LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-683260

Solving Graph Laplacian Systems Through Recursive Bisections and Two-Grid Preconditioning

C. Ponce, P. S. Vassilevski

February 18, 2016

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

SOLVING GRAPH LAPLACIAN SYSTEMS THROUGH RECURSIVE BISECTIONS AND TWO-GRID PRECONDITIONING

COLIN PONCE[†] AND PANAYOT S. VASSILEVSKI[‡]

Abstract.
We present a parallelizable direct method for computing the solution to graph Laplacian-based linear systems derived from graphs that can be hierarchically bipartitioned with small edge cuts. For a graph of size n with constant-size edge cuts, our method decomposes a graph Laplacian in time $O(n \log n)$, and then uses that decomposition to perform a linear solve in time $O(n \log n)$.

We then use the developed technique to design a preconditioner for graph Laplacians that do not have this property. Finally, we augment this preconditioner with a two-grid method that accounts for much of the preconditioner's weaknesses. We present an analysis of this method, as well as a general theorem for the condition number of a general class of two-grid support graph-based preconditioners. Numerical experiments illustrate the performance of the studied methods.

Key words. graph Laplacian, recursive bisection, support graph preconditioners, two-grid methods.

1. Introduction. Recent years have seen an explosion of interest in studying large networks. The graphs representing these networks often reach hundreds of millions or billions of nodes; as a result, only the most scalable of algorithms are practical in network analysis.

One problem often of interest is solving linear systems based on graph Laplacians. Graph Laplacians appear in many areas, such as information dispersal through social networks or electricity flow through resistor networks. When graphs are large, general sparse linear solvers are not typically fast enough to be feasible. Thus, we must develop specialized graph Laplacian linear solvers.

In this paper we first consider graphs that can be hierarchically bipartitioned with small edge cuts. This graph structure appears in networks in which a small number of "high-bandwidth" interconnects stretch between distant clusters of nodes. For example, transportation systems and wide area computer networks often have this property.

Our method takes a two-step approach: first, perform a fast computation that results in highly structured error; second, exploit that structure to make error correction cheap. In our case, we recursively solve linear systems over each of the isolated graph partitions, ignoring the edge cuts. This results in an error vector that lies in a low-dimensional subspace determined by the edge cut. We then correct the error using the Sherman-Morrison-Woodbury formula.

We then extend this method to develop a preconditioner for graph Laplacians that lack this property. We recursively partition the graph and remove between-partition edges as needed until we obtain a *support graph* that has the desired property. The solution of linear systems of the graph Laplacian of this support graph acts as our preconditioner.

Finally, we accelerate this preconditioner using two-grid techniques. A simple node aggregation technique leads to a coarse basis whose span often approximately captures the error modes that most damage the condition number of the preconditioned system. We develop a formula for the condition number of two-grid support graph preconditioned systems, and use this to develop guidelines for support graph creation.

Our direct solver is similar in spirit to nested dissection [7, 11]. This technique is also a direct solution method for linear systems based on graphs. It recursively splits the graph into two roughly equal sets by finding a separating set of nodes. Ordering the system matrix according to this recursive partitioning leads to a fast method for such linear systems. While our method also uses a recursive partitioning of a graph, it is based on edge separators instead of node separators.

[†]Department of Computer Science, Cornell University, New York, USA, (cponce@cs.cornell.edu).

[‡]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, California, USA, (panayot@llnl.gov).

oThis research was conducted with Government support under and awarded by DoD, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program.

Support graph preconditioning has grown in popularity since Spielman and Teng's seminal work on using ultra-sparsifiers to create support graphs with good conditioning [12]. Since its publication, various authors have improved on the graph-theoretic algorithms underlying the preconditioner, thus improving its theoretical complexity [9].

While these papers develop algorithms with good theoretical complexity, practical implementations of these methods do not exist. In this paper we take the approach of developing implementable algorithms and demonstrate their performance on graph Laplacian systems derived from both synthetic and real-world graphs.

A key element of our preconditioner is the use of coarse-grid corrections. Multigrid algorithms were first developed as effective preconditioners for large finite element problems on geometric meshes. Since its introduction in [4], algebraic multigrid methods (AMG) have adapted the original multigrid scheme for use in linear systems with no underlying geometry.

The rest of the paper is organized as follows. Section 2 defines the graph Laplacian and the problem we solve. Section 3 derives our direct hierarchical method. Section 4 proves the complexity requirements for the algorithm. Section 5 shows how this method can be used to construct a support graph preconditioner, and Section 6 improves this preconditioner using a two-grid approach. Finally, Section 7 shows experimental results, and Section 8 concludes.

2. Problem Statement. Suppose we have a graph $G = (V, E)$, where V is a set of vertices, and E is a set of edges (pairs of vertices $(u, v) \in V \times V$), with each edge having an associated weight w_{uv}. We consider undirected graphs, which means that if $(u, v) \in E$, then $(v, u) \in E$ with $w_{vu} = w_{uv}$. In what follows, we assume that G is a connected graph; that is, any two vertices u and u' can be connected by a path of edges $(u_{s-1}, u_s) \in E$, $s = 1, 2, \ldots, m = m(u, u')$, where $u_0 = u$ and $u_m = u'$. Another notation that we use in what follows is $\mathbf{1} = (1)$ being the constant vector with unit entries; also the ith unit coordinate vector is denoted by $\boldsymbol{e}_i$.

The *graph Laplacian* is a matrix representation of an undirected graph. It is defined as follows:

$$\begin{aligned} L_{uv} &= \begin{cases} -w_{uv} & (u,v) \in E \\ 0 & \text{otherwise} \end{cases} \\ L_{uu} &= \sum_{v:(u,v)\in E} w_{uv} \end{aligned} \tag{2.1}$$

Note that $L = L^T$ and the sum of each row and each column of L is 0, and so $L\mathbf{1} = L^T\mathbf{1} = \mathbf{0}$.

Thus, the problem of interest is to solve the linear system

$$L\boldsymbol{x} = \boldsymbol{b} \tag{2.2}$$

for $\boldsymbol{x}$, where $\mathbf{1}^T\boldsymbol{b} = 0$.

In Section 3, we focus on connected graphs that have a p-cut of η edges, with each component $G_1, ..., G_p$ of roughly equal size. Let $V_1, ..., V_p$ denote the vertices of the components, and $n_i = |V_i|$. Let C denote the set of edges of the cut, so that $\eta = |C|$.

3. Derivation.

3.1. A Two-Level Hierarchy. The graph Laplacian is a singular matrix. First, we modify the problem to an equivalent but invertible one.

LEMMA 3.1. *Given a connected graph Laplacian L, some non-zero vector $\boldsymbol{e}$ with non-negative entries, and some $w > 0$, let*

$$\mathcal{L} = L + w\boldsymbol{e}\boldsymbol{e}^T. \tag{3.1}$$

Then

1. *$\mathcal{L}$ is invertible.*
2. *$\mathcal{L}$ is positive-definite.*

3. If $\mathbf{1}^T\boldsymbol{b} = 0$ and $\mathcal{L}\boldsymbol{x} = \boldsymbol{b}$, then $L\boldsymbol{x} = \boldsymbol{b}$ as well.

Proof. To prove (1), suppose $\mathcal{L}$ is singular. Then there exists a vector $\boldsymbol{x}$ such that

$$L\boldsymbol{x} = -w\boldsymbol{e}\boldsymbol{e}^T\boldsymbol{x}.$$

The only null vector of the left-hand side of this equation is $\mathbf{1}$, which is not a null vector of the right-hand side, so $L\boldsymbol{x} \neq \mathbf{0}$. So, for the above relation to hold, we must have $L\boldsymbol{x} \propto \boldsymbol{e}$. But $\boldsymbol{e} \notin \text{range}(L)$, as it is not orthogonal to $\mathbf{1}$. Thus, there is no such vector $\boldsymbol{x}$.

