
  3) For each $t = 1$ to $n$:
       a) Compute buckets of 1-degrees $(V_1, \ldots, V_B)$
          $\leftarrow$ ApproxDegreeDS_Report$()$.
       b) Let $i_{\min}$ be the min index of nonempty buckets.
       c) Set candidates$[t] \leftarrow \emptyset$.
       d) For each $i = i_{\min}$ to $B$, perturb and
          rank vertices by their approximate 1-degree,
          candidates$[t] \leftarrow$ candidates$[t] \cup$
          ExpDecayedCandidates$(V_i, \hat{\epsilon}, i)$.
       e) Trim candidates$[t]$ so that its entries
          $(\delta_u, u, i)$ satisfy $(1-\delta_u)(1+\hat{\epsilon})^i < (1+\hat{\epsilon})^7 \cdot$
          $\min_{(\delta_v, v, j) \in \text{candidates}[t]} (1-\delta_v)(1+\hat{\epsilon})^j$.
       f) Let $u_t$ be the vertex that is the minimizer
          over all $(\delta_v, v, i) \in$ candidates$[t]$ of
          $(1-\delta_v)$ EstimateFill1Degree$(v, \epsilon)$.
       g) ApproxDegreeDS_Pivot$(u_t)$.
  4) Return $(u_1, u_2, \ldots, u_n)$.
```

Figure 6: Pseudocode for the $(1+\epsilon)$-approximate marginal minimum degree ordering algorithm.

A. Implicitly Sampling ϵ-Decayed Minimums

The key idea in this section is the notion of ϵ-decay, which we use to slightly perturb approximate 1-degree sequences. It is motivated by the need to decorrelate the list of vertices grouped approximately by their 1-degree from previous sources of randomness in the algorithm. In the following definition, n is the number of vertices in the original graph before pivoting and $c_1 > 1$ is a constant.

Definition 6. Given $(x_1, x_2, \ldots, x_k) \in \mathbb{R}^k$, we construct the corresponding *ϵ-decayed sequence* $(y_1, y_2, \ldots, y_k)$ by independently sampling the exponential random variables $\delta_i \sim \hat{\epsilon} \cdot \text{Exp}(1)$, where $\hat{\epsilon} = \epsilon/(c_1 \log n)$ as in line 1 in ApproxMinDegreeSequence, and letting $y_i \leftarrow (1-\delta_i)\, x_i$. We say that the *$\epsilon$-decayed minimum* of $(x_1, x_2, \ldots, x_k)$ is the value $\min(y_1, y_2, \ldots, y_k)$.

Definition 7. Given an error $\epsilon > 0$ and an ϵ-approximate 1-degree estimation routine Estimate1Degree(G, u), an *ϵ-decayed minimum degree ordering* is a sequence such that:

1) The vertex u_t corresponds to the ϵ-decayed minimum

of ESTIMATE1DEGREE $\left(G_{t-1}^{+}, v\right)$ over all remaining vertices $v \in V_{t-1}^{+}$.
2) Graph G_t^{+} is obtained after eliminating u_t from G_{t-1}^{+}.

Observe that the randomness of this perturbed degree estimator is regenerated at each step and thus removes any previous dependence. Next, we show that this adjustment is a well-behaved approximation, and then we show how to efficiently sample an ϵ-decayed minimum degree.

Lemma 5. *Let Y be an ϵ-decayed minimum of $(x_1, x_2, \ldots, x_k)$. We have $Y \geq (1-\epsilon)\min(x_1, x_2, \ldots, x_k)$ with high probability.*

By the previous lemma, to produce a $(1+\epsilon)$-approximate marginal minimum degree ordering, it suffices to compute an ϵ-decayed minimum degree ordering. Specifically, at each step we only need to find the ϵ-decayed minimum among the approximate fill 1-degrees of the remaining vertices. It turns out, however, that computing the approximate 1-degree for each remaining vertex in every iteration is expensive, so we avoid this problem by using EXPDECAYEDCANDIDATES on each bucket of vertices to carefully select a representative subset of candidates, and then we pivot out the minimizer over all buckets. The pseudocode for this subroutine is given in Figure 7, where we again let $\hat{\epsilon} = \epsilon/(c_1 \log n)$ for some constant $c_1 > 1$. Next, we show that this sampling technique is equivalent to finding the ϵ-decayed minimum over all remaining vertices with high probability.

```
EXPDECAYEDCANDIDATES($S, \hat{\epsilon}$, label)
Input: sequence $S = (s_1, s_2, \ldots, s_k)$ whose values are within a factor of $(1+c_2\hat{\epsilon})$ of each other for some constant $c_2 > 0$, error $\hat{\epsilon}$, label corresponding to $S$.
