SUPPLEMENTARY MATERIALS: AC(K): ROBUST SOLUTION OF LAPLACIAN EQUATIONS BY RANDOMIZED APPROXIMATE CHOLESKY FACTORIZATION *

YUAN GAO†, RASMUS KYNG‡, AND DANIEL A. SPIELMAN §

SM1. Linear algebra and Cholesky background. We introduce some basic notation and definitions from linear algebra.

Laplacians and multi-graphs. For some variants of our algorithms, we will need to work with multi-graphs. We can consider an undirected multi-graph $G = (V, E)$ with positive multi-edge weights $w : E \to \mathbb{R}_+$. Note that different multi-edges between the same vertices may have different weights. In this setting, we again let $n = |V|$ denote the number of vertices and we let $m = |E|$ denote the number of multi-edges. Similar to the case of graphs without multi-edges, we can assign arbitrary directions to multi-edges to define the Laplacian of a multi-edge e as $w(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top$ and define the Laplacian of G as $\boldsymbol{L} = \sum_{e\in E} w(e)\boldsymbol{b}_e\boldsymbol{b}_e^\top$. Note that different multi-graphs may have the same Laplacian. This occurs because the coefficient of the term $\boldsymbol{b}_{(u,v)}\boldsymbol{b}_{(u,v)}^\top$ only depends on the sum $\sum_{e\in E_{uv}} w(e)$ where E_{uv} is the set of multi-edges between vertices u and v. Thus, for example, we can take a multi-graph G and produce another multi-graph G' with the same Laplacian by splitting each multi-edge e of G into k copies with weight $w(e)/k$ each.

In Section SM2, we introduce algorithms that use multi-graphs when computing an approximate Cholesky factorization of a Laplacian.

Upper and lower triangular matrices. We say a square matrix $\boldsymbol{\mathcal{U}}$ is upper triangular if it has nonzero entries $\boldsymbol{\mathcal{U}}(i, j) \neq 0$ only for $i \leq j$ (i.e. above the diagonal). Similarly, we say a square matrix $\boldsymbol{\mathcal{L}}$ is lower triangular if it has nonzero entries $\boldsymbol{\mathcal{L}}(i, j) \neq 0$ only for $i \geq j$ (i.e. below the diagonal). Often, we will work with matrices that are not upper or lower triangular, but for which we know a permutation matrix $\boldsymbol{P}$ s.t. $\boldsymbol{P\mathcal{U}P}^\top$ is upper (respectively lower) triangular. For computational purposes, this is essentially equivalent to having an upper or lower triangular matrix, and *we will refer to such matrices as upper (or lower) triangular.* The algorithms we develop for factorization will always compute the necessary permutation.

Pseudo-inverse of a product. The following fact is useful, since we often need to apply the pseudo-inverse of a matrix.

FACT SM1.1 (Pseudo-inverse of a product). *Suppose $\boldsymbol{M} = \boldsymbol{ABC}$ is square real matrix, where $\boldsymbol{A}$ and $\boldsymbol{C}$ are non-singular. Then*

$$\boldsymbol{M}^\dagger = \boldsymbol{\Pi}_{\boldsymbol{M}}\boldsymbol{C}^{-1}\boldsymbol{B}^\dagger\boldsymbol{A}^{-1}\boldsymbol{\Pi}_{\boldsymbol{M}^\top}.$$

LU-decomposition and Cholesky factorization of singular matrices. An LU-decomposition of a square matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is a factorization $\boldsymbol{M} = \boldsymbol{\mathcal{L}\mathcal{U}}$, where $\boldsymbol{\mathcal{L}}$ is a lower-triangular matrix and $\boldsymbol{\mathcal{U}}$ is an upper-triangular matrix, both up to a permutation.

When the matrix $\boldsymbol{M}$ is symmetric and positive semi-definite, we get a special case of LU-decomposition, where $\boldsymbol{\mathcal{U}} = \boldsymbol{\mathcal{L}}^\top$, i.e. $\boldsymbol{M} = \boldsymbol{\mathcal{L}\mathcal{L}}^\top$. This is known as a Cholesky Factorization.

When $\boldsymbol{M}$ is non-singular, the linear equations $\boldsymbol{\mathcal{L}y} = \boldsymbol{b}$ and $\boldsymbol{\mathcal{U}x} = \boldsymbol{y}$ can be solved by forward and backward substitution algorithms respectively (e.g. see [SM2]). These run in time $O(\text{nnz}(\boldsymbol{\mathcal{L}}))$ and $O(\text{nnz}(\boldsymbol{\mathcal{U}}))$, that is, $\boldsymbol{\mathcal{L}}^{-1}$ and $\boldsymbol{\mathcal{U}}^{-1}$ can be applied in time proportional to the number of nonzeros in $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{U}}$ respectively. This means that if a decomposition $\boldsymbol{\mathcal{L}}$,$\boldsymbol{\mathcal{U}}$ of $\boldsymbol{M}$ is known, then linear systems in $\boldsymbol{M}$ be solved in time $O(\text{nnz}(\boldsymbol{\mathcal{L}}) + \text{nnz}(\boldsymbol{\mathcal{U}}))$, since $\boldsymbol{Mx} = \boldsymbol{b}$ implies $\boldsymbol{x} = \boldsymbol{\mathcal{U}}^{-1}\boldsymbol{\mathcal{L}}^{-1}\boldsymbol{b}$.

When $\boldsymbol{M}$ is singular the same forward and backward substitution algorithms can be used to compute the pseudo-inverse, in some cases. We focus on a special case where we can instead factor $\boldsymbol{M}$ as $\boldsymbol{M} = \boldsymbol{\mathcal{L}}'\boldsymbol{\mathcal{D}}\boldsymbol{\mathcal{U}}'$, where $\boldsymbol{\mathcal{D}}$ is singular and $\boldsymbol{\mathcal{L}}'$, $\boldsymbol{\mathcal{U}}'$ are non-singular.

*The authors are listed alphabetically. The research leading to these results has received funding from the grant "Algorithms and complexity for high-accuracy flows and convex optimization" (no. 200021 204787) of the Swiss National Science Foundation. It was also supported in part by NSF Grant CCF-1562041, ONR Award N00014-16-2374, and a Simons Investigator Award to Daniel Spielman.

†CISPA, part of the work was conducted as a master's thesis at ETH Zurich; part of this work was also conducted while the author was at MPI for Informatics. (yuan.gao@cispa.de)

‡ETH Zurich (kyng@inf.ethz.ch)

§Yale University (spielman@cs.yale.edu)

For the case of interest to us, Laplacians of connected graphs, this slightly modified factorization can be trivially obtained from an LU-decomposition, by setting $\boldsymbol{\mathcal{L}}'$ equal to $\boldsymbol{\mathcal{L}}$ except $\boldsymbol{\mathcal{L}}(n,n) = 1$ and $\boldsymbol{\mathcal{U}}'$ equal to $\boldsymbol{\mathcal{U}}$ except $\boldsymbol{\mathcal{U}}(n,n) = 1$ and taking $\boldsymbol{\mathcal{D}}$ to be the identity matrix, except $\boldsymbol{\mathcal{D}}(n,n) = 0$. For our approximate factorizations of Laplacians, the matrix $\boldsymbol{M} = \boldsymbol{\mathcal{L}}'\boldsymbol{\mathcal{D}}\boldsymbol{\mathcal{U}}'$ is symmetric with $\boldsymbol{\mathcal{U}}' = (\boldsymbol{\mathcal{L}}')^\top$, hence $\boldsymbol{\Pi}_{\boldsymbol{M}^\top} = \boldsymbol{\Pi}_{\boldsymbol{M}}$. Now, by Fact SM1.1,

$$\boldsymbol{M}^\dagger = \boldsymbol{\Pi}_{\boldsymbol{M}}(\boldsymbol{\mathcal{L}}')^{-\top}\boldsymbol{\mathcal{D}}^\dagger\boldsymbol{\mathcal{L}}^{-1}\boldsymbol{\Pi}_{\boldsymbol{M}}. \tag{SM1.1}$$

The kernel of $\boldsymbol{M}$ is precisely the span of the indicator vectors of each connected component of the associated graph. Hence $\boldsymbol{\Pi}_{\boldsymbol{M}}$ can be applied by, for each connected component, subtracting the mean value for that component from every entry of the component.