To show (2), Note that

$$\boldsymbol{x}^T\mathcal{L}\boldsymbol{x} = \boldsymbol{x}^T L\boldsymbol{x} + w(\boldsymbol{e}^T\boldsymbol{x})^2.$$

Both terms on the right-hand side are at least zero, so $\mathcal{L}$ is positive semidefinite. By (1), $\mathcal{L}$ is invertible, and so is positive definite.

To prove (3), note that

$$0 = \mathbf{1}^T\boldsymbol{b} = \mathbf{1}^T\left(L + w\boldsymbol{e}\boldsymbol{e}^T\right)\boldsymbol{x} = \mathbf{0}^T\boldsymbol{x} + w(\mathbf{1}^T\boldsymbol{e})(\boldsymbol{e}^T\boldsymbol{x}).$$

The factor $\mathbf{1}^T\boldsymbol{e} \neq 0$, so the above relation holds if and only if $\boldsymbol{e}^T\boldsymbol{x} = 0$. Therefore,

$$L\boldsymbol{x} = \mathcal{L}\boldsymbol{x} = \boldsymbol{b}.$$

□

Consider the block-diagonal matrix

$$\hat{L} = \begin{bmatrix} L_1 & & \\ & \ddots & \\ & & L_p \end{bmatrix}. \tag{3.2}$$

Then L has the form

$$L = \hat{L} + \sum_{(u,v)\in C} w_{uv}\boldsymbol{d}_{uv}\boldsymbol{d}_{uv}^T, \tag{3.3}$$

where $\boldsymbol{d}_{uv} = \boldsymbol{e}_u - \boldsymbol{e}_v$. Note that for $\mathcal{L} = L + w\boldsymbol{e}_{u_p}\boldsymbol{e}_{u_p}^T$, we have

$$\mathcal{L} = \hat{L} + w\boldsymbol{e}_{u_p}\boldsymbol{e}_{u_p}^T + \sum_{(u,v)\in C} w_{uv}\boldsymbol{d}_{uv}\boldsymbol{d}_{uv}^T.$$

We choose $w = 1$.

Now, we wish to solve for $\boldsymbol{x}$ in Equation (2.2). We will do this by first solving a set of smaller systems of equations on each of the $L_1, ..., L_p$. But we want each of these sub-systems to be invertible as well. So, for each $j = 1, ..., p$, let u_j denote the index of a vertex in V_j. Then

$$\begin{aligned} \mathcal{L} &= (\hat{L} + \sum_{j=1}^{p} \boldsymbol{e}_{u_j}\boldsymbol{e}_{u_j}^T) - \sum_{j=1}^{p-1} \boldsymbol{e}_{u_j}\boldsymbol{e}_{u_j}^T + \sum_{(u,v)\in C} w_{uv}\boldsymbol{d}_{uv}\boldsymbol{d}_{uv}^T \\ &= \hat{\mathcal{L}} \quad - \sum_{j=1}^{p-1} \boldsymbol{e}_{u_j}\boldsymbol{e}_{u_j}^T + \sum_{(u,v)\in C} w_{uv}\boldsymbol{d}_{uv}\boldsymbol{d}_{uv}^T \\ &= \hat{\mathcal{L}} \quad + F^T W F, \end{aligned} \tag{3.4}$$

where

$$F = [E\ D] \tag{3.5}$$
$$E = \left[\boldsymbol{e}_{u_1}\ \cdots\ \boldsymbol{e}_{u_{p-1}}\right] \tag{3.6}$$
$$D = [\cdots\ \boldsymbol{d}_{uv}\cdots]_{(u,v)\in C} \tag{3.7}$$
$$W = \text{diag}(-T, S) \tag{3.8}$$
$$T = \text{diag}(\mathbf{1}) \tag{3.9}$$
$$S = \text{diag}(\cdots, w_{u,v}, \cdots). \tag{3.10}$$

Note that the matrix $\hat{\mathcal{L}}$ is block-diagonal of the form

$$\hat{\mathcal{L}} = \begin{bmatrix} L_1 + \boldsymbol{e}_{u_1}\boldsymbol{e}_{u_1}^T & & \\ & \ddots & \\ & & L_p + \boldsymbol{e}_{u_p}\boldsymbol{e}_{u_p}^T \end{bmatrix},$$

where each block has the same form as in Equation (3.1). Hence each block of $\hat{\mathcal{L}}$ is itself an invertible matrix, by Lemma 3.1, so $\hat{\mathcal{L}}$ is invertible.

LEMMA 3.2. *The matrix*

$$-T^{-1} + E^T\left(\hat{\mathcal{L}} + DSD^T\right)^{-1}E \tag{3.11}$$

is symmetric negative-definite.

Proof. As shown above,

$$\mathcal{L} = \hat{\mathcal{L}} + D^TSD - E^TTE$$

is symmetric positive-definite. Then so is

$$\left(\hat{\mathcal{L}} + D^TSD\right)^{-1/2}\mathcal{L}\left(\hat{\mathcal{L}} + D^TSD\right)^{-1/2} = I - RR^T$$

where

$$R = \left(\hat{\mathcal{L}} + D^TSD\right)^{-1/2}E^TT^{1/2}.$$

Then

$$I - R^TR = I - T^{1/2}E^T\left(\hat{\mathcal{L}} + D^TSD\right)^{-1}ET^{1/2}$$

is also symmetric positive definite. Pre- and post-multiplication by $T^{-1/2}$ proves the result. ∎

THEOREM 3.3.

$$\mathcal{L}^{-1} = \hat{\mathcal{L}}^{-1} + \hat{\mathcal{L}}^{-1}F\left(W^{-1} + F^T\hat{\mathcal{L}}^{-1}F\right)^{-1}F^T\hat{\mathcal{L}}^{-1}. \tag{3.12}$$

Proof. We must show that $W^{-1} + F^T\hat{\mathcal{L}}^{-}F$ is invertible. Note that

$$W^{-1} + F^T\hat{\mathcal{L}}^{-1}F = \begin{bmatrix} -T^{-1} + E^T\hat{\mathcal{L}}^{-1}E & E^T\hat{\mathcal{L}}^{-1}D \\ D^T\hat{\mathcal{L}}^{-1}E & S^{-1} + D^T\hat{\mathcal{L}}^{-1}D \end{bmatrix}. \tag{3.13}$$

The lower-right block is symmetric positive-definite, hence invertible. The respective Schur complement is

$$-T^{-1} + E^T\left(\hat{\mathcal{L}}^{-1} - \hat{\mathcal{L}}^{-1}D\left(S^{-1} + D^T\hat{\mathcal{L}}^{-1}D\right)^{-1}D^T\hat{\mathcal{L}}^{-1}\right)E.$$

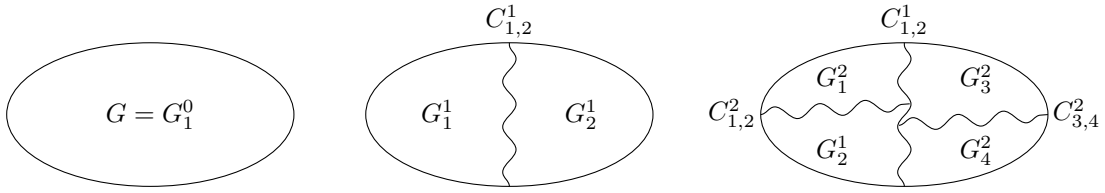


FIG. 3.1. *Three different views of the hierarchically decomposed graph G. Here $G = (V, E)$ is decomposed as $G = (V_1^1 \cup V_2^2, E_1^1 \cup E_2^1 \cup C_{1,2}^1)$, and $G_1^1 = (V_1^1, E_1^1)$ and $G_2^1 = (V_2^1, E_2^1)$ are decomposed similarly.*

Note that $\hat{\mathcal{L}}, S$, and $S^{-1} + D^T\hat{\mathcal{L}}^{-1}D$ are all invertible. We may therefore apply the Sherman-Morrison-Woodbury formula (see, e.g. Proposition 3.5 of [13]) to rewrite the above as

$$-T^{-1} + E^T\left(\hat{\mathcal{L}} + DSD^T\right)^{-1}E.$$

Lemma 3.2 shows us that this matrix is negative definite and therefore invertible.

