AN EXPLORATION OF NESTED DISSECTION FOR SPARSE LINEAR SYSTEMS

SIMON OPSAHL (SOPSAHL@MIT.EDU)*

Abstract. This work is a replication and analysis of six data reduction techniques used to improve the performance of METIS on the minimum fill-in problem. By relying on recent developments in graph partitioning theory, we improve upon the runtime and performance of a well-accepted standard for approximating the minimum fill-in ordering for Cholesky decomposition of highly sparse matrices. We evaluate fill-in, algorithm runtime, operation count, and volume of the graph reduction across five different data reduction strategies.

Key words. METIS, Graph Partitioning, Nested Dissection, Sparse Linear Solvers, Cholesky Factorization

1. Introduction. In linear systems, there is a need for efficient solvers of the $Ax = b$ problem. Cholesky decomposition ($A = LL^*$) is one such solution for Hermitian positive-definite matrices. In this decomposition, you turn the full matrix into its lower triangular equivalent in $O(\frac{n^3}{3})$ [11]. The linear system can then be solved by a forward computation of $Ly = b$ and then a backwards computation of $L^*x = y$, both $O(n^2)$ because of the triangularity of the matrix.

One class of linear systems that requires special consideration is graph-based linear systems like road networks and social networks. These systems often have very large n and are highly sparse. With a sparse A, both complexities of the Cholesky decomposition and computation may drop, but such improvement is heavily dependent on the sparsity structure [11]. In addition, Cholesky decomposition may naively introduce additional non-zero elements on any step of the elimination process. In the worst-case, this can result in a fully dense L as a decomposition of a highly sparse A.

It has been shown that for each sparse matrix A, there exists an optimal ordering of rows or columns such that $PAP^T = LL^*$ minimizes the fill-in. The search for this optimal P is NP-complete, however, and thus there exists the need for efficient and accurate approximations of the optimal ordering [13]. Fortunately, there has been a wide corpus of research into graph partitioning algorithms. If you view each row/column as a node, and each non-zero element as an undirected edge, then the optimal permutation in the minimum fill-in problem is a direct analogue to the optimal elimination ordering in a undirected graph. When a node is eliminated, an edge is added between its neighbors to form a clique. The property you minimize over is the number of edges added to the graph.

As the minimal fill-in problem is NP-complete, there have been a host of accurate heuristics used to create an elimination ordering. Developments began with the *minimum degree algorithm*, where the node with the minimum degree is eliminated on each step [11, 1]. The computational cost of such an algorithm is high, because there is a lot of time spent on updating node degree. As a result, newer algorithms were developed. George et al. introduced the concept of *nested dissection*, where a node separator splits the graph into subgraphs that are then recursively partitioned [2, 6]. The algorithm still makes use of the minimum degree heuristic when the subgraphs are small [10, 3].

Since the initial research into nested dissection, there have been a number of improvements in the algorithm to increase parallelism [8, 5]. The current state of the

*MIT, Cambridge, MA. 18.335 SP25: Intro to Numerical Methods.

art is METIS, introduced in 1998 by Karypis and Kumar [5], which uses a multilevel approach. In this work, we use the PyMetis [7] Python package developed in 2022.

Not much has changed in the field of nested dissection and solutions for the minimum fill-in problem since the development of METIS, even though there have been a number of improvements in graph partitioning techniques [10]. In this work, we aim to replicate Ost et al. They utilize data reduction techniques found in graph partitioning theory to reduce the graph that METIS operates on, improving both the runtime and fill-in of the overall ordering.

2. Algorithm. Ost et al. [10] introduces five data reduction steps that can improve the overall speed and fill-in of METIS. The steps are meant to take advantage of exact eliminations, exact contractions, inexact eliminations, and inexact contractions in order to reduce the size of the graph prior to METIS partitioning. By reducing the size of the graph, the work that is done by METIS is reduced, and thus inclusion of a collection of the proposed preprocessing steps is shown to lead to an improvement in overall runtime of the algorithm. Improvement in fill-in is likely a result of the reliability of the data reduction rules that are chosen.