Relative condition number. Given positive semi-definite matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ with the same dimensions and $\ker(\boldsymbol{B}) \subseteq \ker(\boldsymbol{A})$, we define the condition number of $\boldsymbol{A}$ relative to $\boldsymbol{B}$ as the ratio between the maximum and minimum nonzero eigenvalues of $\boldsymbol{A}\boldsymbol{B}^\dagger$.

SM2. A more accurate sampling rule using multi-edge sampling. In this section, we introduce two variants of the CliqueTreeSample sampling rule. The first variant is conceptually the simplest but performs poorly in practice, necessitating our introduction of the second variant, which is slightly more complex but performs much better in practice.

The first of these variants splits the original edges into multi-edges, which results in more fine-grained and precise sampling. The second variant introduces a second important tweak, namely merging multi-edges if too many appear between a pair of nodes.

In our experimental evaluations, we will refer to the version that splits the edges into x copies initially with no merging as AC-sx. We refer to the version that splits the edges into x copies initially and merges multi-edges to maintain at most y copies per edge as AC-sxmy. AC(k) refers to AC-skmk, and as shown in Section SM5, this generally seems to be a better trade-off than any setting AC-sxmy with $x \neq y$. AC is formally equivalent to AC(1), but we have produced separate code optimized for this case.

Generally, AC-sx produces the most reliable factorization while AC is the fastest. AC-sxmy is faster than AC-sx at computing a factorization, often much faster, while still having significantly improved reliability compared to AC.

Approximate Cholesky factorization with multi-edges and edge splits. Inspired by [SM1], our first variant splits the edges of the graph into k multi-edges, each with $\frac{1}{k}$ of the original edge weights. Note that the resulting multi-graph has the same Laplacian as the original graph, as the associated Laplacian only depends on the sum of multi-edge weights between each pair of vertices.

While Algorithm 3.2 works with a Laplacian and its associated graph, our next algorithms will explicitly maintain a multi-graph and its associated Laplacian. Because of the initial splitting of the edges and sampling based on a larger number of multi-edges, the variants we introduce in this section are more robust and are observed to produce better approximations of the Cholesky factorization. However, this comes at the cost of a higher number of nonzeros in the output factorization and longer running time.

We start by introducing a new overall factorization algorithm framework, Algorithm SM2.1, which computes a factorization of the input Laplacian by calling a sampling routine CliqueSampleMultiEdge that uses multi-edges when sampling. We then provide two different variants of CliqueSampleMultiEdge, stated in Algorithms SM2.2 and SM2.3.

Our first multi-edge-based clique sampling algorithm is Algorithm SM2.2, which we refer to as CliqueTreeSampleMultiEdge.

Algorithm SM2.1 Algorithm ApproximateCholeskyMultiEdge($\boldsymbol{L}$) outputs a lower triangular matrix $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{n\times n}$ that gives an approximate Cholesky factorization of the input Laplacian $\boldsymbol{L} \in \mathbb{R}^{n\times n}$.

procedure ApproximateCholeskyMultiEdge ($\boldsymbol{L}$)
  $\boldsymbol{S} \leftarrow \boldsymbol{L}$ and $G \leftarrow$ a multi-graph with $\boldsymbol{L}$ as its Laplacian.
    ▷ Different algorithm variants may use different rules for computing the multi-graph G given $\boldsymbol{L}$
  for $v = 1$ to $n-1$ **do** ▷ the order can be chosen adaptively
    $\boldsymbol{l}_v \leftarrow \frac{1}{\sqrt{\boldsymbol{S}(v,v)}}\boldsymbol{S}(:,v)$ ▷ $\boldsymbol{S}(:,v)$ is the vth column of $\boldsymbol{S}$
    $\boldsymbol{S} \leftarrow \boldsymbol{S} - \textsc{Star}[\boldsymbol{S}]_v + \mathsf{CliqueSampleMultiEdge}(v, \boldsymbol{S}, G)$
    ▷ $\mathsf{CliqueSampleMultiEdge}(v, \boldsymbol{S}, G)$ updates the multi-graph G and the updated $\boldsymbol{S}$ is the associated Laplacian.
  return $\boldsymbol{\mathcal{L}} = \begin{pmatrix}\boldsymbol{l}_1 & \boldsymbol{l}_2 & \cdots & \boldsymbol{l}_{n-1} & \boldsymbol{0}\end{pmatrix}$ ▷ The final column should be the all-zero vector.

Algorithm SM2.2 Algorithm CliqueTreeSampleMultiEdge($v, \boldsymbol{S}, G$) updates the multi-graph G to eliminate vertex v and add an approximate elimination clique with multi-edges in its place *and* returns $\tilde{\boldsymbol{C}}$ which approximates the elimination clique $\textsc{Clique}[\boldsymbol{S}]_v$.

procedure CliqueTreeSampleMultiEdge($v, \boldsymbol{S}, G$)
  $\tilde{\boldsymbol{C}} \leftarrow \boldsymbol{0} \in \mathbb{R}^{n\times n}$
  Let $N(G, v)$ be the set of vertices that neighbor v in G
  Let $E(G, v, i)$ be the set of multi-edges between v and i in G
  Let $w(G, v, i) = \sum_{e\in E(G,v,i)} w(e)$ be the sum of weights of multi-edges between v and i in G
  Let $w(G, v) = \sum_{i\in N(G,v)} w(G, v, i)$ be the sum of weights of multi-edges incident to v in G
    ▷ The quantities above are updated as G changes
  $d \leftarrow w(G, v)$ is the *initial* weight of multi-edges incident to v
    ▷ d is *not* updated as G changes
  for all vertices $i \in N(G, v)$ **do** ▷ pick any ordering on the neighbors
    $t \leftarrow |E(G, v, i)|$
    $\bar{w}_{vi} \leftarrow w(G, v, i)/t$ ▷ Average of the multi-edge weights between v and i in G
    Delete multi-edges $E(G, v, i)$ from G.
    ▷ Update values defined in Lines 3-6 correspondingly
    $w_{\text{new}} \leftarrow \bar{w}_{vi} \cdot \frac{w(G,v)}{d}$ ▷ Weight used for samples
    for $h = 1$ to t **do**
      Sample index j with probability $p(j) = \frac{w(G,v,j)}{w(G,v)}$
      $\tilde{\boldsymbol{C}} \leftarrow \tilde{\boldsymbol{C}} + w_{\text{new}} \cdot (\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^\top$
      Add a multi-edge between i and j of weight w_{new} to G
  return $\tilde{\boldsymbol{C}}$

Similar to Algorithm 3.3, our Algorithm SM2.2 can also be instantiated with different orderings on the neighbors of v, and again we choose an ordering by increasing weight $w(G, v, i)$.