Output: candidates for the $\epsilon$-decayed minimum of $S$.
  1) Sample order statistics from Exp(1) in decreasing order until $X_{(i)}^k \geq X_{(k)}^k - c_2$:
     $(X_{(k)}^k, X_{(k-1)}^k, \ldots, X_{(k-m+1)}^k) \leftarrow$
     SAMPLEDECREASINGEXPONENTIALS($k, c_2$).
  2) For each $i = 1$ to $m$, let $\delta_i \leftarrow \hat{\epsilon} \cdot X_{(k-i+1)}^k$.
  3) Assign each $\delta_i$ to an random element $s_{\pi(i)}$ in $S$ without replacement.
  4) Return $[(\delta_1, s_{\pi(1)}, \texttt{label}), \ldots, (\delta_m, s_{\pi(m)}, \texttt{label})]$.
```

Figure 7: Pseudocode for generating an expected constant-size list of candidates for the ϵ-decayed minimum of a sequence of values that are within $(1+c_2\hat{\epsilon})$ of each other.

Note that the input sequence to EXPDECAYEDCANDIDATES requires that all its elements are within a factor of $(1+c_2\hat{\epsilon})$ of each other. We achieve this easily using the vertex buckets returned by APPROXDEGREEDS_REPORT in Section IV-B when $\hat{\epsilon}$ is the error tolerance. The next lemma shows that the approximate vertex 1-degrees in any such bucket satisfy the required input condition.

Lemma 6. *For any bucket V_i of vertices returned by* APPROXDEGREEDS_REPORT, *there is a constant $c_2 > 0$ such that all of the approximate 1-degrees are within a factor of $(1+c_2\hat{\epsilon})$ of each other.*

The most important part of EXPDECAYEDCANDIDATES is generating the order statistics efficiently. In [12, Figure 9] (SAMPLEDECREASINGEXPONENTIALS) we show how to iteratively sample the variables $X_{(i)}^k$ in decreasing order using [12, Lemma 5.3]. This technique is critically important for us because we only consider order statistics satisfying the condition $X_{(k)}^k - c_2$, which is at most a constant number of variables in expectation.

Next, to show that our algorithm is correct, we must prove that (1) the algorithm selects a bounded number of candidates in expectation at each step, and (2) the true ϵ-decayed minimum belongs to the candidate list. We analyze both of these conditions in the following lemma.

Lemma 7. *If $x_1, x_2, \ldots, x_k$ are within a factor of $(1+c_2\hat{\epsilon})$ of each other, then the ϵ-decayed minimum is among the candidates returned by* EXPDECAYEDCANDIDATES$((x_1, x_2, \ldots, x_k), \hat{\epsilon}, \cdot)$. *Furthermore, the expected number of candidates returned is bounded by the constant e^{c_2}.*

We cannot simply work with the first nonempty bucket because the randomness introduces a $1 \pm \epsilon$ perturbation. Furthermore, the bucket containing the vertex with minimum degree is dependent on the randomness of the sketches (as discussed in Theorem 4). To bypass this problem we inject additional, uncorrelated randomness into the algorithm at each step to find $O(1)$ candidates for each of the $O(\log n\hat{\epsilon}^{-1})$ buckets, which increases the number of global candidates to $O(\log n\hat{\epsilon}^{-1})$. Then in the penultimate step of each iteration, before we compute the approximate 1-degrees of candidate vertices (which is somewhat expensive), we carefully filter the global list so that the global ϵ-decayed minimum remains in the list with high probability.

Lemma 8. *Let (δ_u, u, i) be the entry over all entries $(\delta_v, v, j) \in \texttt{candidates}[t]$ that minimizes $(1-\delta_v)$* ESTIMATEFILL1DEGREE(v, ϵ). *Then with high probability we have* $(1-\delta_u)(1+\hat{\epsilon})^i \leq$
$(1+\hat{\epsilon})^7 \min_{(\delta_v, v, j) \in \texttt{candidates}[t]} (1-\delta_v)(1+\hat{\epsilon})^j$.

Now that we have all of the building blocks for decorrelation via ϵ-decayed minimums, we can prove the correctness of the $(1+\epsilon)$-approximate marginal minimum degree algorithm and bound its running time. Due to space constraints, we defer the details of the proof to the full version [12, Section 5.3].

VI. ESTIMATING THE FILL 1-DEGREE OF A VERTEX

This section discusses routines for approximating the fill 1-degree of a vertex in a partially eliminated graph. We also show how to maintain the partially eliminated graph throughout the course of the algorithm, which allows us to prove Theorem 7. The partially eliminated graph we use for degree estimation is the component graph G°, where connected components of the eliminated vertices are contracted into single vertices called *component* vertices. See Section II-A for a detailed explanation.