For the data reduction rules, we rely on the following definitions given a graph $G = (V, E)$:

- The open neighborhood ($\Gamma(u)$) of a node $u \in V$ is defined as the set of neighbors of u. In other words, $\Gamma(u) \coloneqq \{v : (u, v) \in E\}$.
- The closed neighborhood ($\Gamma[u]$) of a node $u \in V$ is defined as the set of neighbors of u along with u itself. In other words, $\Gamma[u] \coloneqq \{v : (u, v) \in E\} \cup \{u\}$.
- A clique is defined as any set of nodes $C \subseteq V$ in which there exists an edge between every node. In other words, $\forall u, v \in C, (u, v) \in E$.
- The degree of a node ($\deg(u)$) is defined as the number of neighbors. In other words, $\deg(u) \coloneqq |\Gamma(u)|$.
- A subgraph G_W of a set of vertices $W \subseteq V$ is defined as the set of nodes in W paired with the set of edges between member nodes of W. In other words, $G_W = (W, E_W)$, where $E_W \coloneqq \{(u, v) : u, v \in W \wedge (u, v) \in E\}$.
- For a node u to be *eliminated*, it is added to the elimination order σ, then all of its edges are removed from the graph. In addition, edges are added to the graph until the neighbors of u form a clique. The number of edges added is the fill-in. In other words, $\sigma \leftarrow \sigma u$ and $G \leftarrow (V \setminus \{u\}, E \cup \{(v, w) : v, w \in \Gamma(u)\} \setminus \{(u, v) : v \in \Gamma(u)\}$.
- For a set of nodes $W \subseteq V$ to be *contracted*, all nodes in W are reduced to one node w, and all outgoing edges from G_W are now edges of w. In other words, $G \leftarrow (V \cup \{w\} \setminus W, E \cup \{(w, v) : v \notin W \wedge \exists u \in W, (u, v) \in E\} \setminus E_W)$. In addition, when constructing the final elimination ordering, nodes that are contracted can be eliminated together.

2.1. Simplicial Reduction. Simplicial reduction is an exact elimination rule that relies on the fact that any node u whose open neighborhood is a clique can be eliminated first. In practice, this can be a very costly operation, as iterating through the set of all pairs of neighbors of all nodes is upper bounded by $O(n^3)$. As a result, Ost et al. enforce a degree threshold Δ. When scanning through the set of nodes, we only check for cliques if the degree of u is less than or equal to Δ. This significantly reduces the time complexity to $O(n\Delta^2)$. This rule is exact because elimination of any node with a neighborhood clique results in zero fill-in. As a result, there exists a minimal elimination ordering where u is eliminated first. The implementation can be

found in Algorithm 2.1.

Algorithm 2.1 Simplicial Reduction

```
Graph: G = (V, E)
Degree Threshold: Δ
for all u ∈ V do
  if deg(u) ≤ Δ then
    Given the subgraph G_Γ(u)
    if |E_Γ(u)| == deg(u)·(deg(u) − 1)/2 then
      Eliminate u
    end if
  end if
end for
```

2.2. Indistinguishable Reduction. Indistinguishable reduction contracts two nodes u and v if $\Gamma[u] = \Gamma[v]$. This reduction relies on the observation that u and v are essentially indistinguishable in an elimination ordering. As a result, they can be eliminated together without penalty. In practice, you can save unnecessary comparisons between neighborhoods by constructing a hash of u equal to the sum of some function $f(v)$ over $v \in \Gamma[u]$. This requires iterating through the m edges. Then, you only compare neighborhoods of two nodes if their hashes are equal. In the worst case this is $O(m + m \cdot \deg(v))$. An implementation can be found in Algorithm 2.2.

Algorithm 2.2 Indistinguishable Reduction

```
Graph: G = (V, E)
Initialize H
for all u ∈ V do
  H[u] ← Σ_{v∈Γ[u]} f(v)
end for
for all (u, v) ∈ E do
  if H[u] = H[v] then
    if Γ[u] = Γ[v] then
      Contract u and v into new node w
      Update G
    end if
  end if
end for
```

2.3. Twin Reduction. Twin reduction is an exact data reduction technique that relies on a very similar observation to indistinguishable reduction. Namely, a set of twins, or nodes that have the same open neighborhood, can be eliminated together in an optimal ordering. The cost of creating a clique among the neighbors of a node is only paid by one twin. Afterwards, the twin is simplicial and can be immediately eliminated. In practice, we must construct a hash table just as with indistinguishable reduction. Then, sort the hash and compare adjacent entries. If the hash and degree are identical, then check the neighbors. This is in the worst case the cost of identical degree and hash, or $O(mn)$. An implementation can be found in Algorithm 2.3.