Like our other clique sampling routines, Algorithm SM2.2 is correct in expectation, as stated in the following claim.

Claim SM2.1. *The output of Algorithm SM2.2 satisfies*

$$\mathbb{E}\left[\textsc{CliqueTreeSampleMultiEdge}(v, \boldsymbol{S}, G)\right] = \textsc{Clique}[\boldsymbol{S}]_v$$

The proof is simple and similar to the proof in the simple graph case, and hence we omit it.

Approximate Cholesky factorization with multi-edge splits and merging. We provide a second CliqueSampleMultiEdge routine which is a compromise between the previous variant and our basic version of CliqueTreeSample (Algorithm 3.3). This variant also splits the edges into k copies of multi-edge but only keeps track of the multi-edges up to some limit l. In other words, if there are more than l multi-edges for the same edge, this variant merges them down to l copies. We summarize the CliqueSample procedure of this variant as Algorithm SM2.3. This variant yields a compromise that works well in practice, as shown by our experiments in Section 4 and Section SM5.

We believe this is because Algorithm SM2.3 allows more fine-grained sampling than Algorithm 3.3 without leading to a build-up of large numbers of multi-edges late in the elimination as the graph gets denser. This undesirable phenomenon can occur in Algorithm SM2.2. Algorithm 3.3 is equivalent to a special case of this variant where we set both k and l to be 1, while Algorithm SM2.2 can be recovered by setting l to be infinity.

Algorithm SM2.3 Algorithm CliqueTreeSampleMultiEdgeMerge$(l, v, \boldsymbol{S}, G)$ updates the multi-graph G to eliminate vertex v and add an approximate elimination clique with multi-edges in its place *and* returns $\tilde{\boldsymbol{C}}$ which approximates the elimination clique $\text{Clique}[\boldsymbol{S}]_v$.

1: **procedure** CliqueTreeSampleMultiEdgeMerge$(l, v, \boldsymbol{S}, G)$
2: $\quad\tilde{\boldsymbol{C}} \leftarrow \mathbf{0} \in \mathbb{R}^{n \times n}$
3: $\quad$Let $N(G, v)$ be the set of vertices that neighbor v in G
4: $\quad$Let $E(G, v, i)$ be the set of multi-edges between v and i in G
5: $\quad$Let $w(G, v, i) = \sum_{e \in E(G,v,i)} w(e)$ be the sum of weights of multi-edges between v and i in G
6: $\quad$Let $w(G, v) = \sum_{i \in N(G,v)} w(G, v, i)$ be the sum of weights of multi-edges incident to v in G
7: $\quad$ ▷ The quantities above are updated as G changes
8: $\quad d \leftarrow w(G, v)$ is the *initial* weight of multi-edges incident to v
9: $\quad$ ▷ d is *not* updated as G changes
10: $\quad$**for all** vertices $i \in N(G, v)$ **do** ▷ pick any ordering on the neighbors
11: $\quad\quad t \leftarrow \min(|E(G, v, i)|, l)$
12: $\quad\quad \bar{w}_{vi} \leftarrow w(G, v, i)/t$ ▷ Average of the multi-edge weights between v and i in G.
13: $\quad\quad$Delete multi-edges $E(G, v, i)$ from G
14: $\quad\quad$ ▷ Update values defined in Lines 3-6 correspondingly
15: $\quad\quad w_{\text{new}} \leftarrow \bar{w}_{vi} \cdot \frac{w(G,v)}{d}$ ▷ Weight used for samples
16: $\quad\quad$**for** $h = 1$ to t **do**
17: $\quad\quad\quad$Sample index j with probability $p(j) = \frac{w(G,v,j)}{w(G,v)}$
18: $\quad\quad\quad\tilde{\boldsymbol{C}} \leftarrow \tilde{\boldsymbol{C}} + w_{\text{new}} \cdot (\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^\top$
19: $\quad\quad\quad$Add a multi-edge between i and j of weight w_{new} to G
20: $\quad$**return** $\tilde{\boldsymbol{C}}$

Note that Algorithm SM2.3 differs only from Algorithm SM2.2 by the choice of $t \leftarrow \min(|E(G, v, i)|, l)$ in Line 11.

Again, the procedure is correct in expectation.

Claim SM2.2.

$$\mathbb{E}\left[\text{CliqueTreeSampleMultiEdgeMerge}(l, v, \boldsymbol{S}, G)\right] = \text{Clique}[\boldsymbol{S}]_v$$

The proof is simple, and essentially identical to the non-multi-edge version, so we omit it. This variant has the advantage of producing a relatively more robust approximate Cholesky factorization, but at the same time not creating too many nonzeros in the output factorization. Again, with a higher number of initial splits and a higher limit for merging, this variant produces a more reliable factorization, but also has a longer runtime.

SM3. A row-operation format for Cholesky factorization output. In this section, we describe an alternative approach to representing a Cholesky factorization. In this form, the factorization becomes a product of row operation matrices. The two representations, either as a single lower triangular or as a product of row operation matrices, still result in the exact same matrix, and the representations only take constant factor difference in space consumption. Our implemented code uses this alternate row operation representation, which we have found to be more efficient as it allows certain optimizations of space usage. With appropriate optimizations, it may be possible to make standard representations of Cholesky factorization comparably efficient.

We can use the row-operations form in conjunction with the clique sampling introduced in the previous section, and again we obtain a fast algorithm for approximate Cholesky factorization. The choice of row-operations form or lower-triangular form has no impact on the sampling algorithm and does not change the output of the algorithm, it only computes a different representation of the output.[1]

[1] This is true in the RealRAM model, but the two forms may have different numerical stability properties in finite precision arithmetic.

It is well-known that Cholesky factorization can be interpreted as a sequence of row and column operations. We can use this interpretation to write the lower-triangular matrix $\boldsymbol{\mathcal{L}}$ of a Cholesky factorization $\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top$ as a product of matrices that perform a row operation.

A less appreciated fact, however, is that we have some flexibility when choosing the scaling of the row operation. We introduce a particular choice of scaling with an interesting property: When we apply a row operation to a Laplacian matrix, as an intermediate step in eliminating column v, we remove the entry of row i corresponding to the edge from v to i and in the process we create new entries that correspond exactly to the *elimination star* defined in Equation (3.3).

This motivates our decomposition of the elimination clique from Equation (3.2) into elimination stars.

We assume we are dealing with a connected graph, and describe a row-operation representation of partial Cholesky factorization when it is used to eliminate a single vertex. We can eliminate multiple vertices by repeated application.