Our goal is to efficiently approximate the fill 1-degree of a given remaining vertex u. By the definition of fill 1-degree and the neighborhoods of component graphs, it follows that $\deg^+(u) + 1 = |\{u\} \cup N^\circ_{\text{rem}}(u) \cup \bigcup_{x \in N^\circ_{\text{comp}}(u)} N^\circ_{\text{rem}}(x)|$. In other words, the fill 1-neighborhood of u is set of remaining 1-neighbors of u in the original graph in addition to the remaining neighbors of each component neighbor of u.

This union-of-sets structure has a natural $(0,1)$-matrix interpretation, where columns correspond to remaining vertices and rows correspond to neighboring component neighborhoods of u (along with an additional row for the 1-neighborhood of u). For each row i, set the entry $A(i,j) = 1$ if vertex j is in the i-th neighborhood set and let $A(i,j) = 0$ otherwise. The problem can then be viewed as querying for the number of nonzero columns of A. Specifically, we show how one can accurately estimate fill 1-degrees using the following matrix queries:

- RowSize(A, i): Return the number of nonzero elements in row i of A.
- SampleFromRow(A, i): Returns a column index j uniformly at random from the nonzero entries of row i of A.
- QueryValue(A, i, j): Returns the value of $A(i,j)$.

The main result in this section is the follow matrix sampler.

Lemma 9. *The algorithm* EstimateNonzeroColumns *uses the three operations above that takes as input (implicit) access to a matrix A and an error ϵ, and returns an ϵ-approximation to the number of nonzero columns in A with high probability. The expected total number of operations used is $O(r \log^2 n\epsilon^{-2})$, where r is the number of rows and n is the number of columns in A.*

Before analyzing this matrix-based estimator, we verify that Lemma 9 can be used in the graph-theoretic setting to prove Theorem 7. We use the following tools for querying degrees and sampling neighbors in a component graph.

Lemma 10. *We can maintain a component graph under vertex pivots in a total time of $O(m \log^2 n)$. Additionally, this component graph data structure grants $O(\log n)$ time oracle access for:*

- *Querying the state of a vertex.*
- *Querying the component or remaining neighborhood (and hence degree) of a vertex.*
- *Uniformly sampling a remaining neighbor of a component or remaining vertex.*
- *Uniformly sampling a random component vertex.*

Lemma 9 and Lemma 10 directly imply Theorem 7, which allows us to efficiently estimate the fill 1-degrees of vertices throughout the algorithm. Once again, we refer the reader to the full version [12, Section 6 and 7] for the proofs of Lemmas 9 and 10.

VII. Maintaining Graphs Under Pivots

In this section we discuss how to maintain a dynamic 1-neighborhood sketch (described in Definition 5) of a fill graph undergoing vertex eliminations to prove Theorem 5. Since the minimum key value $R(v)$ in the 1-neighborhood of a vertex continually changes, we track the minimizer of a vertex via an eager-propagation routine that informs the neighbors of a pivoted vertex about its minimum key and propagates key values throughout the graph as needed.

In Figure 8 we list the data structures used to achieve this. We give the high-level pseudocode for the core subroutines PivotVertex, Meld, and InformRemaining.

```
1) A set $V^\circ_{\text{rem}}$ containing the remaining vertices.
2) A set $V^\circ_{\text{comp}}$ containing the component vertices.
3) For each $x \in V^\circ_{\text{comp}}$, a corresponding min heap
   remaining[$x$] that contains the key values $R(v)$
   of its remaining neighbors $v \in N^\circ_{\text{rem}}(x)$.
4) For each $u \in V^\circ_{\text{rem}}$, a min heap fill[$u$] that
   contains the union of remaining[$x$].Min() for
   each component vertex $x \in N^\circ_{\text{comp}}(u)$, as well as
   the key values of the vertices in $N^\circ_{\text{rem}}(u)$.
```

Figure 8: Data structures needed to maintain G° and an ℓ_0-sketch of G^+ under vertex pivots.

```
PivotVertex($v$)
Input: $G^\circ_t$, a vertex $v \in V^\circ_{\text{rem}}$ to be pivoted.
Output: Vertices in $V^\circ_{\text{rem}}$ whose Minimizers changed.
  1) For each vertex $y \in N^\circ_{\text{rem}}(v)$, remove $v$ and insert
     remaining[$v$].Min() to fill[$y$].
