Lecture 27: Fast Laplacian Solvers

Scribed by Eric Lee, Eston Schweickart, Chengrun Yang

November 21, 2017

1 How Fast Laplacian Solvers Work

We want to solve $Lx = b$ with L being a Laplacian matrix. Recall that $L\vec{1} = \vec{0}$ and $L \succeq 0$. If the original graph is connected (i.e., we can't partition L to be block diagonal), then L has exactly 1 zero eigenvalue, with corresponding eigenvector $\vec{1}$.

There are a number of ways to solve this problem, including iterative methods such as conjugate gradient (with modifications to account for the zero eigenvalue). However, Laplacian matrices have a well-studied structure that supports direct methods as well. Here, we introduce one such method, derived from [1]; though it is not the most asymptotically efficient method in the literature, it is relatively simple.

Notation: $A \succeq B$ means $A - B \succeq 0$.

To solve this problem "fast", we have the following steps:

(i) Compute a sparse, approximate Cholesky factorization $P\mathcal{L}D\mathcal{L}^T P^T$, in which P is a permutation matrix, $\mathcal{L}$ is lower triangular, D is diagonal. Furthermore, $\text{nnz}(\mathcal{L})$ is in the same order as $\text{nnz}(L)\log^3 n$, and if we let $Z = P\mathcal{L}D\mathcal{L}^T P^T$, $\frac{1}{2}L \preceq Z \preceq \frac{3}{2}L$.

(ii) Iterative refinement with $Z^\dagger$:

$$x^{(0)} = \vec{0}$$

$$x^{(i+1)} = x^{(i)} - \frac{1}{2}Z^\dagger(Lx^{(i)} - b)$$

This is similar to iterative refinement. Each iteration requires some sparse matrix-vector multiplications and a couple triangular solves, which can can be done in time proportional to the number of nonzeros in the matrix; overall, one iteration takes only $O(\text{nnz}(L)\log^3 n)$ computations, which is fast.

We can prove that after t iterations, we can get an ϵ-accurate solution, i.e. $||X^{(t)} - L^\dagger b||_L \leq \epsilon ||L^\dagger b||_L$ where $t \in O(\log(\frac{1}{\epsilon}))$.

In order to perform step (i), we will first consider the dense Cholesky factorization, and show how it can be modified in order to produce the decomposition specified.

2 Dense Cholesky Factorization

Let $S^{(0)} = L$. In the first step to solve $Lx = b$, we might want to eliminate variable 1 and get a smaller system $S^{(1)}x' = b'$, in which $b'(1) = 0$, and $x' = \begin{bmatrix} 0 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$.

Algebraically,

$$S^{(1)} \leftarrow L - \frac{1}{L(1,1)} L(:,1)L(:,1)^T = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \widetilde{S^{(1)}} & \\ 0 & & & \end{bmatrix}$$

where $\widetilde{S^{(1)}}$ is a Laplacian matrix of size $(n-1) \times (n-1)$, and is known as the *Schur complement*. Further, we abstract 1 to a vertex of the graph v_1 and define

$$\alpha_1 = L(v_1, v_1)$$

$$c_1 = \frac{1}{\alpha_1} L(:, v_1)$$

then $S^{(1)} = L - \alpha_1 c_1 c_1^T$.

Algorithm 1 Dense Cholesky Decomposition

Input matrix L
$S^{(0)} \leftarrow L$
for $k = 1, 2, \cdots, n-1$ **do**
    select a variable $v_k \in V \backslash \{v_1, \cdots, v_{k-1}\}$
    $\alpha_k \leftarrow S^{(k-1)}(v_k, v_k)$
    $c_k \leftarrow \frac{1}{\alpha_k} S^{(k-1)}(:, v_k)$
    $S^{(k)} \leftarrow S^{(k-1)} - \alpha_k c_k c_k^T$ #update
end for
Set $\alpha_n \leftarrow S^{(n-1)}(v_n, v_n)$, $c_n = e_{v_n}$
$C \leftarrow (c_1, c_2, \cdots, c_n)$
$D \leftarrow$ diagonal matrix with $D(i,i) = \alpha_i$ (Note: $L = \sum_{i=1}^n \alpha_i c_i c_i^T = CDC^T$)
$P \leftarrow$ permutation matrix, s.t. $Pe_i = e_{v_i}$
$\mathcal{L} \leftarrow P^T C$ (lower-triangular)
Output $L = P\mathcal{L}D\mathcal{L}^T P^T$, a Cholesky decomposition of L