We let

$$\boldsymbol{L} = \begin{pmatrix} d & -w & -\boldsymbol{a}^\top \\ -w & \begin{pmatrix} w & \mathbf{0}^\top \\ \mathbf{0} & \operatorname{diag}(\boldsymbol{a}) \end{pmatrix} + \boldsymbol{L}_{-1} & \\ -\boldsymbol{a} & & \end{pmatrix}.$$

Degree-1 elimination. When eliminating a vertex of degree 1, i.e. when $\boldsymbol{a} = \mathbf{0}$, we use the following factorization.

$$\begin{pmatrix} w & 0 & \mathbf{0}^\top \\ 0 & & \\ \mathbf{0} & \boldsymbol{L}_{-1} & \end{pmatrix} = \begin{pmatrix} 1 & 0 & \mathbf{0}^\top \\ 1 & & \\ \mathbf{0} & \boldsymbol{I} & \end{pmatrix} \begin{pmatrix} w & -w & \mathbf{0}^\top \\ -w & \begin{pmatrix} w & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{0} \end{pmatrix} + \boldsymbol{L}_{-1} & \\ \mathbf{0} & & \end{pmatrix} \begin{pmatrix} 1 & 1 & \mathbf{0}^\top \\ 0 & & \\ \mathbf{0} & \boldsymbol{I} & \end{pmatrix}.$$

And hence, by applying inverses of the outer matrices in the factorization,

$$\begin{pmatrix} w & -w & \mathbf{0}^\top \\ -w & \begin{pmatrix} w & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{0} \end{pmatrix} + \boldsymbol{L}_{-1} & \\ \mathbf{0} & & \end{pmatrix} = \begin{pmatrix} 1 & 0 & \mathbf{0}^\top \\ -1 & & \\ \mathbf{0} & \boldsymbol{I} & \end{pmatrix} \begin{pmatrix} w & 0 & \mathbf{0}^\top \\ 0 & & \\ \mathbf{0} & \boldsymbol{L}_{-1} & \end{pmatrix} \begin{pmatrix} 1 & -1 & \mathbf{0}^\top \\ 0 & & \\ \mathbf{0} & \boldsymbol{I} & \end{pmatrix}.$$

Edge elimination. When the vertex we're in the process of eliminating has degree more than 1, we instead apply the following factorization, where $d = \mathbf{1}^\top \boldsymbol{a} + w$ and $\theta = w/d$.

(SM3.1)
$$\begin{aligned} &\begin{pmatrix} d(1-\theta)^2 & 0 & -(1-\theta)\boldsymbol{a}^\top \\ 0 & \begin{pmatrix} w - w^2/d & -\frac{w}{d}\boldsymbol{a}^\top \\ -\frac{w}{d}\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a}) \end{pmatrix} + \boldsymbol{L}_{-1} & \\ -(1-\theta)\boldsymbol{a} & & \end{pmatrix} \\ &= \begin{pmatrix} 1-\theta & 0 & \mathbf{0}^\top \\ \theta & & \\ \mathbf{0} & \boldsymbol{I} & \end{pmatrix} \begin{pmatrix} d & -w & -\boldsymbol{a}^\top \\ -w & \begin{pmatrix} w & \mathbf{0}^\top \\ \mathbf{0} & \operatorname{diag}(\boldsymbol{a}) \end{pmatrix} + \boldsymbol{L}_{-1} & \\ -\boldsymbol{a} & & \end{pmatrix} \begin{pmatrix} 1-\theta & \theta & \mathbf{0}^\top \\ 0 & & \\ \mathbf{0} & \boldsymbol{I} & \end{pmatrix}. \end{aligned}$$

Note that the matrix on the LHS is a Laplacian because $d(1-\theta)^2 = (1-\theta)\mathbf{1}^\top\boldsymbol{a}$. And again, by applying inverses of the outer matrices in the factorization, we have

$$\begin{aligned} &\begin{pmatrix} d & -w & -\boldsymbol{a}^\top \\ -w & \begin{pmatrix} w & \mathbf{0}^\top \\ \mathbf{0} & \operatorname{diag}(\boldsymbol{a}) \end{pmatrix} + \boldsymbol{L}_{-1} & \\ -\boldsymbol{a} & & \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{1-\theta} & 0 & \mathbf{0}^\top \\ \frac{-\theta}{1-\theta} & & \\ \mathbf{0} & \boldsymbol{I} & \end{pmatrix} \begin{pmatrix} d(1-\theta)^2 & 0 & -(1-\theta)\boldsymbol{a}^\top \\ 0 & \begin{pmatrix} w - w^2/d & -\frac{w}{d}\boldsymbol{a}^\top \\ -\frac{w}{d}\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a}) \end{pmatrix} + \boldsymbol{L}_{-1} & \\ -(1-\theta)\boldsymbol{a} & & \end{pmatrix} \begin{pmatrix} \frac{1}{1-\theta} & \frac{-\theta}{1-\theta} & \mathbf{0}^\top \\ 0 & & \\ \mathbf{0} & \boldsymbol{I} & \end{pmatrix}. \end{aligned}$$

Schur complement invariance. We can also see that the Schur complement onto the remaining vertices is

$$\begin{aligned} \boldsymbol{S} &= \textsc{Sc}\left[\begin{pmatrix} d & -w & -\boldsymbol{a}^\top \\ -w & \begin{pmatrix} w & \mathbf{0}^\top \\ \mathbf{0} & \operatorname{diag}(\boldsymbol{a}) \end{pmatrix} + \boldsymbol{L}_{-1} & \\ -\boldsymbol{a} & & \end{pmatrix}\right]_{[n]\setminus\{1\}} \\ &= \begin{pmatrix} w - w^2/d & -\frac{w}{d}\boldsymbol{a}^\top \\ -\frac{w}{d}\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a}) - \frac{1}{d}\boldsymbol{a}\boldsymbol{a}^\top \end{pmatrix} + \boldsymbol{L}_{-1} \end{aligned}$$

$$= \mathrm{Sc}\left[\begin{pmatrix} d(1-\theta)^2 & 0 & -(1-\theta)\boldsymbol{a}^\top \\ 0 & \begin{pmatrix} w - w^2/d & -\frac{w}{d}\boldsymbol{a}^\top \\ -\frac{w}{d}\boldsymbol{a} & \mathrm{diag}(\boldsymbol{a}) \end{pmatrix} + \boldsymbol{L}_{-1} \\ -(1-\theta)\boldsymbol{a} & \end{pmatrix}\right]_{[n]\setminus\{1\}}.$$

Eliminating a vertex, one edge at a time. Let us summarize these observations into a statement about how to write a factorization of $\boldsymbol{L}$ that eliminates the first vertex, *which we will now denote by vertex 0.* Assume vertex 0 has degree k, and that its neighbors are vertices $1, 2, \ldots, k$.

$$\boldsymbol{L} = \begin{pmatrix} d & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \mathrm{diag}(\boldsymbol{a}) + \boldsymbol{L}_{-0} \end{pmatrix}. \tag{SM3.2}$$

We denote the weight on the edges from vertex 0 to its neighbors by $\boldsymbol{a}(1), \boldsymbol{a}(2), \ldots, \boldsymbol{a}(k)$. We then write

$$\boldsymbol{L} = \boldsymbol{\mathcal{L}}_1 \boldsymbol{\mathcal{L}}_2 \ldots \boldsymbol{\mathcal{L}}_k \begin{pmatrix} \phi & \boldsymbol{0}^\top \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix} \boldsymbol{\mathcal{L}}_k^\top \ldots \boldsymbol{\mathcal{L}}_2^\top \boldsymbol{\mathcal{L}}_1^\top \tag{SM3.3}$$

where $\boldsymbol{S} = \mathrm{Sc}[\boldsymbol{L}]_{[n]\setminus\{1\}}$, and where for $i < k$, we have

$$\boldsymbol{\mathcal{L}}_i = \begin{pmatrix} \frac{1}{1-\theta_i} & \boldsymbol{0}^\top \\ \frac{-\theta_i}{1-\theta_i}\boldsymbol{e}_i & \boldsymbol{I} \end{pmatrix} = \boldsymbol{I} + \frac{\theta_i}{1-\theta_i}(\boldsymbol{e}_1 - \boldsymbol{e}_i)\boldsymbol{e}_1^\top$$

where $\boldsymbol{e}_i$ is the i basis vector in dimension $n-1$, and $\theta_i = \frac{\boldsymbol{a}(i)\Pi_{j<i}(1-\theta_i)}{d\cdot\Pi_{j<i}(1-\theta_i)^2} = \frac{\boldsymbol{a}(i)}{d\cdot\Pi_{j<i}(1-\theta_i)}$. We can simplify this, using the observation that $\Pi_{j<i}(1-\theta_j) = \frac{d-\sum_{j<i}\boldsymbol{a}(j)}{d}$. Hence

$$\theta_i = \frac{\boldsymbol{a}(i)}{d - \sum_{j<i}\boldsymbol{a}(j)}.$$

The last factor is given by,

$$\boldsymbol{\mathcal{L}}_k = \begin{pmatrix} 1 & \boldsymbol{0}^\top \\ -\boldsymbol{e}_k & \boldsymbol{I} \end{pmatrix} = \boldsymbol{I} - \boldsymbol{e}_i\boldsymbol{e}_1^\top$$

and $\phi = \boldsymbol{a}(k)\Pi_{j<k}(1-\theta_j) = \frac{\boldsymbol{a}(k)^2}{d}$.