  2) For each vertex $w \in N^\circ_{\text{comp}}(v)$,
       a) remaining[$w$].Delete($R(v)$).
       b) If $R(v)$ was the old minimum in
          remaining[$w$], run InformRemaining[$w$].
       c) Meld($v, w$).
  3) Report any vertex whose Minimizer changes.
```

Figure 9: Pseudocode for pivoting a vertex.

To give some intuition for our algorithm, consider the case where no vertex is deleted but we merge neighborhoods of vertices. In this case, as the neighborhood of a particular vertex grows, the expected number of times the minimum R value in this neighborhood changes is $O(\log n)$. In particular, it is possible for the min at some vertex to change $\Omega(n)$ times due to repeated deletions. As a result, we can only bound the total, or average number of propagations. This leads to a much more involved amortized analysis, where we also use backwards analysis to explicitly bound the probability of each informing operation.

```
MELD$(v, w)$
Input: A graph state $G$, two component vertices $v$ and $w$ to be melded.
Output: Vertices in $N^{\circ}_{\text{rem}}(w)$ whose MINIMIZERs changed.
  1) For $x \in \{v, w\}, r_x = \texttt{remaining}[x].\text{MIN}()$.
  2) WLOG $r_v < r_w$. INFORMREMAINING$(w, r_w, r_v)$.
  3) Merge remaining heaps of $v$ and $w$.
  4) Report any vertex whose MINIMIZER changes.
```

Figure 10: Pseudocode for melding two component vertices.

```
INFORMREMAINING$(w, R_{\text{old}}, R_{\text{new}})$
Input: component graph $G^{\circ}$, a vertex $w \in V^{\circ}_{\text{comp}}$, old and new values for $R_{\min}(N^{\circ}_{\text{rem}}(w))$: $R_{\text{old}}$ and $R_{\text{new}}$.
Output: Vertices in $N^{\circ}_{\text{rem}}(w)$ whose MINIMIZERs changed.
  1) For each $v \in N^{\circ}_{\text{rem}}(w)$, replace the entry $R_{\text{old}}$ from fill$[v]$ with $R_{\text{new}}$.
  2) Report any vertex whose MINIMIZER changes.
```

Figure 11: Pseudocode for messaging remaining neighbors.

Given a component graph G°_t and a (remaining) vertex u to be pivoted, we use the routine PIVOTVERTEX to produce a new graph G°_{t+1}. In terms of the structure of the graph, our routine does the same thing as the traditional quotient graph model for symmetric factorization [16].

Therefore we turn our attention to the problem of maintaining the minimum R values of the neighborhoods. For a subset of vertices $V' \subseteq V^{\circ}_{\text{comp}}$, let $R_{\min}(V')$ denote the minimum R value among all its vertices. Specifically, we want to maintain the values $R_{\min}(N^{\circ}_{\text{rem}}(w))$ for every $w \in V^{\circ}_{\text{comp}}$ and $R_{\min}(N^{+}(v))$ for every $v \in V^{\circ}_{\text{rem}}$. This update procedure is a notification mechanism. When the status of a vertex changes, we update the data structures of its neighbors correspondingly. The `fill`[u] heap will then give $R_{\min}(N^{+}(u))$ and be used to estimate the fill-degree of each remaining vertex as described in Section IV.

Suppose a remaining vertex v is pivoted. Then, for a component vertex w, the content of `remaining`$[w]$ changes only if v is its neighbor. In particular, since v is no longer a remaining vertex, its entry needs to be removed from `remaining`$[z]$. Since v is now a component vertex, we need to construct `remaining`$[v]$ and update the `fill` heaps of its remaining neighbors appropriately. Furthermore, if $R(v)$ was the minimum element in `remaining`$[w]$, this is no longer the case and the other remaining neighbors of w need to be notified of this (so they can update their *fill* heaps). This is done via the call to INFORMREMAINING. The last step consists of melding the (now component) vertex v with its existing component neighbors via calls to MELD. Note that, at all times, we make a note of any remaining vertex whose MINIMIZER is updated due to the pivoting.

For every component vertex w such that $R(v)$ is the minimum value in `remaining`(w), the routine INFORMREMAINING is responsible for updating the contents in the `fill` heaps of remaining vertices adjacent to w. This routine is also required when we merge two component vertices in the algorithm MELD, since there are now more entries in the `fill` heaps of adjacent remaining vertices. We break down the cost of calls to INFORMREMAINING into two parts: when it is invoked by PIVOTVERTEX, and when it is invoked by MELD.