Thus, both the lower-right block of Equation (3.13) and its Schur complement are invertible, proving that $W^{-1} + F^T\hat{\mathcal{L}}^{-1}F$ is invertible. Therefore, we may again apply the Sherman-Morrison-Woodbury formula to Equation (3.4) to obtain the result. □

3.2. Multilevel Hierarchies. We can modify this decomposition to allow it to handle much larger cuts by imposing some extra restrictions on the structure of the cut. In particular, suppose the graph can be recursively cut into p components such that each p-cut consists of at most η edges. Then we may apply our decomposition recursively, decomposing each sub-problem using another call to the decompose function.

This creates a hierarchy of subgraphs. Let G_i^ν denote subgraph i at hierarchy level ν, for $\nu = 0, ..., \nu_M$, where $\nu_M \simeq O(\log n)$. So $G = G_1^0$, and in Equaton 3.2, each L_i refers to G_i^1. We may globally index all level-ν subgraphs as $G_1^\nu, ..., G_{p^\nu}^\nu$, or we may refer to the children of G_i^ν as $G_{i,1}^\nu, ..., G_{i,p}^\nu$. See Figure 3.1 for a diagram of this hierarchy.

Define graph Laplacians L_i^ν, vertex sets V_i^ν, subgraph edge sets E_i^ν similarly. We define $C_{i,j}^\nu$ to be the level-ν edge cut between G_i^ν and G_j^ν, or C^ν to be the total level-ν edge cut.

Algorithms for graph decomposition as well as solution are shown in Algorithms 1 and 2.

4. Complexity. In this section we derive the time and space complexity of the hierarchical decomposition algorithm. Assume that at each level of the hierarchy, we partition the graph into at most p partitions with an edge cut of size at most η.

4.1. Solve Time Complexity.

4.1.1. For Dense $\boldsymbol{b}$. We consider here the case in which the decomposition has already been computed at for each of the $\log_p n$ levels of the hierarchy, and we wish to find an $\boldsymbol{x}$ such that $\mathcal{L}\boldsymbol{x} = \boldsymbol{b}$ for some $\boldsymbol{b}$ such that $\mathbf{1}^T\boldsymbol{b} = 0$. In the following, we use $\widehat{W}$ as defined in Algorithm 1.

Suppose we have already computed $\tilde{\boldsymbol{x}} = (\hat{\mathcal{L}}_i^\nu)^{-1}\boldsymbol{b}$, and we wish to compute $(\mathcal{L}_i^\nu)^{-1}\boldsymbol{b}$. The amount of time required for this computation is given by the following steps:

1. Compute $\boldsymbol{b}' = F^T\tilde{\boldsymbol{x}}$. The matrix F has special sparse structure (see (3.5)), so this takes time $\eta + p - 1$.
2. Solve a linear system $\widehat{W}\boldsymbol{y} = \boldsymbol{b}'$. Assuming we have already factored $\widehat{W}$, this takes time at most $(\eta + p - 1)^2$.
3. Compute $\boldsymbol{y}' = \hat{F}\boldsymbol{y}$. The matrix $\hat{F}$ is in general dense, and has size $n_\nu \times \eta + p - 1$. So, this takes time $n_\nu(\eta + p - 1)$.
4. Add the result as $\boldsymbol{x} = \tilde{\boldsymbol{x}} + \boldsymbol{y}'$. This takes time n_ν.

So, this step requires time $O(n_\nu(\eta + p))$. But at the ν'th level of the hierarchy, $n_\nu = n/p^\nu$, so in total this step takes time $O(np^{-\nu}(\eta + p))$.

Algorithm 1 Construction of HDecomp Object

function HATLINV($\boldsymbol{b}; H.P, \nu$)
  Split $\boldsymbol{b}$ into subvectors $\boldsymbol{b}_1, ..., \boldsymbol{b}_p$.
  for $j = 1, ..., p$ **do**
    $\boldsymbol{x}_j \leftarrow$ HSOLVE($H.P[j], \boldsymbol{b}_j$)
  end for
  return $[x_1, ..., x_p]$.
end function
procedure HDECOMP(G_i^ν, ν)
  Initialize object H
  Find $G_{i,1}^{\nu+1}, ..., G_{i,p}^{\nu+1}$.
  for subgraph $G_{i,j}^{\nu+1}, j = 1, ..., p$ **do**
    $H.P[j] \leftarrow$ HDECOMP($G_{i,j}^{\nu+1}, \nu + 1$)
  end for
  $\widehat{L}^{-1} \leftarrow$ HATLINV($\cdot\,; H.P, \nu + 1$).
  Construct F as in (3.5).
  Compute $\widehat{F} \leftarrow \widehat{\mathcal{L}}^{-1} F$.
  Compute and factor $\widehat{W} \leftarrow W^{-1} + F^T \widehat{F}$.
  Return $\widehat{L}^{-1}, \widehat{W}, F, \widehat{F}$.
end procedure

Algorithm 2 Hierarchical Solve

Require: $\mathbf{1}^T \boldsymbol{b} = 0$.
procedure HSOLVE($H, \boldsymbol{b}$)
  if $\boldsymbol{b} = 0$ **then**
    return $\mathbf{0}$.
  end if
  $\widehat{\mathcal{L}}^{-1}, \widehat{W}^{-1}, F, \widehat{F} \leftarrow H$.
  Recursively compute $\tilde{\boldsymbol{x}} \leftarrow \widehat{\mathcal{L}}^{-1} \boldsymbol{b}$.
  Solve $\boldsymbol{y} \leftarrow \widehat{W}^{-1} F^T \tilde{\boldsymbol{x}}$.
  $\boldsymbol{x} \leftarrow \tilde{\boldsymbol{x}} - \widehat{F} \boldsymbol{y}$.
  return $\boldsymbol{x}$.
  if top of hierarchy **then**
    $\boldsymbol{x} \leftarrow \boldsymbol{x} - n^{-1} \mathbf{1}\mathbf{1}^T \boldsymbol{x}$.
  end if
end procedure

Now we wish to do this for each $i = 1, ..., p^\nu$ in this level of the hierarchy. Thus, the time required to solve $\mathcal{L}_i^\nu \boldsymbol{x} = \boldsymbol{b}$ for all i is

$$O\left(p^\nu \frac{n}{p^\nu}(\eta + p)\right) = O(n(\eta + p)). \tag{4.1}$$

Note that this time is independent of the hierarchy level ν. We must do this for each level of the hierarchy, of which there are $\log_p n$, for a final solve time complexity of

$$O((\eta + p) n \log_p n). \tag{4.2}$$

4.1.2. For Sparse $\boldsymbol{b}$. If $\boldsymbol{b}$ has only a small number of nonzero entries, the problem is easier. Suppose that $\boldsymbol{b}$ has only κ nonzero entries. Then at each hierarchy level ν, $\boldsymbol{b}_i^\nu = \mathbf{0}$ for all but at most κ of the indices i. The solution on these sub-vectors is simply $\mathbf{0}$.