Computing an edgewise Cholesky factorization. In this section, we briefly remark how to repeatedly eliminate vertices to obtain a full edgewise Cholesky factorization.

Let us slightly modify the notation for the first elimination to write

$$\boldsymbol{L} = \boldsymbol{\mathcal{L}}_1^{(1)}\boldsymbol{\mathcal{L}}_2^{(1)}\cdots\boldsymbol{\mathcal{L}}_{k_1}^{(1)} \begin{pmatrix} \phi_1 & \boldsymbol{0}^\top \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix} (\boldsymbol{\mathcal{L}}_{k_1}^{(1)})^\top \cdots (\boldsymbol{\mathcal{L}}_2^{(1)})^\top (\boldsymbol{\mathcal{L}}_1^{(1)})^\top,$$

where k_1 denotes the degree of the vertex we eliminated, and $\boldsymbol{S} = \mathrm{Sc}[\boldsymbol{L}]_{[n]\setminus\{1\}}$.

Now if we recursively factor the remaining matrix $\boldsymbol{S}$ we can eventually write

$$\boldsymbol{L} = \left(\Pi_{i=1}^n \Pi_{j\sim i}\boldsymbol{\mathcal{L}}_j^{(i)}\right)\boldsymbol{\Phi}\left(\Pi_{i=1}^n \Pi_{j\sim i}\boldsymbol{\mathcal{L}}_j^{(i)}\right)^\top. \tag{SM3.4}$$

Note that $j \sim i$ denotes the set of j that are neighbors of i in the Schur complement that i is being eliminated from. $\boldsymbol{\Phi}$ is a diagonal matrix with $\boldsymbol{\Phi}(i,i)$ being the diagonal entry that results from the final single-edge elimination of vertex i. Each row operation matrix $\boldsymbol{\mathcal{L}}_j^{(i)}$ only has two entries where it differs from the identity matrix. Hence $\boldsymbol{\mathcal{L}}_j^{(i)}x$ can be computed from x in $O(1)$ time by only modifying a constant number of entries of x. Similarly, the inverse of these row operation matrices can be applied in $O(1)$.

We summarize edgewise Cholesky factorization in the pseudo-code below. The output is a diagonal matrix $\boldsymbol{\Phi}$ and a sequence of row-operation matrices $\left\{\boldsymbol{\mathcal{L}}_i^{(v)}\right\}_{v,i}$, which gives an edgewise Cholesky factorization of a Laplacian in the sense of Equation (SM3.4).

Algorithm SM3.1 Algorithm EdgewiseCholesky($\boldsymbol{L}$) outputs an edgewise Cholesky factorization of the input Laplacian $\boldsymbol{L} \in \mathbb{R}^{n \times n}$.

procedure EdgewiseCholesky($\boldsymbol{L}$)
  $\boldsymbol{\Phi} \leftarrow \boldsymbol{0}$
  $\boldsymbol{S} \leftarrow \boldsymbol{L}$
  for $v = 1$ to $n - 1$ **do** ▷ Eliminate any $n-1$ vertices in any order.
    $d \leftarrow \boldsymbol{S}(v, v)$; $\boldsymbol{a} \leftarrow -\boldsymbol{S}(v, :)$; $\boldsymbol{a}(v) \leftarrow 0$
    for $i \in \{\text{nbrs. of } v\}$, excluding one nbr. denoted by index k_v **do**
      $\theta_i \leftarrow \frac{\boldsymbol{a}(i)}{\boldsymbol{1}^\top \boldsymbol{a}}$ ▷ The neighbor eliminations can
      $\boldsymbol{\mathcal{L}}_i^{(v)} \leftarrow \boldsymbol{I} + \frac{\theta_i}{1-\theta_i}(\boldsymbol{e}_v - \boldsymbol{e}_i)\boldsymbol{e}_v^\top$ ▷ be performed in any order.
      $\boldsymbol{a}(i) \leftarrow 0$
    $\boldsymbol{\mathcal{L}}_{k_v}^{(v)} \leftarrow \boldsymbol{I} + \boldsymbol{e}_{k_v}\boldsymbol{e}_v^\top$
    $\boldsymbol{\Phi}(v, v) \leftarrow \frac{\boldsymbol{a}(k_v)^2}{d}$
    $\boldsymbol{S} \leftarrow \boldsymbol{S} - \text{Star}[\boldsymbol{S}]_v + \text{Clique}[\boldsymbol{S}]_v$
  return $\boldsymbol{\Phi}, \left\{\boldsymbol{\mathcal{L}}_i^{(v)}\right\}_{v,i}$

Obtaining an algorithm for edgewise Approximate Cholesky factorization. In Section 3.1, we saw a meta-algorithm ApproximateCholesky (Algorithm 3.2), which shows how we can replace the elimination clique in standard to Cholesky factorization (Cholesky, Algorithm 3.1) with a sampled clique approximation to obtain an approximate Cholesky factorization. And we introduced a new approach to clique sampling, CliqueTreeSample (Algorithm 3.3), which we can plug into the ApproximateCholesky meta-algorithm.

Similarly, if we replace the elimination clique in the edgewise Cholesky factorization of the previous section, then we immediately get an approximate edgewise Cholesky factorization. We frame this as a meta-algorithm, using an arbitrary multi-edge clique sampling routine, which we refer to as CliqueSampleMultiEdge.

The algorithm appears below as Algorithm SM3.2.