3 Cholesky and Sparsity

In order to understand how to construct a sparse Cholesky decomposition, it is informative to first consider the dense Cholesky algorithm to identify where sparsity can be lost as the algorithm progresses. Consider the Laplacian of a star graph, that is, a graph where each vertex shares an edge with a single central vertex. If the central vertex is 1, that is, it corresponds to the first row and column of the Laplacian matrix, then the Laplacian matrix has a dagger structure, where the

only nonzeros reside on the diagonal, the first column, and the first row. For example, the sparsity pattern Laplacian of the 5-star graph is shown below:

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & & & \\ \times & & \times & & \\ \times & & & \times & \\ \times & & & & \times \end{bmatrix}$$

However, this sparsity is destroyed when constructing the Schur complement. Recall $S^{(1)} = L - \alpha_v c_v c_v^T$. The vector c_v contains an entry for every edge incident to v, so the low-rank matrix $c_v c_v^T$ contains the square of this number of entries. This implies that, even though $S^{(1)}$ will have zeros along the first column and row (eliminating $O(\deg(v))$ entries), subtracting L_v adds $O(\deg^2(v))$ additional entries. In the context of graph theory, this amounts to removing a vertex's edges from the graph, but forming a clique with each of its former neighbors. This can be disastrous in the case of a star graph structure. If the central vertex is chosen, $S^{(1)}$ becomes a dense matrix with zeros only in the first row and column, so the sparsity pattern is effectively destroyed:

$$\begin{bmatrix} & & & & \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & \times & \times & \times & \times \end{bmatrix}$$

Even for general graphs, computing $S^{(i)}$ for arbitrary i could potentially destroy any sparsity that was apparent in the original structure, and the problem compounds as the algorithm progresses. Some sparsity can be maintained by carefully choosing the variable v_k in step 4 of algorithm 1 such that not too many edges are added to the subsequent graph. For instance, in the case of a star graph, sparsity is maintained if the central vertex is chosen last. However, choosing the optimal ordering of variables is hard in general. Heuristics for variable selection exist, but they lack any guarantees on the level of sparsity maintained. We will now introduce an algorithm that sacrifices numerical accuracy to gain sparsity guarantees.

4 Sparse, Approximate Cholesky

The algorithm for the sparse, approximate Cholesky decomposition is quite similar to the standard Cholesky decomposition. However, sparse Cholesky seeks a sparse $\mathcal{L}$ instead of a dense one and is willing to incur some approximation error as a trade-off. In this light, the key difference between dense and sparse Cholesky is the change from a (possibly dense) Schur Complement (step 7 in Algorithm 1) at each step in the inner loop to a sparse approximation of the Schur Complement.

As a preprocessing step, the input graph is first converted to a *multigraph*, where pairs of vertices can share multiple edges. Any such multigraph's Laplacian can be represented by a sum of Laplacians, so this does not change the underlying problem.

Algorithm 2 Sparse Approximate Cholesky Decomposition

Input $L \in \mathbb{R}^{n \times n}, \epsilon, \delta$
$\hat{S}^{(0)} \leftarrow L$ with each edge e split into $\rho = \lceil 12(1+\delta)^2 \epsilon^{-2} log^2(n) \rceil$ sub-edges $\hat{e}_1 \dots \hat{e}_\rho$, with corresponding weights $w(\hat{e}_i) = \frac{1}{w(e)}$ for $i \in \{1, \dots, \rho\}$
Define $D \leftarrow \mathbf{0}_{n \times n}$ i.e. a diagonal matrix containing all zeros
Let π be a uniformly random permutation over $\{1, \dots, n\}$ and let P_π be its permutation matrix
for $k = 1, 2, \cdots, n-1$ **do**
    $D(\pi(k), \pi(k)) \leftarrow \hat{S}^{(k-1)}(\pi(k), \pi(k))$
    $c_k \leftarrow \hat{S}^{(k-1)}(:, \pi(k)) / D(\pi(k), \pi(k))$ if $D(\pi(k), \pi(k)) \neq 0$ or 0 otherwise
    $C_k \leftarrow cliquesample(\hat{S}^{(k-1)}, \pi(k))$
    $\hat{S}^{(k)} \leftarrow \hat{S}^{(k-1)} - \hat{S}^{(k-1)}_{\pi(k)} + C_k$ where $\hat{S}^{(k-1)}_{\pi(k)}$ is the Restricted Laplacian of $\hat{S}^{(k-1)}$
end for
$D \leftarrow \hat{S}^{(n)}$
$\mathcal{L} \leftarrow P_\pi^T (c_1, c_2, \cdots, c_n)$
Output $L \approx P_\pi \mathcal{L} D \mathcal{L}^T P_\pi^T$, a Cholesky decomposition of L

This algorithm is a bit hard to parse, but there are a few things to note.