So, at the ν'th level of the hierachy, instead of needing to compute p^ν inverses $(\mathcal{L}_i^\nu)^{-1}\boldsymbol{b}_i^\nu$, we need only compute κ of them. Thus, the time requirement at each level of the hierarchy is not $O((\eta+p)n)$, but

$$O\left((\eta+p)\kappa\frac{n}{p^\nu}\right). \tag{4.3}$$

We must do this for each of the $\log_p n$ levels of the hierarchy. However, at each hierarchy level ν, there are only at most κ node sets V_i^ν for which $\boldsymbol{b}$ has nonzero values. Therefore, the time requirement at each hierarcy level ν is given by (4.3), and so the total time complexity is

$$\begin{aligned} O\left(\sum_{\nu=1}^{\log_p n}\kappa(\eta+p)np^{-\nu}\right) &\leq O\left((\eta+p)n\kappa\sum_{\nu=1}^{\infty}p^{-\nu}\right) \\ &= O\left((\eta+p)n\kappa\right) \end{aligned} \tag{4.4}$$

4.2. Decomposition Time Complexity. Now we derive the time complexity associated with computing the decomposition at each level. We do not propose a particular algorithm or time complexity associated with the partitioning step; we simply refer to the time required to partition a graph as $\psi(n,p)$. A number of efficient partitioning algorithms and software packages are available, such as METIS [8] or SCOTCH [5]. Note that we also assume that whatever partitioning algorithm is used it successfully finds an edge cut of size at most η.

Construction of the hierarchy occurs in a top-down fashion, followed by computation of $\widehat{F}_i^\nu$ and computation and factoring of $\widehat{W}$, which occurs in a bottom-up fashion. For the top-down phase, we simply call whatever partitioning algorithm we use. At level ν in the hierarchy, each call costs $\psi(n/p^\nu,p)$, and there are p^ν of them to perform. Thus, the cost of the top-down phase is

$$O\left(\sum_{\nu=1}^{\log n}p^\nu\psi(n/p^\nu,p)\right) \tag{4.5}$$

If the partitioning method takes time $O(n)$, then this step takes $O(n\log_p n)$ time.

Now, for the bottom-up stage. Assume that the decomposition has been completed for all hierarchy levels beyond ν. We wish to compute $\widehat{F}_i^\nu = \widehat{\mathcal{L}}^{-1}F$. Because each column of F_i^ν is sparse with $\kappa\leq 2$, computing each column of $\widehat{F}_i^\nu$ requires the sparse solve time $O((\eta+p)np^{-\nu})$ (see (4.2)). There are $(\eta+p)$ such vectors to compute, so the computation costs time $O((\eta+p)^2np^{-\nu})$.

We must then compute and factor $\widehat{W}$. The matrix W is diagonal of size $\eta+p-1$, so the computation of W^{-1} requires time $O(\eta+p-1)$. The computation $(F_i^\nu)^T\widehat{F}_i^\nu$ requires constant time for each entry, because F_i^ν is sparse, for a total of $O((\eta+p)^2)$. Therefore, the construction of $\widehat{W}$ requires time

$$O\left((\eta+p)^2np^{-\nu}+(\eta+p)^2\right) = O\left((\eta+p)^2np^{-\nu}\right).$$

The factorization of $\widehat{W}$ then takes time $O((\eta+p)^3)$. Thus the total computation time is

$$O\left((\eta+p)^2np^{-\nu}+(\eta+p)^3\right). \tag{4.6}$$

There are p^ν such decompositions to compute at for level ν, resulting in a time of

$$O\left((\eta+p)^2n+(\eta+p)^3p^\nu\right) \tag{4.7}$$

for level ν.

We must do this for each of the $\log_p n$ levels, resulting in a total time complexity of

$$\begin{aligned} O\left(\sum_{\nu=1}^{\log_p n} (\eta+p)^2 n + (\eta+p)^3 p^\nu\right) =& O\left((\eta+p)^2 n \log n + (\eta+p)^3 \sum_{\nu=1}^{\log_p n} p^\nu\right) \\ =& O\left((\eta+p)^2 n \log n + (\eta+p)^3 \frac{p^{1+\log_p n}-1}{p-1}\right) \\ =& O\left((\eta+p)^2 n \log n + (\eta+p)^3 n\right). \end{aligned} \tag{4.8}$$

If we assume constant η and p, then this simplifies to

$$O\left(n \log n\right). \tag{4.9}$$

4.3. Storage Complexity. At level ν of the hierarchy, we must store $\hat{F}$ and the factorization of $\widehat{W}$ for each partition. The matrix $\hat{F}$ takes storage of size $(\eta+p)n/p^\nu$, and the factorization of $\widehat{W}$ takes storage of size $(\eta+p)^2$. Level ν of the hierarchy has p^ν such partitions, and so each level requires

$$O\left(p^\nu((\eta+p)np^{-\nu} + (\eta+p)^2)\right) = O(n(\eta+p) + p^\nu(\eta+p)^2)$$

storage. As there are $\log n$ levels, the total storage complexity is

$$O\left(\sum_{\nu=1}^{\log_p n} n(\eta+p) + p^\nu(\eta+p)^2\right) = O\left((\eta+p)n\log n + (\eta+p)^2 n\right).$$

5. Preconditioning. When a graph of interest does not have a hierarchy of p-cuts of size η for acceptably small p and η, we can can still use the above method to construct a preconditioner for the associated linear system. Of the original graph $G = (V, E)$, we build a *support graph* $G_S = (V, E_S)$ where $E_S \subset E$ such that G_S has the desired hierarchical structure. Also let $G_O = (V, E_O)$, where $E_O = E \setminus E_S$.

Now, let L be the graph Laplacian matrix of G, L_S the graph Laplacian of G_S, and L_O the graph Laplacian of G_O. Consider the use of L_S as a preconditioner for L. We would typically write the associated stationary iteration as

$$\boldsymbol{x}_t = (I - \omega^{-1} L_S^{-1} L)\boldsymbol{x}_{t-1} + \omega^{-1} L_S^{-1}\boldsymbol{b} \tag{5.1}$$

for some parameter ω. However, as L_S is a singular matrix, we need to define what we mean by L_S^{-1}. Note that L and L_S have the same null space, $\{\alpha\mathbf{1}, \alpha \in \mathbb{R}\}$, and so L_S is invertible on range(L). The solution to any linear system $L_S\boldsymbol{x} = \boldsymbol{b}$ is only determined up to a constant, so to define it uniquely we select the solution such that $\boldsymbol{x}^T\mathbf{1} = 0$.

With the above definition of L_S^{-1}, i.e., $L_S^{-1} = L_S^\dagger$, we wish to ensure that (5.1) is a convergent iteration. To that end, note that the eigenvalues and eigenvectors of $L_S^\dagger L$ are the same as that of the generalized eigenvalue problem

$$\lambda L_S \boldsymbol{x} = L\boldsymbol{x}. \tag{5.2}$$

Therefore, the eigenvalues of the error iteration matrix $\mathcal{E} = I - \omega^{-1} L_S^\dagger L$ are

$$\tilde{\lambda}_i = 1 - \omega^{-1}\lambda_i, \tag{5.3}$$

where λ_i is an eigenvalue of Equation (5.2). Thus, the iteration is convergent as long as $|\tilde{\lambda}_i| < 1$ for all i.