Algorithm SM3.2 Algorithm ApproximateEdgewiseCholesky($\boldsymbol{L}$) outputs an approximate edgewise Cholesky factorization $\boldsymbol{\Phi}, \left\{\boldsymbol{\mathcal{L}}_i^{(v)}\right\}_{v,i}$ such that $\left(\Pi_{i=1}^n \Pi_{j \sim i} \boldsymbol{\mathcal{L}}_j^{(i)}\right) \boldsymbol{\Phi} \left(\Pi_{i=1}^n \Pi_{j \sim i} \boldsymbol{\mathcal{L}}_j^{(i)}\right)^\top \approx \boldsymbol{L}$.

procedure ApproximateEdgewiseCholesky($\boldsymbol{L}$)
  $\boldsymbol{\Phi} \leftarrow \boldsymbol{0}$
  $\boldsymbol{S} \leftarrow \boldsymbol{L}$
  for $v = 1$ to $n - 1$ **do** ▷ Eliminate any $n-1$ vertices in any order.
    $d \leftarrow \boldsymbol{S}(v, v)$; $\boldsymbol{a} \leftarrow -\boldsymbol{S}(v, :)$; $\boldsymbol{a}(v) \leftarrow 0$
    for $i \in \{\text{nbrs. of } v\}$, excluding one nbr. denoted by index k_v **do**
      $\theta_i \leftarrow \frac{\boldsymbol{a}(i)}{\boldsymbol{1}^\top \boldsymbol{a}}$ ▷ The neighbor eliminations can
      $\boldsymbol{\mathcal{L}}_i^{(v)} \leftarrow \boldsymbol{I} + \frac{\theta_i}{1-\theta_i}(\boldsymbol{e}_v - \boldsymbol{e}_i)\boldsymbol{e}_v^\top$ ▷ be performed in any order.
      $\boldsymbol{a}(i) \leftarrow 0$
    $\boldsymbol{\mathcal{L}}_{k_v}^{(v)} \leftarrow \boldsymbol{I} + \boldsymbol{e}_{k_v}\boldsymbol{e}_v^\top$
    $\boldsymbol{\Phi}(v, v) \leftarrow \frac{\boldsymbol{a}(k_v)^2}{d}$
    $\boldsymbol{S} \leftarrow \boldsymbol{S} - \text{Star}[\boldsymbol{S}]_v + \text{CliqueSampleMultiEdge}(v, \boldsymbol{S})$
  return $\boldsymbol{\Phi}, \left\{\boldsymbol{\mathcal{L}}_i^{(v)}\right\}_{v,i}$

The outputs of ApproximateEdgewiseCholesky and ApproximateCholesky are identical when viewed as linear operators. We prove this claim formally below.

Claim SM3.1. *Suppose we run* ApproximateCholesky *and* ApproximateEdgewiseCholesky *using the same elimination ordering, and using* CliqueTreeSample *as the clique sampling routine, and using the same outputs of* CliqueTreeSample*, i.e. the same outcomes of the random samples. Then the output*

factorizations $\boldsymbol{\mathcal{L}}$ *from* ApproximateCholesky *and* $\boldsymbol{\Phi}, \left\{\boldsymbol{\mathcal{L}}_i^{(v)}\right\}_{v,i}$ *from* ApproximateEdgewiseCholesky *will satisfy*

$$\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top = \left(\Pi_{i=1}^n \Pi_{j\sim i} \boldsymbol{\mathcal{L}}_j^{(i)}\right) \boldsymbol{\Phi} \left(\Pi_{i=1}^n \Pi_{j\sim i} \boldsymbol{\mathcal{L}}_j^{(i)}\right)^\top.$$

In other words, the only difference between the two algorithms is in the formatting of the output.

We prove the claim in Appendix SM3.1.

Remark SM3.2. *The claim above shows the algorithms differ only in the formatting of their outputs. In fact, we can also convert the output of either to the output of the other in linear time.*

From this, we can also deduce that

$$\mathbb{E}\left[\left(\Pi_{i=1}^n \Pi_{j\sim i} \boldsymbol{\mathcal{L}}_j^{(i)}\right) \boldsymbol{\Phi} \left(\Pi_{i=1}^n \Pi_{j\sim i} \boldsymbol{\mathcal{L}}_j^{(i)}\right)^\top\right] = \boldsymbol{L}.$$

Furthermore, we can see by inspection that Claim 3.2 essentially applies: Each row operation $\boldsymbol{\mathcal{L}}_j^{(i)}$ consists of an identity matrix with two entries adjusted, one on the diagonal, and one below the diagonal. This means the matrix and its inverse can be applied in $O(1)$ time, and the overall sequence of row operations (or their inverses) can be applied in $O(m \log m)$ time when the input matrix has $O(m)$ nonzero entries.

SM3.1. Equivalence of the output representations. In this section, we prove Claim SM3.1, about the equivalence of edgewise and standard Cholesky factorization.

Proof. We can write a Laplacian $\boldsymbol{L}$ with terms corresponding to edges of the first vertex explicitly separated out as:

$$\boldsymbol{L} = \begin{pmatrix} d & -\boldsymbol{a}^\top \\ -\boldsymbol{a} & \operatorname{diag}(\boldsymbol{a}) + \boldsymbol{L}_{-1} \end{pmatrix}. \tag{SM3.5}$$

Here $\boldsymbol{L}_{-1}$ is the graph Laplacian corresponding to the induced subgraph on the vertex set with vertex 1 removed.

From Section 3.1, we know that it is possible to write

$$\boldsymbol{L} = \begin{pmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \boldsymbol{S} \end{pmatrix} + \frac{1}{d} \begin{pmatrix} d \\ -\boldsymbol{a} \end{pmatrix} \begin{pmatrix} d \\ -\boldsymbol{a} \end{pmatrix}^\top \tag{SM3.6}$$

where $\boldsymbol{S} = \textsc{Sc}[\boldsymbol{L}]_{[n]\setminus\{1\}}$.

If vertex 1 has k neighbors, then using the elimination procedure described in Section SM3 to eliminate the first row and column of the matrix will result in a partial factorization. Thus,

$$\boldsymbol{L} = \boldsymbol{\mathcal{L}}_1 \boldsymbol{\mathcal{L}}_2 \ldots \boldsymbol{\mathcal{L}}_k \begin{pmatrix} \phi & \mathbf{0}^\top \\ \mathbf{0} & \boldsymbol{S} \end{pmatrix} \boldsymbol{\mathcal{L}}_k^\top \ldots \boldsymbol{\mathcal{L}}_2^\top \boldsymbol{\mathcal{L}}_1^\top \tag{SM3.7}$$

where each $\boldsymbol{\mathcal{L}}_i^\top$ is a lower-triangular matrix corresponding to a single row-operation.

We can prove the following:

$$\begin{pmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \boldsymbol{S} \end{pmatrix} = \boldsymbol{\mathcal{L}}_1 \boldsymbol{\mathcal{L}}_2 \cdots \boldsymbol{\mathcal{L}}_k \begin{pmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \boldsymbol{S} \end{pmatrix} \boldsymbol{\mathcal{L}}_k^\top \cdots \boldsymbol{\mathcal{L}}_2^\top \boldsymbol{\mathcal{L}}_1^\top \tag{SM3.8}$$

and

$$\frac{1}{d} \begin{pmatrix} d \\ -\boldsymbol{a} \end{pmatrix} \begin{pmatrix} d \\ -\boldsymbol{a} \end{pmatrix}^\top = \boldsymbol{\mathcal{L}}_1 \boldsymbol{\mathcal{L}}_2 \cdots \boldsymbol{\mathcal{L}}_k \begin{pmatrix} \phi & \mathbf{0}^\top \\ \mathbf{0} & \mathbf{0} \end{pmatrix} \boldsymbol{\mathcal{L}}_k^\top \cdots \boldsymbol{\mathcal{L}}_2^\top \boldsymbol{\mathcal{L}}_1^\top. \tag{SM3.9}$$

To establish Equation (SM3.8), we observe that each row operation factor takes the form

$$\boldsymbol{\mathcal{L}}_i = \begin{pmatrix} \boldsymbol{b}_i & \mathbf{0}^\top \\ | & \boldsymbol{I}_{(n-1)\times(n-1)} \end{pmatrix}$$

for some vector $\boldsymbol{b}_i \in R^n$. But, given any vector $\boldsymbol{b}_i \in R^n$ and any matrix $\boldsymbol{C} \in R^{(n-1)\times d}$, with any number of columns d, we have

$$\begin{pmatrix} \boldsymbol{b} & \mathbf{0}^\top \\ | & \boldsymbol{I}_{(n-1)\times(n-1)} \end{pmatrix} \begin{pmatrix} \mathbf{0}^\top \\ \boldsymbol{C} \end{pmatrix} = \begin{pmatrix} \mathbf{0}^\top \\ \boldsymbol{C} \end{pmatrix}.$$

Repeatedly applying this observation the right hand side of Equation (SM3.8) lets us conclude that it equals the left hand side, proving the equation holds.