Algorithm 2.3 Twin Reduction

```
Graph: G = (V, E)
Initialize H
for all u ∈ V do
  H[u] ← Σ_{v∈Γ(u)} hash(v)
end for
Sort H → H'
for h_i, h_{i+1} ∈ H' do
  if deg(h_i) = deg(h_{i+1}) and H'[h_i] = H'[h_{i+1}] then
    if Γ(h_i) = Γ(h_{i+1}) then
      Contract h_i and h_{i+1} into new node w
      Update G
    end if
  end if
end for
```

2.4. Path Compression. Path compression is an exact data reduction technique relying on the observation that any path of degree-2 nodes can be eliminated together. The order of elimination depends on which neighbor is eliminated first, however. In practice, you can contract nodes by iterating over the adjacent nodes until a degree-2 node is not encountered. Because this requires walking each edge at most once, path compression is upper-bounded by $O(m)$. An implementation can be found in Algorithm 2.4.

Algorithm 2.4 Path Compression

```
Graph: G = (V, E)
Initialize R ← ∅
for all u ∈ V do
  if deg(u) = 2 and u ∉ R then
    Initialize P ← [u]
    Initialize S = Γ(u)
    while S do
      x = S.pop()
      if deg(x) = 2 and x ∉ R then
        P ← P + [x]
        S ← S ∪ Γ(x)
      end if
    end while
    if |P| > 1 then
      Contract P
    end if
    R ← R ∪ P
  end if
end for
```

2.5. Degree 2 Elimination. Degree-2 elimination is an inexact reduction rule that relies on the fact that eliminating a degree-2 node is exact if that node is not a separator, or is a part of a cycle. Empirically found to be worthwhile, degree-2

elimination can approximate the minimum ordering of G assuming the majority of nodes reside in cycles, and the penalty if not is low. The cost of degree-2 elimination is just the cost of iterating through the nodes until a degree-2 node is not encountered. As a result, it is upper bounded by $O(n^2)$. An implementation can be found in Algorithm 2.5.

Algorithm 2.5 Degree-2 Elimination

```
Graph: G = (V, E)
for all u ∈ V do
  if deg(u) = 2 then
    Eliminate u
    Restart 2.4
  end if
end for
```

2.6. Triangle Contraction. Triangle contraction is an inexact reduction rule that relies on the fact that if there are two adjacent nodes u and v both with degree 3 and $|\Gamma(u) \cap \Gamma(v)| \geq 1$, then eliminating u and v results in a fill-in of only two edges. In practice, this requires walking an edge at most once, and thus we are upper bounded by $O(m)$. The implementation can be found in Algorithm 2.6.

Algorithm 2.6 Triangle Contraction

```
Graph: G = (V, E)
Initialize R ← ∅
for all u ∈ V do
  if deg(u) = 3 and u ∉ R then
    Initialize P ← {u}, S ← [u]
    while S do
      x ← S.pop()
      if deg(x) = 3 and x ∉ R then
        R ← R ∪ {x}
        P ← P ∪ {x}
        for all y ∈ Γ(x) do
          if deg(y) = 3 and y ∉ R and |Γ(x) ∩ Γ(y)| ≥ 1 then
            S ← S + [y]
          end if
        end for
      end if
    end while
    if |P| > 1 then
      Contract P
    end if
  end if
end for
```

3. Methods. In our methodology, we use Python to write and profile all of the data reduction techniques. The testing pipeline is as follows:

3.1. Load the graph. We tested on three large road networks from the SNAP dataset [9] because Ost et al. reported their highest improvements on road networks [10]. We relied on networkx [4] for graph storage and manipulation. The three road networks tested are as follows: roadNet-TX, roadNet-CA, and roadNet-PA.

3.2. Transform the graph. We then transformed the graph with a combination of any of the six data reduction techniques described in section 2. Within this step, we recorded the number of reductions each technique contributed to the overall total. In addition, we recorded a rough estimate of the operational cost of each reduction dependent on the degree of nodes and the loop ordering. In our experiments, we tested the following five preprocessing algorithms to replicate results from Ost et al. [10]:

- METIS: No preprocessing.
- SITP12: Simplicial reduction with a degree threshold of 12, followed by indistinguishable and twin reductions, followed by path compression.
- SIDTr12: Simplicial reduction with a degree threshold of 12, followed by indistinguishable reduction, followed by degree-2 elimination, followed by triangle contraction.
- SITD6: Simplicial reduction with a degree threshold of 6, followed by indistinguishable and twin reductions, followed by degree-2 elimination.
- SD18: Simplicial reduction with a degree threshold of 18 followed by degree-2 elimination.

3.3. Run METIS. We then used the reduced graph as input to a nested dissection algorithm from PyMetis [7]. We computed the elimination order and profiled the runtime of METIS over $n = 10$ iterations of the algorithm. The elimination order of the reduced graph was then used to find the overall elimination order of the original graph by combining the eliminated nodes and expanding the contracted nodes when appropriate. For all but path compression, contracted nodes could be expanded as is, but nodes reduced by path compression were expanded in order of which neighbor appeared first in the elimination order.

