VAC: A Volume-sampling-based Elimination Rule for Approximate Cholesky Factorization

Yves Baumann, Rasmus Kyng*, Gernot Zöcklein*

July 28, 2026

Abstract

We propose Volume Appproximate Cholesky (VAC), an alternative sampling rule for practical approximate Cholesky algorithms. Our rule samples a uniformly random spanning tree of the arising product clique to reduce the fill-in generated at each step. Sampling a random spanning tree preserves the edgewise marginals of the provably correct scheme of Kyng-Sachdeva (FOCS'16), while ensuring connectivity in the spirit of the practical rule proposed in Gao-Kyng-Spielman (SISC'26). Our sampling method is simple, provably linear time and also admits a $O(\log n)$ depth parallel implementation.

1 Preliminaries

The main object of study will be *product cliques*. These are the family of graphs that arise as a fill-in when implementing a cholesky factorization of the Laplacian L of a graph. A line of work has shown both theoretical guarantees as well as practical performance for *Approximate Cholesky* algorithms, where the dense fill-in cliques are replaced by sparse sub samples [5, 2]. In this note, we show how to sample a uniformly random spanning tree of a product clique.

To start, let $V = \{1, \ldots, n\}$ with $n \geq 2$, and let $w : V \to \mathbb{R}_{>0}$ be a positive vertex-weight function. Write

$$W := \sum_{x \in V} w(x).$$

Definition 1.1 (Product clique)**.** *The* product clique K_w *is the complete graph on* V *with edge conductances*

$$c_{uv} := w(u)w(v), \qquad u \neq v.$$

For an edge $e = (u, v)$, let

$$b_e := e_u - e_v.$$

If L is the Laplacian of K_w, the leverage score of edge $e = (u, v)$ is defined as

$$\tau_{uv} := c_{uv} \cdot b_e^\top L^+ b_e.$$

Note that we can write the Laplacian as

$$L = W \operatorname{diag}(w) - ww^\mathsf{T}.$$

We will need the standard Matrix Determinant lemma. See for example [4] for a proof.

*The research leading to these results has received funding from the starting grant "A New Paradigm for Flow and Cut Algorithms" (no. TMSGI2 218022) and grant no. 200021 204787 of the Swiss National Science Foundation.

Lemma 1.2 (Matrix Determinant Lemma)**.** *Let $A \in \mathbb{R}^{n\times n}$ be an invertible matrix, and let $u, v \in \mathbb{R}^n$. Then*

$$\det(A + uv^\top) = (1 + u^\top A^{-1} v)\det(A).$$

2 The weighted spanning-tree distribution

For a spanning tree T of a graph with edge conductance $c \in \mathbb{R}^E$, write

$$c(T) := \prod_{e\in T} c_e.$$

Let $\mathcal{T}$ be a random spanning tree. The weighted uniform spanning-tree distribution is

$$\mathbb{P}[\mathcal{T} = T] = \frac{c(T)}{\sum_{T'} c(T')}.$$

In the product clique,

$$c(T) = \prod_{\{u,v\}\in T} w(u)w(v) = \prod_{x\in V} w(x)^{\deg_T(x)}.$$

Lemma 2.1. *The weighted spanning-tree partition function of K_w is*

$$Z_w := \sum_T c(T) = W^{n-2} \prod_{x\in V} w(x).$$

Consequently, every fixed spanning tree T satisfies

$$\mathbb{P}[\mathcal{T} = T] = \frac{1}{W^{n-2}} \prod_{x\in V} w(x)^{\deg_T(x)-1}.$$

Proof. Fix a root $r \in V$. By the matrix-tree theorem (see, e.g., section 13 in [6]), Z_w is the determinant of the principal minor obtained from L by deleting row and column r. Writing $D_{-r} := \mathrm{diag}(w(x) : x \neq r)$ and $w_{-r} := (w(x))_{x\neq r}$, the matrix determinant lemma gives

$$\begin{aligned}
Z_w &= \det\big(WD_{-r} - w_{-r}w_{-r}^\mathsf{T}\big) \\
&= \det\big(WD_{-r}I - WD_{-r}(WD_{-r})^{-1}w_{-r}w_{-r}^\mathsf{T}\big) \\
&= \det(WD_{-r})\det\Big(I - (WD_{-r})^{-1}w_{-r}w_{-r}^\mathsf{T}\Big) \\
&= \det(WD_{-r})\det\Big(I - w_{-r}^\mathsf{T}(WD_{-r})^{-1}w_{-r}\Big) \\
&= W^{n-1}\prod_{x\neq r} w(x)\left(1 - \frac{W - w(r)}{W}\right) \\
&= W^{n-2}\prod_{x\in V} w(x).
\end{aligned}$$

Dividing $c(T) = \prod_x w(x)^{\deg_T(x)}$ by Z_w gives the claimed law. □

Leverage scores from spanning-tree marginals We start with the standard identity

$$\mathbb{P}[e \in \mathcal{T}] = \tau_e$$

in order to relate weighted spanning-tree marginals to leverage scores. See for example [6] for a proof of this identity.