Now, for an eigenvalue λ_i, we have

$$\lambda_i = \frac{\langle \boldsymbol{x}^i, L\boldsymbol{x}^i\rangle}{\langle \boldsymbol{x}^i, L_S\boldsymbol{x}^i\rangle} = 1 + \frac{\langle \boldsymbol{x}^i, L_O\boldsymbol{x}^i\rangle}{\langle \boldsymbol{x}^i, L_S\boldsymbol{x}^i\rangle}. \tag{5.4}$$

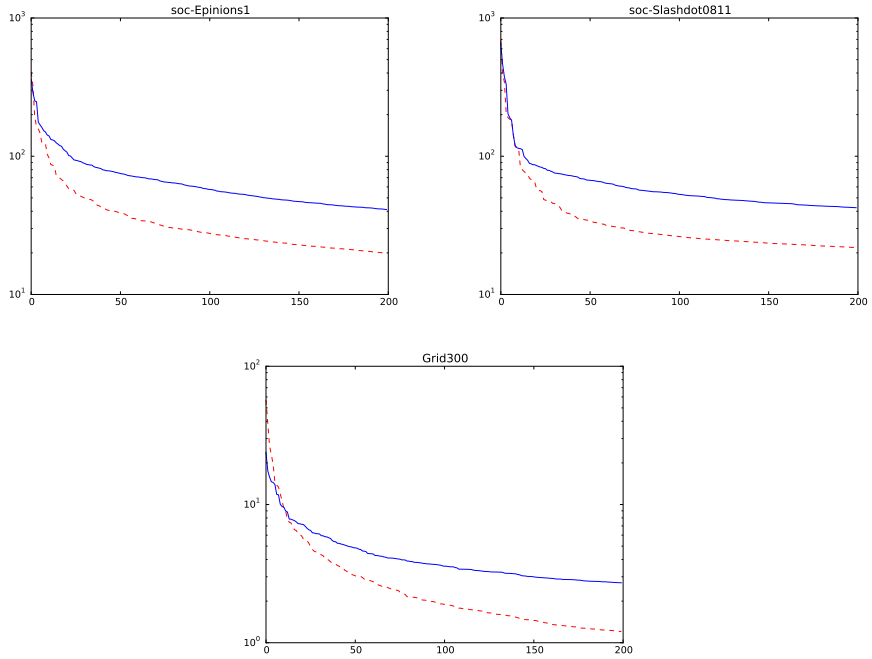


FIG. 5.1. *The blue line shows the first 200 eigenvalues for the matrix $L_S^\dagger L$ on the Epinions1 and Slashdot0811 networks with a maximum atomic subset size of 256, as well as on a 256×256 2D grid with a maximum atomic subset size of 64. The dotted red lines show the top eigenvalues of restricting to vectors that are constant on atomic subsets; that is, the square roots of eigenvalues of $H^T H$ in Equation* (6.3).

Graph Laplacians are positive semidefinite, and so we see that $\lambda_i \geq 1$ for all i.

Therefore, $|\tilde{\lambda}_i| < 1 \; \forall i$ if and only if $\omega > \lambda_1/2$, where λ_1 is the largest eigenvalue of Equation (5.2).

For small-to-medium sized graphs, this preconditioner is often highly effective on its own. However, for larger graphs, this may not be the case.

To illustrate why, consider Figure 5.1, which shows the top part of the spectrum for the preconditioned matrix $L_S^\dagger L$ in solid blue lines for a few example networks. The graphs used are the Epinions1 and Slashdot0811 networks from the Stanford Large Network Database [10], as well as a 256×256 2D grid. As can be seen, the eigenvalues tend to follow a distrbution that is visually similar to a power law, where the vast majority of the eigenvalues are quite close to 1, but a significant number of large eigenvalues still exist.

As can be seen here, the largest eigenvalue can be quite large. In fact, suppose that the initial cut bisecting G has n^β edges in it, for some $0 < \beta < 1$. There are only η such edges in the support graph G_S, so if

$$\boldsymbol{x} = \begin{bmatrix} \mathbf{1}^T & -\mathbf{1}^T \end{bmatrix}^T,$$

with all 1's on one side of the cut and all -1's on the other, then

$$\frac{\langle \boldsymbol{x}, L\boldsymbol{x} \rangle}{\langle \boldsymbol{x}, L_S \boldsymbol{x} \rangle} \propto n^\beta.$$

In 3D grids, $\beta = 2/3$, and in social networks, we expect β to typically be near 1. This suggests that L_S by itself is not an effective preconditioner for social networks. In the following section, we show how these damaging error modes can often be mitigated by using a two-grid preconditioning scheme.

6. Two-Grid Preconditioning. As discussed above, a few large eigenvalues of $L_S^\dagger L$ tend to damage the condition number of the system. We can avoid much of this problem through the use of two-grid preconditioning.

6.1. A Two-Grid Preconditioner.

6.1.1. Elements of a Multilevel Preconditioner. Traditional algebraic multigrid makes use of two ingredients: a *smoother* and a *coarse-grid correction*. The smoother is a preconditioner M^{-1} whose application tends to most effectively shrink those error modes associated with the largest eigenvalues of $M^{-1}L$, such as a Jacobi or Gauss-Seidel iteration.

The coarse-grid correction, on the other hand, shrinks those error modes associated with the smallest eigenvalues (the *algebraically smooth* modes) of $M^{-1}L$. This is done by defining an interpolation operator $P \in \mathbb{R}^{+^{n \times n_c}}, n_c \ll n$, that spans a *coarse subspace* and an associated *coarse matrix* $L_c = P^T L P$. Note that we impose P to have nonnegative entries. One then makes a coarse-grid correction by restricting the current residual to the coarse subspace and solving the resulting problem with L_c. Note that if the rows of P sum to 1, then L_c is also a graph Laplacian.

DEFINITION 6.1 (Solving problems with a coarse graph Laplacian).

Let P be a piecewise constant interpolant such that $P\mathbf{1}_c = \mathbf{1}$ and form $L_c = P^T L P$. For any given $\boldsymbol{b}_c$: $(\mathbf{1}_c)^T \boldsymbol{b}_c = 0$, we define the unique solution to the coarse problem

$$L_c \boldsymbol{x}_c = \boldsymbol{b}_c,$$

as follows. We select $\boldsymbol{x}_c$ such that $(\mathbf{1})^T P \boldsymbol{x}_c = 0$, which is equivalent to $(\mathbf{1}_c)^T (P^T P)\boldsymbol{x}_c = 0$, or $(P\boldsymbol{x}_c)^T (P\mathbf{1}_c) = 0$. We use the notation $\boldsymbol{x}_c = L_c^{\dagger} \boldsymbol{b}_c$.

The traditional combined three-step AMG preconditioner reads as follows:

1. Apply smoother:

$$\boldsymbol{x}_{t+\frac{1}{3}} = (I - M^{-1}L)\boldsymbol{x}_t + M^{-1}\boldsymbol{b}.$$

2. Apply coarse-grid correction:

$$\boldsymbol{x}_{t+\frac{2}{3}} = (I - PL_c^{\dagger}P^T L)\boldsymbol{x}_{t+\frac{1}{3}} + PL_c^{\dagger}P^T \boldsymbol{b}.$$

3. Apply smoother (in a symmetric fashion):

$$\boldsymbol{x}_{t+1} = (I - M^{-T}L)\boldsymbol{x}_{t+\frac{2}{3}} + M^{-T}\boldsymbol{b}.$$

This results in an error iteration matrix

$$\mathcal{E} = (I - M^{-1}L)(I - PL_c^{\dagger}P^T L)(I - M^{-1}L). \tag{6.1}$$

The (weighted) Jacobi smoother M performs local updates only. This leaves global error components which can be targeted by a coarse-grid correction. In our case, L_S is a global operator (similar to L) rather than a local operator. After applying $L_S^{\dagger}$, most of the error is eliminated, but, as we argue in Section 6.1.2, still there are error components that can be well approximated by a coarse-grid correction. We note that in these two cases we are looking at the spectra of two different operators, $L_S^{\dagger}L$ and $M^{-1}L$.