Now, by equating the two right hand sides of Equation (SM3.6) and Equation (SM3.3), and simplifying using Equation (SM3.8), we conclude that Equation (SM3.9) holds.

The proof of Equation (SM3.9) only relies on the row operation matrices having the form

$$\boldsymbol{\mathcal{L}}_i = \begin{pmatrix} \boldsymbol{b}_i & \mathbf{0}^\top \\ | & \boldsymbol{I}_{(n-1)\times(n-1)} \end{pmatrix}$$

for some $\boldsymbol{b}_i$ and does not require $\boldsymbol{S}$ to be a Schur complement.

Consequently, it holds for any $\tilde{\boldsymbol{S}}$ matrix that

$$\begin{pmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \tilde{\boldsymbol{S}} \end{pmatrix} = \boldsymbol{\mathcal{L}}_1 \boldsymbol{\mathcal{L}}_2 \cdots \boldsymbol{\mathcal{L}}_k \begin{pmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \tilde{\boldsymbol{S}} \end{pmatrix} \boldsymbol{\mathcal{L}}_k^\top \cdots \boldsymbol{\mathcal{L}}_2^\top \boldsymbol{\mathcal{L}}_1^\top \tag{SM3.10}$$

and hence

$$\begin{pmatrix} 0 & \mathbf{0}^\top \\ \mathbf{0} & \tilde{\boldsymbol{S}} \end{pmatrix} + \frac{1}{d} \begin{pmatrix} d \\ -\boldsymbol{a} \end{pmatrix} \begin{pmatrix} d \\ -\boldsymbol{a} \end{pmatrix}^\top = \boldsymbol{\mathcal{L}}_1 \boldsymbol{\mathcal{L}}_2 \cdots \boldsymbol{\mathcal{L}}_k \begin{pmatrix} \phi & \mathbf{0}^\top \\ \mathbf{0} & \tilde{\boldsymbol{S}} \end{pmatrix} \boldsymbol{\mathcal{L}}_k^\top \cdots \boldsymbol{\mathcal{L}}_2^\top \boldsymbol{\mathcal{L}}_1^\top. \tag{SM3.11}$$

∎

Thus, by taking $\tilde{\boldsymbol{S}} = \boldsymbol{L}_{-1} + \textsc{CliqueTreeSample}(v, \boldsymbol{S})$, we can conclude that the partial factorizations computed in one step of $\textsc{ApproximateCholesky}(\boldsymbol{L})$ and $\textsc{EdgewiseApproximateCholesky}(\boldsymbol{L})$ are identical. We can apply this equivalence repeatedly to conclude that the two approximate factorizations returned by $\textsc{ApproximateCholesky}(\boldsymbol{L})$ and $\textsc{EdgewiseApproximateCholesky}(\boldsymbol{L})$ are identical, assuming we (a) use the same elimination ordering for both, and (b) use $\textsc{CliqueTreeSample}$ as the clique sampling routine, and using the same outputs of $\textsc{CliqueTreeSample}$, i.e. the same outcomes of the random samples. Thus the approximate factorizations returned by $\textsc{ApproximateCholesky}(\boldsymbol{L})$ and $\textsc{EdgewiseApproximateCholesky}(\mathbf{L})$ are identical when regarded as operators.

SM4. Unbiased Approximate Cholesky factorization. In this section, we prove Claim 3.1, which was first shown in [SM1].[2] We first restate the claim:

Claim SM4.1. *When* $\textsc{ApproximateCholesky}$ *is run with an unbiased clique sampling routine* $\textsc{CliqueSample}$ *so that*

$$\mathbb{E}\left[\textsc{CliqueSample}(v, \boldsymbol{S})\right] = \textsc{Clique}[\boldsymbol{S}]_v.$$

Then letting $\boldsymbol{\mathcal{L}} = \textsc{ApproximateCholesky}(\boldsymbol{L})$*, we have*

$$\mathbb{E}\left[\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top\right] = \boldsymbol{L}.$$

In other words, the approximate Cholesky factorization equals the original matrix in expectation, which is in turn also equal, viewed as a linear operator, to any exact Cholesky factorization.

Proof. Let $\boldsymbol{S}_i$ denote $\boldsymbol{S}$ in Algorithm 3.2 after i eliminations, and note that $\boldsymbol{S}_0 = \boldsymbol{L}$. Let

$$\boldsymbol{L}_i = \boldsymbol{S}_i + \sum_{j=1}^{i} \boldsymbol{l}_j \boldsymbol{l}_j^\top.$$

Suppose the ith vertex to be eliminated is vertex v. Conditional on the samples before the ith elimination, we have

$$\mathbb{E}\left[\boldsymbol{L}_i\right] = \mathbb{E}\left[\boldsymbol{S}_i + \sum_{j=1}^{i} \boldsymbol{l}_j \boldsymbol{l}_j^\top\right].$$

[2] See also Lemma 4.1 in lecture notes at http://kyng.inf.ethz.ch/courses/AGAO20/lectures/lecture9_apxgauss.pdf.

$$
\begin{aligned}
&= \mathbb{E}\left[\boldsymbol{S}_{i-1} - \textsc{Star}[\boldsymbol{S}_{i-1}]_v + \textsc{CliqueSample}(v, \boldsymbol{S}_{i-1}) + \sum_{j=1}^{i} \boldsymbol{l}_j \boldsymbol{l}_j^\top\right] \\
&= \boldsymbol{S}_{i-1} - \textsc{Star}[\boldsymbol{S}_{i-1}]_v + \mathbb{E}\left[\textsc{CliqueSample}(v, \boldsymbol{S}_{i-1})\right] + \sum_{j=1}^{i} \boldsymbol{l}_j \boldsymbol{l}_j^\top \\
&= \boldsymbol{S}_{i-1} - \textsc{Star}[\boldsymbol{S}_{i-1}]_v + \textsc{Clique}[\boldsymbol{S}_{i-1}]_v + \sum_{j=1}^{i} \boldsymbol{l}_j \boldsymbol{l}_j^\top \\
&= \boldsymbol{S}_{i-1} + \sum_{j=1}^{i-1} \boldsymbol{l}_j \boldsymbol{l}_j^\top \\
&= \boldsymbol{L}_{i-1}.
\end{aligned}
$$

We can now chain together expectations to conclude that over all the randomness of the algorithm

$$\mathbb{E}\left[\boldsymbol{\mathcal{L}}\boldsymbol{\mathcal{L}}^\top\right] = \mathbb{E}\left[\boldsymbol{L}_n\right] = \boldsymbol{L}.$$

□

SM5. Fill-in, splitting, and merging, and preconditioner quality: experiments on further variants. In this section, we report some further experimental statistics on the qualities of all variants of the approximate Cholesky factorization on the difficult problems described earlier. The Approximate Cholesky factorizations reported here are

- AC: no splitting, and merging of multi-edges down to 1 copy.
- AC-s1: no splitting and no merging.
- AC-s2: original edges split into 2 copies, no merging.
- AC-s2m2, aka. AC(2): original edges split into 2, merging multi-edges down to 2 copies.
- AC-s3m3, aka. AC(3): original edges split into 2, merging multi-edges down to 2 copies.