Proposition 2.2. *For every distinct $u, v \in V$,*

$$\tau_{uv} = \frac{w(u) + w(v)}{W}.$$

Proof. Fix $e = \{u, v\}$. Contracting e merges u and v into a vertex of weight $w(u) + w(v)$. Indeed, for every $x \notin \{u, v\}$, the total conductance from the contracted vertex to x is

$$w(u)w(x) + w(v)w(x) = (w(u) + w(v))w(x).$$

Thus the contracted graph is again a product clique, and we can use Lemma 2.1 to derive the total weight of spanning trees containing e as

$$c_{uv}\, W^{n-3}(w(u) + w(v)) \prod_{x \neq u,v} w(x).$$

Dividing by the full partition function

$$Z_w = W^{n-2} w(u)w(v) \prod_{x \neq u,v} w(x)$$

gives

$$\mathbb{P}[e \in \mathcal{T}] = \frac{w(u) + w(v)}{W}.$$

The spanning-tree marginal identity now yields the formula for τ_{uv}. □

3 Sampling a Uniform Spanning Tree

We will use Prüfer codes as a sampling device. See [3] for details on the relationship between Prüfer codes and labeled spanning trees of V. Every labeled tree on V corresponds bijectively to a Prüfer sequence

$$P = (P_1, \ldots, P_{n-2}) \in V^{n-2},$$

and if T is the tree corresponding to sequence P, then

$$\deg_T(x) - 1 = |\{t \in \{1, 2, \ldots, n-2\} : P_t = x\}|.$$

Therefore, if the entries of P are sampled independently according to

$$\mathbb{P}[P_t = x] = \frac{w(x)}{W},$$

then

$$\mathbb{P}[P = P(T)] = \frac{1}{W^{n-2}} \prod_{x \in V} w(x)^{\deg_T(x)-1},$$

which is exactly the spanning-tree law derived above. This observation leads to the following simple algorithm for sampling a uniform spanning tree from product cliques. Below, DecodePrüfer is any algorithm that decodes a Prüfer sequence into the tree it encodes.

Algorithm 1 Weighted uniform spanning tree of a product clique

Require: $V = \{1, \ldots, n\}$ and positive weights $w(v)$
1: $W \leftarrow \sum_{v \in V} w(v)$
2: **for** $t = 1, \ldots, n-2$ **do**
3: Independently sample $P_t \in V$ with probability $w(P_t)/W$
4: **end for**
5: $T \leftarrow \text{DecodePrüfer}(P_1, \ldots, P_{n-2})$
6: For every $e = (u, v) \in T$, assign it conductance $W \frac{w(u)w(v)}{w(u)+w(v)}$
7: **return** T

Lemma 3.1. *For the product clique K_w, the re-weighted uniform spanning tree T of Algorithm 1 satisfies that*

$$\mathbb{E}[L_T] = L.$$

Moreover, it can be sampled sequentially with $O(n)$ work. Alternatively, it can be sampled with $O(n \log n)$ work and $O(\log n)$ parallel depth.

Proof. The expectation statement is true because $\Pr[e \in T] = \tau_e$. Also, clearly the Prüfer sequence can be sampled with the stated bounds. Standard decoding then takes linear work; for parallel decoding, one may use the algorithm of [1]. □

4 Use in Approximate Cholesky Algorithms

Sampling a weighted clique typically comes up in approximate Cholesky factorization. Consider a Laplacian S of a graph H and a vertex v of H. The standard Cholesky factorization of S is given in algorithm 2

Algorithm 2 Cholesky Factorization

Require: Graph Laplacian L from graph $G = (V, E, w)$, $|V| = n$.
Ensure: Lower-triangular matrix $\mathcal{L}$ such that $\mathcal{L}\mathcal{L}^\top = L$
1: $S_0 \leftarrow L$
2: **for** $i = 1, \ldots, n-1$ **do**
3: $\ell_i \leftarrow \dfrac{1}{\sqrt{S_{i-1}(i,i)}} S_{i-1}(:, i)$
4: $S_i \leftarrow S_{i-1} - \dfrac{1}{S_{i-1}(i,i)} S_{i-1}(:, i) S_{i-1}(:, i)^\top$
5: **end for**
6: $\ell_n \leftarrow \mathbf{0}_{n \times 1}$
7: $\mathcal{L} \leftarrow \begin{bmatrix} \ell_1 & \cdots & \ell_n \end{bmatrix}$
8: **return** $\mathcal{L}$