6.1.2. A Coarse Subspace. Assume that the nodes in G are ordered according to their hierachical decomposition as described in Section 3. Let

$$\boldsymbol{\ell}_i = \begin{bmatrix} \mathbf{0}^T & \cdots & \mathbf{0}^T & \mathbf{1}^T & \mathbf{0}^T & \cdots & \mathbf{0}^T \end{bmatrix}^T,$$

that is, the vector with ones on the nodes $V_i^{\nu_M}$ of an atomic subset, and zeroes everywhere else. Recalling that ν_M is the deepest level of the bisection hierarchy of Section 3, consider the matrix

$$\tilde{P} = \begin{bmatrix} \sqrt{n_{\nu_M,1}^{-1}}\boldsymbol{\ell}_1 & \cdots & \sqrt{n_{\nu_M,p^{\nu_M}}^{-1}}\boldsymbol{\ell}_{p^{\nu_M}} \end{bmatrix}. \tag{6.2}$$

Let

$$H = L_S^{\dagger} L \tilde{P} \tag{6.3}$$

In Figure 5.1, we show the square roots of the first 200 eigenvalues of the matrix $H^T H$ for each network in dashed red lines. This figure shows that these $\tilde{P}$ approximately span the invariant subspace associated with the largest eigenvalues of $L_S^\dagger L$.

So, define

$$P = \begin{bmatrix} \boldsymbol{\ell}_1 & \cdots & \boldsymbol{\ell}_{p^{\nu_M}} \end{bmatrix}. \tag{6.4}$$

This matrix is the same as $\tilde{P}$ except all its nonzero entries are 1. Then span(P) is our coarse subspace of dimension $n_c = p^{\nu_M}$. Note that, if one wishes to use a smaller coarse grid, one can simply take the subsets associated with another hierarchy level ν.

6.1.3. A Two-Grid Preconditioner. Unlike Jacobi or Gauss-Seidel smoothing, because $\sigma(L_S^\dagger L) \geq 1$, L_S is not convergent as a stationary iteration without weighting the iteration in such a way that would limit its effectiveness on the lowest-eigenvalue modes. This problem gets even worse if $L_S^\dagger$ is used as M^{-1} in the traditional AMG method of Section 6.1.1, as Equation (6.1) shows that the eigenvalues of $(I - L_S^\dagger L)$ would be squared. So, rather than placing the coarse-grid correction in the middle of the two-grid preconditioner as in classical algebraic multigrid, we place L_S in the middle. This leads to the three-step preconditioner

1. Apply coarse-grid correction:

$$\boldsymbol{x}_{t+\frac{1}{3}} = (I - PL_c^\dagger P^T L)\boldsymbol{x}_t + PL_c^\dagger P^T \boldsymbol{b}.$$

2. Apply L_S:

$$\boldsymbol{x}_{t+\frac{2}{3}} = (I - L_S^\dagger L)\boldsymbol{x}_{t+\frac{1}{3}} + L_S^\dagger \boldsymbol{b}.$$

3. Apply coarse-grid correction:

$$\boldsymbol{x}_{t+1} = (I - PL_c^\dagger P^T L)\boldsymbol{x}_{t+\frac{2}{3}} + PL_c^\dagger P^T \boldsymbol{b}.$$

First of all, we note that the above algorithm is well-defined, namely, the actions of $L_c^\dagger$, $L_c^\dagger P^T L$, and $L_S^\dagger$, $L_S^\dagger L$, are well-defined, since $\mathbf{1}_c^T (P^T L) = (P\mathbf{1}_c)^T L = \mathbf{1}^T L = 0$ (see Definition 6.1).

The above algorithm has the following property: if $\mathbf{1}^T \boldsymbol{x}_t = 0$, then also $\mathbf{1}^T \boldsymbol{x}_{t+s} = 0$ for $s = 1/3, 2/3, 1$. This is due to the fact that $\mathbf{1}^T P L_c^\dagger = \mathbf{1}_c^T (P^T P) L_c^\dagger = 0$ and $\mathbf{1}^T L_S^\dagger = 0$.

The above algorithm results in an error iteration matrix

$$\mathcal{E} = (I - PL_c^\dagger P^T L)(I - L_S^\dagger L)(I - PL_c^\dagger P^T L). \tag{6.5}$$

We do not claim that E has a norm less than one (in fact, we have $E\mathbf{1} = \mathbf{1}$), rather we will use this expression (or the algorithm above) to define a preconditioner B^{-1}.

6.2. Analysis of the Two-Grid Preconditioner. We wish to develop the tools to analyze the rate of convergence of this preconditioned system. Traditionally, a preconditioner is described by a matrix B, but the action of the preconditioner is through the application of B^{-1}. The convergence rate of a preconditioned conjugate gradient iteration is then determined by $\sqrt{\kappa_2/\kappa_1}$ ([1]), where

$$\kappa_1 B \preceq L \preceq \kappa_2 B.$$

In our case, B is not known a priori, and so we study it by deriving B^{-1} from the above algorithm. We can do this by assuming $E = I - B^{-1}L$ and writing (6.5) as

$$\begin{aligned}
\mathcal{E} =& (I - PL_c^\dagger P^T L)(I - L_S^\dagger L)(I - PL_c^\dagger P^T L) \\
=& (I - PL_c^\dagger P^T L)(I - PL_c^\dagger P^T L) - (I - PL_c^\dagger P^T L)L_S^\dagger L(I - PL_c^\dagger P^T L) \\
=& I - PL_c^\dagger P^T L - (I - PL_c^\dagger P^T L)L_S^\dagger (I - LPL_c^\dagger P^T)L \\
=& I - B^{-1}L,
\end{aligned}$$

where

$$B^{-1} = PL_c^\dagger P^T + (I - PL_c^\dagger P^T L)L_S^\dagger (I - LPL_c^\dagger P^T).$$

We notice now that if $\boldsymbol{b}$ is such that $\mathbf{1}^T\boldsymbol{b} = 0$, then $B^{-1}\boldsymbol{b}$ is well-defined. Indeed, since ${\mathbf{1}_c}^T P^T\boldsymbol{b} = \mathbf{1}^T\boldsymbol{b} = 0$, $L_c^\dagger P^T\boldsymbol{b}$ is well-defined. Also, $L_S^\dagger\boldsymbol{b}$ and $L_S^\dagger L$ are well-defined. Similarly, ${\mathbf{1}_c}^T P^T L = \mathbf{1}^T L = 0$, hence $L_c^\dagger P^T L$ is well-defined as well. An additional property of B^{-1} is that $\mathbf{1}^T B^{-1}\boldsymbol{b} = 0$.

We have that $L^\dagger$ is symmetric positive semi-definite, hence $(L^\dagger)^{\frac{1}{2}}$ is well-defined as a symmetric positive semi-definite matrix. The following idenity holds:

$$L(L^\dagger)^{\frac{1}{2}} = L^{\frac{1}{2}}.$$

Now, consider

$$L^{\frac{1}{2}}\mathcal{E}(L^\dagger)^{\frac{1}{2}} = (I - L^{\frac{1}{2}}PL_c^\dagger P^T L^{\frac{1}{2}})(I - L^{\frac{1}{2}}L_S^\dagger L^{\frac{1}{2}})(I - L^{\frac{1}{2}}PL_c^\dagger P^T L^{\frac{1}{2}})L^{\frac{1}{2}}(L^\dagger)^{\frac{1}{2}}.$$

We also have,

$$L^{\frac{1}{2}}(L^\dagger)^{\frac{1}{2}} - L^{\frac{1}{2}}B^{-1}L^{\frac{1}{2}} = L^{\frac{1}{2}}\mathcal{E}(L^\dagger)^{\frac{1}{2}}.$$

Therefore, for any $\boldsymbol{x}:\ \mathbf{1}^T\boldsymbol{x} = 0$, we have $L^{\frac{1}{2}}(L^\dagger)^{\frac{1}{2}}\boldsymbol{x} = \boldsymbol{x}$, hence

$$\boldsymbol{x}^T(I - L^{\frac{1}{2}}B^{-1}L^{\frac{1}{2}})\boldsymbol{x} = \boldsymbol{y}^T(I - L^{\frac{1}{2}}L_S^\dagger L^{\frac{1}{2}})\boldsymbol{y} \leq 0, \tag{6.6}$$

where $\boldsymbol{y} = (I - L^{\frac{1}{2}}PL_c^\dagger P^T L^{\frac{1}{2}})\boldsymbol{x}$. That is, the operator $I - L^{\frac{1}{2}}B^{-1}L^{\frac{1}{2}}$ is negative semi-definite in the subspace $\{\boldsymbol{x}:\ \mathbf{1}^T\boldsymbol{x} = 0\}$. In other words, the operator $L^\dagger - B^{-1}$ is negative semi-definite in the same subspace, which implies that B^{-1} is positive definite (since $L^\dagger$ is positive definite) in the same subspace. Hence, we can define its inverse, B, well-defined in the subspace $\{\boldsymbol{x}:\ \mathbf{1}^T\boldsymbol{x} = 0\}$. More over, we have the inequality

$$B \preceq L. \tag{6.7}$$

We now present a useful lemma.