Lemma 11. *The expected total number of updates to remaining vertices made by* INFORMREMAINING *when invoked from* PIVOTVERTEX *over any sequence of n pivots that are independent of the R values is $O(m)$.*

Lemma 12. *Over any fixed sequence of calls to* MELD*, the expected number of updates to the* `fill` *heaps in remaining vertices is bounded by $O(m \log n)$.*

The above lemmas directly imply the main result for this section, Theorem 5. We defer the proofs of Lemma 11 and Lemma 12 to the full version [12, Section 7].

ACKNOWLEDGEMENTS

We thank John Gilbert and Gramoz Goranci for many helpful discussions regarding various topics in this paper. We also acknowledge Animesh Fatehpuria for independently obtaining a construction for covering set systems, which we use to achieve a hardness result for computing a minimum degree ordering assuming the strong exponential time hypothesis. We also thank the anonymous reviewers for their insightful comments

Matthew Fahrbach is supported in part by a National Science Foundation Graduate Research Fellowship under grant DGE-1650044. Richard Peng and Saurabh Sawlani are supported in part by the National Science Foundation under Grant No. 1637523. Part of this work was done while Shen Chen Xu was at Carnegie Mellon University.

REFERENCES

[1] Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Symposium on Discrete Algorithms (SODA)*, pages 440–452, 2017.

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

[3] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):381–388, September 2004.

[4] Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms (TALG)*, 8(4):35, 2012.

[5] Greg Bodwin and Sebastian Krinninger. Fully dynamic spanners with worst-case update time. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 17:1–17:18, 2016.

[6] Claudson Bornstein, Bruce Maggs, Gary Miller, and R Ravi. Parallelizing elimination orders with linear fill. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 274–283. IEEE, 1997.

[7] Yixin Cao and R. B. Sandeep. Minimum fill-in: Inapproximability and almost tight lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 875–880, 2017.

[8] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, December 1997.

[9] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, and Esmond G. Ng. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):353–376, 2004.

[10] Timothy A Davis, Sivasankaran Rajamanickam, and Wissam M Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016.

[11] Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.

[12] 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. *arXiv preprint arXiv:1804.04239*, 2018.

[13] Eran Gat and Shafi Goldwasser. Probabilistic search algorithms with unique answers and their cryptographic applications. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:136, 2011.

[14] André Gaul, Martin H Gutknecht, Jorg Liesen, and Reinhard Nabben. A framework for deflated and augmented krylov subspace methods. *SIAM Journal on Matrix Analysis and Applications*, 34(2):495–518, 2013.

[15] A. George and W. H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31(1):1–19, March 1989.

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

[17] 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.

[18] Oded Goldreich, Shafi Goldwasser, and Dana Ron. On the possibilities and limitations of pseudodeterministic algorithms. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 127–138. ACM, 2013.

[19] Martin H Gutknecht. A brief introduction to Krylov space methods for solving linear systems. In *Frontiers of Computational Science*, pages 53–62. Springer, 2007.

[20] 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.

[21] Bruce Hendrickson and Alex Pothen. Combinatorial scientific computing: The enabling power of discrete algorithms in computational science. In *Proceedings of the 7th International Conference on High Performance Computing for Computational Science*, VECPAR'06, pages 260–280, 2007.

[22] Bruce Hendrickson and Edward Rothberg. Improving the run time and quality of nested dissection ordering. *SIAM Journal on Scientific Computing*, 20(2):468–489, 1998.

[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] Michael Kapralov, Yin Tat Lee, C. N. Musco, C. P. Musco, and Aaron Sidford. Single pass spectral sparsification in dynamic streams. *SIAM Journal on Computing*, 46(1):456–477, 2017.

[25] Bruce M Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1131–1142. Society for Industrial and Applied Mathematics, 2013.

[26] Rasmus Kyng, Jakub Pachocki, Richard Peng, and Sushant Sachdeva. A framework for analyzing resparsification algorithms. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2032–2043. SIAM, 2017.

[27] Gary L Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 192–201. ACM, 2015.

[28] Danupon Nanongkai and Thatchaphol Saranurak. Dynamic spanning forest with worst-case update time: adaptive, las vegas, and $o(n1/2 - \epsilon)$-time. In *Symposium on Theory of Computing (STOC)*, pages 1122–1129, 2017.

[29] Danupon Nanongkai, Thatchaphol Saranurak, and Christian Wulff-Nilsen. Dynamic minimum spanning forest with subpolynomial worst-case update time. In *Symposium on Foundations of Computer Science (FOCS)*, pages 950–961, 2017.

[30] 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.

[31] Uwe Naumann and Olaf Schenk. *Combinatorial Scientific Computing*. Chapman & Hall/CRC, 1st edition, 2012.

[32] Shay Solomon. Fully dynamic maximal matching in constant update time. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 325–334. IEEE, 2016.

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

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