For ease of notation, we define $\text{STAR}(v, S)$ to be the Laplacian of the subgraph of H consisting of edges incident on v. We define

$$\text{CLIQUE}(v, S) = \text{STAR}(v, S) - \frac{1}{S(v,v)} S(:, v) S(:, v)^\top.$$

For example, suppose

$$L = \begin{pmatrix} W & -a^\top \\ -a & \operatorname{diag}(a) + L_{-1} \end{pmatrix}.$$

Then

$$\operatorname{STAR}(1, L) = \begin{pmatrix} W & -a^\top \\ -a & \operatorname{diag}(a) \end{pmatrix} \qquad \text{and} \qquad \operatorname{CLIQUE}(1, L) = \begin{pmatrix} 0 & 0 \\ 0 & \operatorname{diag}(a) - \frac{1}{W} a a^\top \end{pmatrix}.$$

Now, $\operatorname{CLIQUE}(1, L)$, and specifically $aa^\top$ can be dense. This is the reason why Gaussian Elimination can be slow. At each step, one vertex is eliminated, and the clique is added to the remaining subgraph, which can cost up to $O(\deg(v)^2)$ time. However, the term $aa^\top$ has exactly the *product cliques* structure described in the earlier sections. Previous works have studied how one can sparsify this clique in theory [5] as well as in practice [2] to find good factorizations of the Laplacian S. We propose lemma 3.1 as an alternative clique sampling rule, that can be used to sample the resulting cliques in approximate Cholesky. The result is algorithm 3

Algorithm 3 Volume Approximate Cholesky(s)

Require: Graph Laplacian L from graph $G = (V, E, w)$, $|V| = n$.
Ensure: Lower-triangular matrix $\mathcal{L}$.

$S_0 \leftarrow L$
Generate a random permutation π of $[n]$
for $i = 1, \ldots, n-1$ **do**
    $\ell_i \leftarrow \dfrac{1}{\sqrt{S_{i-1}\big(\pi(i), \pi(i)\big)}} S_{i-1}\big(:, \pi(i)\big)$
    $S_i \leftarrow S_{i-1} - \operatorname{STAR}\big(\pi(i), S_{i-1}\big) + \textsc{WeightedUSTSampling}(\ell_i,\, s)$
        ▷ See algorithm 1
end for
$\ell_n \leftarrow \mathbf{0}_{n \times 1}$
$\mathcal{L} \leftarrow \begin{bmatrix} \ell_1 & \cdots & \ell_n \end{bmatrix}$
return $\mathcal{L}$ and π

From lemma 3.1, we know that algorithm 1 samples an unbiased estimator of the product clique. Which leads to a factorization that is in expectation the original Laplacian, as was described in the following claim by [2]:

Claim 4.1 ([2, Claim 3.1])**.** *When* ApproximateCholesky *is instantiated with an unbiased clique-sampling routine* CliqueSample *whose output is positive semidefinite, its output*

$$\mathcal{L} = \textsc{ApproximateCholesky}(L)$$

satisfies

$$\mathbb{E}\Big[\mathcal{L}\mathcal{L}^\top\Big] = L.$$

5 Acknowledgements

We thank Anup Rao and David Durfee for helpful discussion and for telling us about a different algorithm for sampling a UST from a product clique. Their algorithm was presented to run in time $O(n^2)$, but can be easily implemented to run in time $O(n \log n)$.

ChatGPT 5.5 was used when developing some of the proofs in this article. Codex was used to help with assembling the latex file.

References

[1] Saverio Caminiti, Irene Finocchi, and Rossella Petreschi. On coding labeled trees. *Theoretical Computer Science*, 382(2):97–108, 2007. Latin American Theoretical Informatics. 4

[2] Yuan Gao, Rasmus Kyng, and Daniel A. Spielman. Ac(k): Robust solution of laplacian equations by randomized approximate cholesky factorization. *SIAM Journal on Scientific Computing*, 48(3):A1643–A1684, 2026. 1, 5

[3] Jonathan L Gross, Jay Yellen, and Mark Anderson. *Graph theory and its applications*. Chapman and Hall/CRC, 2018. 3

[4] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985. 1

[5] Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians - fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 573–582, 2016. 1, 5

[6] Daniel A Spielman. *Spectral and Algebraic Graph Theory*. 2025. 2, 3