LEMMA 6.2. *Given two vector spaces V and W contained in some $\mathbb{R}^n$, consider two matrices T and N acting on vectors in $\mathbb{R}^n$. We assume that T is symmetric and it maps V onto itself, whereas N maps V onto W, i.e., for each $\boldsymbol{w} \in W$ there is a $\boldsymbol{v} \in V$ such that $N\boldsymbol{v} = \boldsymbol{w}$. Moreover, we assume that $T - N^T N$ is s.p.d. on V. Consider*

$$Z = N\left(T - N^T N\right)^{-1} N^T.$$

We have that Z is s.p.d. on W, hence invertible on W, and for each $\boldsymbol{w} \in W$,

$$\frac{\boldsymbol{w}^T Z^{-1}\boldsymbol{w}}{\boldsymbol{w}^T\boldsymbol{w}} = -1 + \min_{\boldsymbol{v}:\ N\boldsymbol{v}=\boldsymbol{w}} \frac{\boldsymbol{v}^T T\boldsymbol{v}}{\boldsymbol{w}^T\boldsymbol{w}}.$$

Proof. This lemma is similar to Lemma 3.1 of [6], as is its proof. For completeness we present its proof here.

We first show that if $N^T\boldsymbol{w} = 0$ for $\boldsymbol{w} \in W$, then $\boldsymbol{w} = 0$. Indeed, since $\boldsymbol{w} = N\boldsymbol{v}$ for some $\boldsymbol{v} \in V$, we have that $N^T N\boldsymbol{v} = 0$. The latter in particular implies $\boldsymbol{v}^T N^T N\boldsymbol{v} = 0$, i.e., $\|N\boldsymbol{v}\| = 0$ and hence $0 = N\boldsymbol{v} = \boldsymbol{w}$. This implies that Z is invertible, hence s.p.d. on W.

Consider now the constrained minimization problem:

Given $\boldsymbol{w} \in W$ compute

$$\frac{1}{2}\,\boldsymbol{v}^T T\boldsymbol{v} \mapsto \min,$$

subject to $N\boldsymbol{v} = \boldsymbol{w}$.

Forming the Lagrangian $\frac{1}{2}\,\boldsymbol{v}^T T\boldsymbol{v} - \boldsymbol{\lambda}^T(\boldsymbol{w} - N\boldsymbol{v})$, the necessary consitions for minimum give

$$\begin{bmatrix} T & N^T \\ N & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} 0 \\ \boldsymbol{w} \end{bmatrix}.$$

An equivalent system is

$$\begin{bmatrix} T - N^T N & N^T \\ N & 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -N^T\boldsymbol{w} \\ \boldsymbol{w} \end{bmatrix}.$$

This system has unique solution since $T - N^T N :\ V \mapsto V$ is s.p.d. and its negative Schur complement $Z = N(T - N^T N)^{-1}N^T :\ W \mapsto W$ is also s.p.d. The solution equals,

$$\begin{aligned} \boldsymbol{v} &= -(T - N^T N)^{-1}N^T(\boldsymbol{\lambda} + \boldsymbol{w}), \\ -Z(\boldsymbol{\lambda} + \boldsymbol{w}) &= \boldsymbol{w}. \end{aligned}$$

That is,

$$\boldsymbol{v} = (T - N^T N)^{-1}N^T Z^{-1}\boldsymbol{w}.$$

Then,

$$\boldsymbol{w}^T Z^{-1}\boldsymbol{w} = -\boldsymbol{w}^T(\boldsymbol{\lambda} + \boldsymbol{w}) = -\boldsymbol{w}^T\boldsymbol{w} - \boldsymbol{w}^T\boldsymbol{\lambda}.$$

On the other hand

$$\boldsymbol{\lambda}^T\boldsymbol{w} = \boldsymbol{\lambda}^T N\boldsymbol{v} = \boldsymbol{v}^T N^T\boldsymbol{\lambda} = \boldsymbol{v}^T(-N^T\boldsymbol{w} - (T - N^T N)\boldsymbol{v}) = -\|\boldsymbol{w}\|^2 - \boldsymbol{v}^T(T - N^T N)\boldsymbol{v}.$$

That is, since $\boldsymbol{v} :\ N\boldsymbol{v} = \boldsymbol{w}$, we have

$$\boldsymbol{w}^T Z^{-1}\boldsymbol{w} = \boldsymbol{v}^T(T - N^T N)\boldsymbol{v} = -\boldsymbol{w}^T\boldsymbol{w} + \boldsymbol{v}^T T\boldsymbol{v}$$

Thus, using the fact that $\boldsymbol{v}$ is the minimizer, we have

$$\frac{\boldsymbol{w}^T Z^{-1}\boldsymbol{w}}{\boldsymbol{w}^T\boldsymbol{w}} = -1 + \min_{\boldsymbol{v}:\ N\boldsymbol{v}=\boldsymbol{w}} \frac{\boldsymbol{v}^T T\boldsymbol{v}}{\boldsymbol{w}^T\boldsymbol{w}},$$

which complets the proof. □

We now present a formula for the condition number of B with respect to L viewed as s.p.d. operators acting on the subspace $S \equiv \{\boldsymbol{x} : \mathbf{1}^T\boldsymbol{x} = 0\}$. This proof draws from that of Theorem 4.1 in [6].

THEOREM 6.3. *Let $\pi_L = PL_c^{\dagger}P^T L$, and let S denote the subspace $\{\boldsymbol{x} : \mathbf{1}^T\boldsymbol{x} = 0\}$. Then*

$$B \preceq L \preceq \kappa B. \tag{6.8}$$

over the subspace S, where

$$\kappa = \sup_{\boldsymbol{w}} \frac{((I - \pi_L)\boldsymbol{w})^T\, L\, ((I - \pi_L)\boldsymbol{w})}{\boldsymbol{w}^T L_S \boldsymbol{w}}. \tag{6.9}$$

Proof. We first note that π_L is a projection (as an operator acting on S). The same holds for its symmeric version

$$\overline{\pi}_L = L^{\frac{1}{2}} PL_c^{\dagger}P^T L^{\frac{1}{2}}.$$

The fact that $B \preceq L$ was shown above (see (6.7)). To prove the other direction, use the fact that for s.p.d. operators L and B (as mappings on S) we have

$$\kappa = \sup_{\boldsymbol{v}\in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}}\boldsymbol{v}}{\|\boldsymbol{v}\|^2}.$$