- $\hat{S}$ is an approximate Schur Complement, and we expect it to be sparse.
- The Restricted Laplacian $S_{\pi(k)}$ of a matrix S removes the $\pi(k)$ row and column of S (the naming is self-evident).
- Finding an optimal ordering of nodes to minimize the number of nonzeros incurred in $\mathcal{L}$ is a known hard problem. As a result, we pick an arbitrary permutation π reordering the nodes uniformly at random so that, in expectation, we are picking some generic ordering of nodes.
- The function $cliquesample(S, k)$ returns i.i.d. samples of edges in an edge set that would have been introduced by eliminating vertex k from the graph S. More generally speaking, one may view $cliquesample(S, k)$ as a the method to construct a sparse Schur Complement via random sampling. We give the algorithm for $cliquesample(S, k)$ below:

Algorithm 3 Clique Sample

Input $S \in \mathbb{R}^{n \times n}, v$
for $i = 1, 2, \cdots, \deg_S(v)$ **do**
    sample e_1 from list of multi-edges on v with probability $w(e)/w_S(v)$
    sample e_2 from list of multi-edges incident on v uniformly at random
    if e_1 has endpoints v, u_1 and e_2 has endpoints v and u_2 respectively, $u_1 \neq u_2$ **then**
        $Y_i \leftarrow \frac{w(e_1)w(e_1)}{(w(e_1)+w(e_2))}(e_{u_2} - e_{u_1})(e_{u_2} - e_{u_1})^T$
    else
        $Y_i = \vec{0}$
    end if
end for
Output $\sum_i Y_i$

Note that $\deg_S(v) = \rho \deg_L(v)$. The parameters ϵ and δ control sampling probability of $cliquesample(S, k)$ by controlling the number of multi-edges created. Consequently, changing ϵ and δ allow one to change the quality of the approximation made by Algorithm 2. This can be made precise through the following theorem:

Theorem 4.1 *Given a connected multigraph $G = (V, E)$ with positive edge weights, graph Laplacian L, $\delta \geq 0$, and $0 < \epsilon \leq \frac{1}{2}$, the Sparse Cholesky algorithm returns a factorization $L \approx P\mathcal{L}D\mathcal{L}^T P^T$ with the property*

$$(1+\epsilon)L \succeq P\mathcal{L}D\mathcal{L}^T P^T \succeq (1-\epsilon)L$$

with probability at least

$$1 - \frac{2}{n^\delta}$$

with the expected number of nonzeros in $\mathcal{L}$ given as

$$nnz(\mathcal{L}) = \mathcal{O}(nnz(L)log^3(n))$$

and the expected runtime

$$\mathcal{O}\big(\frac{\delta^2}{\epsilon^2} nnz(L)log^3(n)\big)$$

Theorem 4.1 gives us bounds and expectations on the runtime, number of nonzeros, and quality of approximation given some ϵ and δ. The proof can be found in [1], but the key point of the proof is that $cliquesample(S, k)$ was constructed specifically to make expectations and probabilities work out nicely.

5 Further Thoughts not Discussed

The paper [1] introducing this fast laplacian solver was very theoretical in nature and did not provide any experimental results. Consequently, a few naturally arising computational questions were left (not necessarily difficult ones) unanswered. One might wonder why sparse and exact Cholesky factorization pivoting schemes were not pursued —are they slower? For example, Nested Dissection [3] is one traditional pivoting scheme used to minimize sparsity in $\mathcal{L}$ without accruing any approximation error, thus allowing for a direct solve via back-substitution. Because L is sparse, another interesting question is whether using Conjugate Gradient on the equation $Lx = b$, perhaps in tandem with the sparse approximate Cholesky factorization as a preconditioner, is faster or more robust compared to the solver in [1].

Currently, the fastest Laplacian solver is the algorithm introduced in [2], which operates in nearly $\mathcal{O}(nnz(L)\log^{\frac{1}{2}} n)$ time (up to polylog factors). However, the machinery behind this algorithm is far more complex than the solver in [1]. In short, the method recursively constructs preconditioners for the input Laplacian by looking at sparse subproblems, which involve a carefully chosen spanning tree and a small number of off-tree edges, weighted by a metric that is similar to the leverage score introduced in previous lectures.

References

[1] Rasmus Kyng and Sushant Sachdeva. *Approximate Gaussian Elimination for Laplacians: Fast, Sparse, and Simple.* 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2016.

[2] Cohen, Michael B., et al. *Solving SDD linear systems in nearly $m \log^{\frac{1}{2}} n$ time.* Proceedings of the forty-sixth annual ACM symposium on Theory of computing. ACM, 2014.

[3] George, Alan *Nested dissection of a regular finite element mesh.* SIAM Journal on Numerical Analysis 10.2 (1973): 345-363.