For these factorizations, we report the following statistics:

- Total time to solve each linear equation, normalized by nonzero count.
- Condition number of the SDDM matrix and the approximate factorizations. This captures the quality of approximate factorizations.
- Ratio between the size of the output factorization and the input Laplacian (adjacency) matrix.

Results are shown in Table SM5.1. Here we remark that AC and AC-s1 did not achieve the target 10^{-8} tolerance on the Sachdeva star when $k = 700$. For Sachdeva stars, most sampled edges of our solvers occur within the complete graphs on the leaves, hence causing the ratio between the sizes of output factorizations and input Sachdeva star Laplacians to be close to 1. We see that the initial splitting plays a very significant role in the quality of the output factorization. AC-s1 produced factorizations that have very large condition numbers relative to the input matrix on Sachdeva star, even though it does not compress any multi-edges created during the elimination. On the other hand, the condition number and iteration counts for AC-s2m2 are still very much comparable to that of AC-s2, despite that it only keeps up to 2 multi-edges. This suggests that allowing no limits on multi-edges does not increase quality much compared to versions that merge, and versions without merging are noticeably slower. AC-s3m3 produced the most reliable factorizations and beats AC-s2 in terms of the runtime, but is not as fast as AC-s2m2.

Table SM5.1: Comparison between variants of the approximate Cholesky factorization using different multi-edge splitting and merging approaches.

Instance	nonzeros	AC				AC-s1				AC-s2			
	nnz	$t_{\text{total}}/\text{ nnz}$	N_{iter}	fill-in	Condition num.	$t_{\text{total}}/\text{ nnz}$	N_{iter}	fill-in	Condition num.	$t_{\text{total}}/\text{ nnz}$	N_{iter}	fill-in	Condition num.
	(M)	μs				μs				μs			
Weighted chimera	54.6	2.88	26	2.5	14.1	4.72	24	3.06	11.7	8.68	19	4.19	7.61
Weighted SDDM chimera	54.2	3.05	24	2.42	11.3	4.3	24	2.79	12.2	7.1	18	3.76	6.62
Sachdeva Star	172	2.2 *	408	1.0	9660.0	3.13*	189	1.0	3090.0	5.97	31	1.0	102.0
High contrast coefficient Poisson grid	200	2.24	60	2.54	80.2	3.82	53	3.11	54.7	6.32	44	4.33	35.8
Anisotropic coef. Poisson grid, variable weight	200	2.73	39	2.43	27.2	3.24	32	2.9	16.7	5.73	23	3.95	9.14

Instance	nonzeros	AC-s2m2				AC-s3m3			
	nnz	$t_{\text{total}}/\text{ nnz}$	N_{iter}	fill-in	Condition num.	$t_{\text{total}}/\text{ nnz}$	N_{iter}	fill-in	Condition num.
	(M)	μs				(μs)			
Weighted chimera	54.6	3.9	19	3.45	7.81	5.13	17	4.2	6.15
Weighted SDDM chimera	54.2	4.26	19	3.27	8.18	5.39	17	3.94	5.83
Sachdeva Star	172	0.84	44	1.0	64.6	1.15	31	1.0	21.4
High contrast coefficient Poisson grid	200	3.44	45	3.57	40.6	4.4	38	4.39	35.5
Anisotropic coef. Poisson grid, variable weight	200	2.33	26	3.33	11.5	3.12	21	3.99	8.15

SM6. Interior Point Method equations on Spielman graphs. In this appendix, we report the results of additional experiments on solving Laplacian linear equations that arise from taking Interior Point Method (IPM) Newton step in when using an IPM to solve maximum flow. In these additional experiments, we test a family of graphs called *Spielman graphs*. These are conjectured to be a hard instance for short-step Maximum Flow IPMs. That is, it is thought that short-step IPMs need many Newton steps to converge on graphs. Spielman graphs are built recursively across a number of levels. There is a unique Spielman graph with k levels, and it has $O(k^3)$ vertices and edges. This family has a high degree of symmetry, which allows an algorithm based on implicit representations (which we call R-space representation) of the flow solution to compute intermediate states of the IPM very quickly. We use this to speed up our experiments, by only employing a Laplacian solver at a few points spaced evenly throughout a run of the IPM, and using R-space representation for majority of the Newton steps. We run the IPM up to precision 10^{-6}. For each number of levels $k = 100, 200, 300, 400, 500, 600$ (or in terms of number of edges of the Spielman graphs: from roughly a million up to 200 million), we take roughly 10 Laplacians that occur when running Maximum Flow IPM. The Laplacians are distributed evenly across precision ranges including the first and the final Laplacians during each run of the IPM. Table SM6.1 reports the details of the performances of the solvers on Laplacians from running our Maximum Flow IPM on Spielman graphs. The results of this experiment are shown in Table SM6.1.

Table SM6.1: Maximum Flow IPM Laplacians from Spielman graphs.

edges	# instances	AC $t_{\text{total}}/\text{ nnz}$			AC(2) $t_{\text{total}}/\text{ nnz}$			CMG $t_{\text{total}}/\text{ nnz}$		
n		median	0.75	max	median	0.75	max	median	0.75	max
(K)		(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)
1030	10	0.162	0.172	0.247	0.28	0.287	0.313	1.81	1.84	1.93
8100	10	0.194	0.195	0.223	0.296	0.305	0.369	1.95	1.97	2.02
2.72×10^4	10	0.229	0.25	0.286	0.346	0.354	0.389	2.24	2.25	2.29
6.44×10^4	10	0.247	0.268	0.291	0.357	0.367	0.396	2.43	2.46	2.71
1.26×10^5	10	0.264	0.279	0.362	0.422	0.428	0.428	2.56	2.57	3.06
2.17×10^5	11	0.418	0.437	0.735	0.502	0.51	0.809	2.63	2.68	3.28

edges	# instances	Hypre $t_{\text{total}}/\text{ nnz}$			PETSc $t_{\text{total}}/\text{ nnz}$			ICCG $t_{\text{total}}/\text{ nnz}$			IC(2)CG $t_{\text{total}}/\text{ nnz}$		
n		median	0.75	max	median	0.75	max	median	0.75	max	median	0.75	max
(K)		(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)	(μs)
1030	10	0.346	0.361	0.449	0.76	1.06	1.81 *	3.82	3.9	4.08	5.93	6.73	6.98
8100	10	0.454	0.48	1.22	1.22	1.85	3.32 *	9.14	10.4 *	10.5 *	11.8	12.8	13
2.72×10^4	10	0.533	0.551	0.621	1.84	2.47	2.61	10.9	16.9 *	20.1 *	13	17.1	20.2
6.44×10^4	10	0.583	0.62	0.725	1.68	2.05	3.27	14 *	14.6 *	15.2 *	14.4	18.2	28
1.26×10^5	10	0.633	0.665	0.768	2.38	2.6	3.24	9.3 *	11 *	19.2 *	16.5	17.2	21.5
2.17×10^5	11	0.643	0.693	0.811	2.31	2.82	3.98	13 *	16.2 *	22.1 *	21.1	22.6	27.4

REFERENCES

[SM1] R. Kyng and S. Sachdeva, *Approximate gaussian elimination for laplacians-fast, sparse, and simple*, in 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), IEEE, 2016, pp. 573–582.

[SM2] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*, vol. 50, Siam, 1997.