3.4. Measure Fill-in. Finally, we measured the fill-in of the Cholesky decomposition. We used the Laplacian matrix of the graph permuted by the elimination order. Then, we used the Cholesky factorization from scikit-sparse [12] to find L. Lastly, we computed fill-in as the number of nonzeros in L compared to the Laplacian.

4. Results. Our results replicated the results found in Ost el al. [10]. We found that as the number of reductions increased, the overall runtime of METIS decreased, as seen in Figure 4.1. Note that the cost of METIS runtime does not account for the data reduction cost, but the hope is that optimized kernels may be able to take advantage of the up to 50% speedups we encountered.

We also found that degree-2 elimination and simplicial reduction shared the largest fraction of reductions among all preprocessing steps, as seen in Figure 4.2. Interestingly, indistinguishable and twin reductions did not significantly contribute to the overall number of reductions, even though their operation count was a sizeable portion of the total estimated operations.

Finally, we found that the fill-in of each of the preprocessing algorithms was up to 2% less than METIS in all but one case, as seen in Figure 4.4. This is consistent with the findings of Ost et al.

5. Discussion. Ultimately, the goal of this review was to recreate the results from Ost et al. [10]. This goal was accomplished. We were able to show that

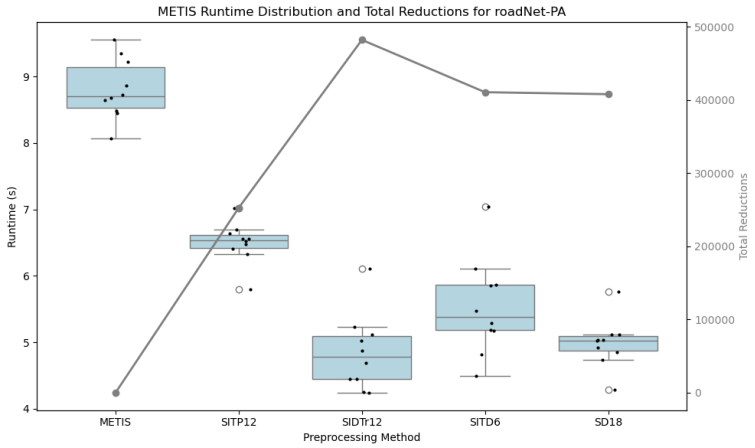

FIG. 4.1. *METIS runtime over each preprocessing method for roadNet-PA.*

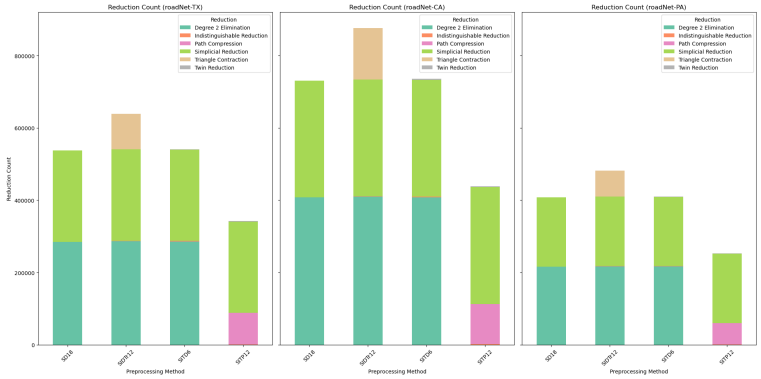

FIG. 4.2. *Reduction count for each data reduction technique over each preprocessing method for each test graph.*

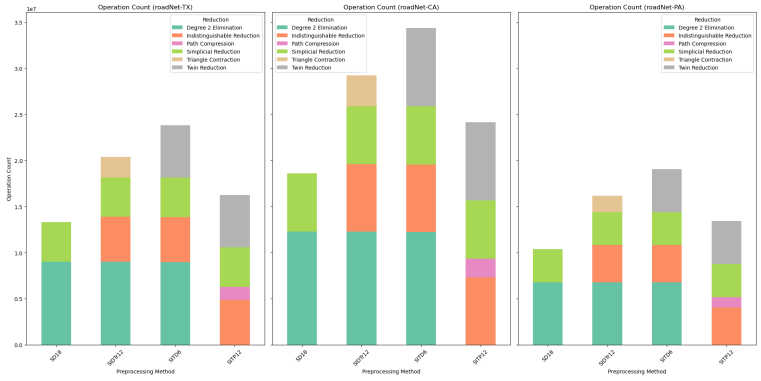

FIG. 4.3. *Operation count for each data reduction technique over each preprocessing method for each test graph.*

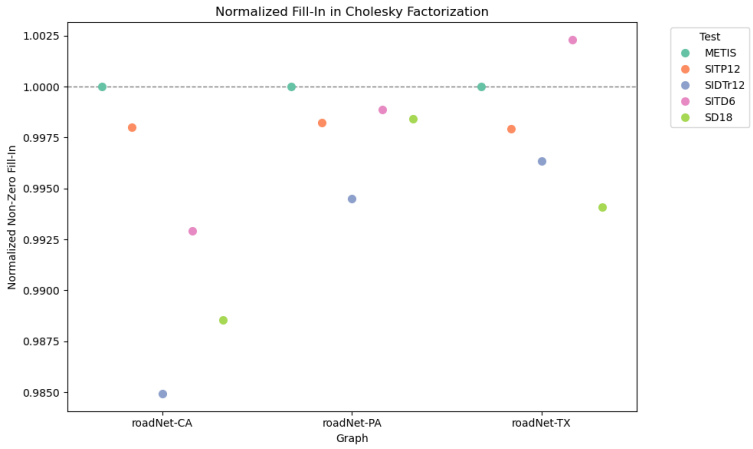

FIG. 4.4. *Normalized fill-in for each preprocessing method for each test graph.*

preprocessing can lead to improvements in METIS performance and runtime. As a result, it is clear that current implementations of nested dissection have much room to improve in light of recent advances in graph partitioning. The minimum fill-in problem remains highly relevant, as computations over sparse networks are found in a number of domains, including modern graph neural networks [14]. Additional work is required to explore this domain and discover additional improvements, many of which may already exist in graph theory.

REFERENCES

[1] C. ASHCRAFT, *Compressed graphs and the minimum degree algorithm*, SIAM Journal on Scientific Computing, 16 (1995), pp. 1404–1411, https://doi.org/10.1137/0916081.

[2] A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363, https://doi.org/10.1137/0710032.

[3] A. GEORGE AND J. W. H. LIU, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1–19, https://doi.org/10.1137/1031001.

[4] A. A. HAGBERG, D. A. SCHULT, AND P. J. SWART, *Exploring network structure, dynamics, and function using networkx*, in Proceedings of the Python in Science Conference, Pasadena, California, June 2008, pp. 11–15, https://doi.org/10.25080/TCWV9851.

[5] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392, https://doi.org/10.1137/S1064827595287997.

[6] M. S. KHAIRA, G. L. MILLER, AND T. J. SHEFFLER, *Nested dissection: A survey and comparison of various nested dissection algorithms*. Unpublished manuscript or preprint.

[7] A. KLOECKNER, M. WALA, N. HARTLAND, A. DANIAL, F. OBERMEYER, C. WU, AND C. GOHLKE, *Pymetis*, July 2022, https://doi.org/10.5281/zenodo.6892213, https://github.com/inducer/pymetis. MIT License.

[8] D. LASALLE AND G. KARYPIS, *Efficient nested dissection for multicore architectures: 21st international conference on parallel and distributed computing, euro-par 2015*, in Euro-Par 2015, 2015, pp. 467–478, https://doi.org/10.1007/978-3-662-48096-0_36.

[9] J. LESKOVEC AND A. KREVL, *SNAP Datasets: Stanford large network dataset collection*. http://snap.stanford.edu/data, June 2014.

[10] L. OST, C. SCHULZ, AND D. STRASH, *Engineering data reduction for nested dissection*, Apr. 2020, https://doi.org/10.48550/arXiv.2004.11315, https://arxiv.org/abs/2004.11315.

[11] W. F. TINNEY AND J. W. WALKER, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proceedings of the IEEE, 55 (1967), pp. 1801–1809, https://doi.org/10.1109/PROC.1967.6011.

[12] P. VIRTANEN, R. GOMMERS, T. E. OLIPHANT, M. HABERLAND, T. REDDY, D. COURNAPEAU,

E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods, 17 (2020), pp. 261–272, https://doi.org/10.1038/s41592-019-0686-2.

[13] M. Yannakakis, *Computing the minimum fill-in is np-complete*, SIAM Journal on Algebraic and Discrete Methods, 2 (1981), pp. 77–79, https://doi.org/10.1137/0602010.

[14] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, *Graph neural networks: A review of methods and applications*, arXiv preprint arXiv:1812.08434, (2021), https://doi.org/10.48550/arXiv.1812.08434, https://arxiv.org/abs/1812.08434.

Appendix. All code and visualization can be found at https://github.com/sopsahl/preMETIS. A README in the top level describes use of the scripts and profiling tools.