We note that the above maximum is achieved in the subspace $\{\boldsymbol{v} = (I - \overline{\pi}_L)\boldsymbol{w},\ \boldsymbol{w} \in S\} \subset S$. We have

$$\kappa = \sup_{\boldsymbol{v}\in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|\boldsymbol{v}\|^2} \tag{6.10}$$

$$= \sup_{\boldsymbol{v}\in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|\overline{\pi}_L \boldsymbol{v}\|^2 + \|(I - \overline{\pi}_L)\boldsymbol{v}\|^2} \tag{6.11}$$

$$\leq \sup_{\boldsymbol{v}\in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|(I - \overline{\pi}_L)\boldsymbol{v}\|^2} \tag{6.12}$$

$$= \sup_{\boldsymbol{w}=(I-\overline{\pi}_L)\boldsymbol{v},\ \boldsymbol{v}\in S} \frac{\boldsymbol{v}^T L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}} \boldsymbol{v}}{\|\boldsymbol{w}\|^2} \tag{6.13}$$

$$\leq \kappa. \tag{6.14}$$

Using the fact that $(I - \overline{\pi}_L)$ is a projection and Equation (6.6), note that, if $\boldsymbol{v} \in S$,

$$\begin{aligned}\boldsymbol{v}^T (I - L^{\frac{1}{2}} B^{-1} L^{\frac{1}{2}})\boldsymbol{v} =& ((I - \overline{\pi}_L)\boldsymbol{v})^T \left(I - ((I - \overline{\pi}_L)) L^{\frac{1}{2}} L_S^{\dagger} L^{\frac{1}{2}} ((I - \overline{\pi}_L))\right) ((I - \overline{\pi}_L)\boldsymbol{v}) \\ \geq& (1 - \kappa) ((I - \overline{\pi}_L)\boldsymbol{v})^T ((I - \overline{\pi}_L)\boldsymbol{v}).\end{aligned}$$

This implies that for $Z = (I - \overline{\pi}_L) L^{\frac{1}{2}} L_S^{\dagger} L^{\frac{1}{2}} (I - \overline{\pi}_L)$, we have (based on (6.10))

$$\kappa = \sup_{\boldsymbol{v}\in S} \frac{\boldsymbol{v}^T Z \boldsymbol{v}}{\boldsymbol{v}^T (I - \overline{\pi}_L)\boldsymbol{v}}.$$

Now, let $N = (I - \overline{\pi}_L) L^{\frac{1}{2}}$. Then $N^T N = L(I - \pi_L)$.

Let $T = L_S + L(I - \pi_L)$. Then T is symmetric and positive semi-definite. Furthermore, $T - N^T N = L_S$. Therefore,

$$Z = (I - \overline{\pi}_L) L^{\frac{1}{2}} L_S^{\dagger} L^{\frac{1}{2}} (I - \overline{\pi}_L) = N(T - N^T N)^{-1} N^T.$$

To use Lemma (6.2), introduce the spaces $V = S$ and let $W = \text{range}(I - \overline{\pi}_L) \cap S$. Then Lemma 6.2 gives

$$\kappa = \sup_{\tilde{\boldsymbol{w}}\in W} \frac{\tilde{\boldsymbol{w}}^T Z \tilde{\boldsymbol{w}}}{\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{w}}} = \sup_{\tilde{\boldsymbol{w}}\in W} \frac{\tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{w}}}{\tilde{\boldsymbol{w}}^T Z^{-1} \tilde{\boldsymbol{w}}} = \frac{1}{\inf_{\tilde{\boldsymbol{w}}\in W} \left(-1 + \inf_{\boldsymbol{v}: N\boldsymbol{v} = \tilde{\boldsymbol{w}}} \left(\boldsymbol{v}^T T \boldsymbol{v} / \tilde{\boldsymbol{w}}^T \tilde{\boldsymbol{w}}\right)\right)}.$$

Now, let $\tilde{\boldsymbol{w}} = L^{\frac{1}{2}} \boldsymbol{w}$. Then $(\tilde{\boldsymbol{w}})^T \tilde{\boldsymbol{w}} = \boldsymbol{w}^T L \boldsymbol{w}$, and

$$\begin{aligned} & N\boldsymbol{v} = L^{\frac{1}{2}} \boldsymbol{w} \\ \implies & L^{\frac{1}{2}} \boldsymbol{w} = (I - \overline{\pi}_L) L^{\frac{1}{2}} \boldsymbol{v} \\ \implies & \boldsymbol{w} = (I - \pi_L)\boldsymbol{v}. \end{aligned}$$

Note that $\boldsymbol{w} \in \text{range}(I - \pi_L) \cap S$, which we denote $\overline{W}$. This leads to

$$\kappa^{-1} = \inf_{\boldsymbol{w}\in\overline{W}} \left(\inf_{\boldsymbol{v}:(I-\pi_L)\boldsymbol{v}=\boldsymbol{w}} \frac{\boldsymbol{v}^T T \boldsymbol{v}}{\boldsymbol{w}^T L \boldsymbol{w}} - 1 \right).$$

But

$$\boldsymbol{v}^T T \boldsymbol{v} = \boldsymbol{v}^T L_S \boldsymbol{v} + \boldsymbol{v}^T N^T N \boldsymbol{v} = \boldsymbol{v}^T L_S \boldsymbol{v} + \boldsymbol{w}^T L \boldsymbol{w},$$

so

$$\kappa^{-1} = \inf_{\boldsymbol{w}\in\overline{W}} \inf_{\boldsymbol{v}:(I-\pi_L)\boldsymbol{v}=\boldsymbol{w}} \frac{\boldsymbol{v}^T L_S \boldsymbol{v}}{\boldsymbol{w}^T L \boldsymbol{w}} = \inf_{\boldsymbol{v}:\boldsymbol{v}\in S} \frac{\boldsymbol{v}^T L_S \boldsymbol{v}}{((I - \pi_L)\boldsymbol{v})^T L ((I - \pi_L)\boldsymbol{v})}.$$

But both the numerator and the denominator ignore any components of $\boldsymbol{v}$ in the direction of $S^{\perp} = \{\alpha \mathbf{1}\}$, so we may write this simply as

$$\kappa = \sup_{\boldsymbol{v}} \frac{((I - \pi_L)\boldsymbol{v})^T L ((I - \pi_L)\boldsymbol{v})}{\boldsymbol{v}^T L_S \boldsymbol{v}}. \tag{6.15}$$

This completes the proof. □

COROLLARY 6.4. *When used in combination with the preconditioned conjugate gradient method, the number of iterations required when using the two-grid preconditioner of Section 6.1 is bounded by* $O(\sqrt{\kappa})$*, where* κ *is given in Equation* (6.9)*.*

Remark In this paper we have not provided specific algorithms for constructing the partition hierarchy or for deciding which edges to keep for L_S. We can, however, use the above theorem to develop some guidelines. Decompose L as $L = L_W + L_B$, where L_W is a disconnected graph Laplacian with the edges within the smallest-level atomic subsets $G_i^{\nu_M}$ and L_B is a graph Laplacian with the between-subset edges. Note that all within-atomic subset edges are part of the support graph Laplacian L_S, so L_W is itself a support graph Laplacian of L_S. We can further decompose L_B as $L_B = L_{BS} + L_{BS^{\perp}}$, where L_{BS} contains the edges of L_B that are in L_S, and $L_{BS^{\perp}}$ contains the edges that are not. Note that $L_S = L_W + L_{BS}$.

The coarse grid ignores all within-atomic subset edges. That is, $P^T L_W = \mathbf{0}^T$. Therefore, $L_c = P^T L P = P^T L_B P$. Thus, we may rewrite Equation (6.9) as

$$\kappa = \sup_{\boldsymbol{w}} \frac{((I - \pi_L)\boldsymbol{w})^T L ((I - \pi_L)\boldsymbol{w})}{\boldsymbol{w}^T (L_W + L_{SB})\boldsymbol{w}},$$

where $\pi_L = P(P^T L_B P)^{\dagger} P^T L_B$.

Suppose that

$$\boldsymbol{w}^T L \boldsymbol{w} = \boldsymbol{w}^T (L_W + L_{BS} + L_{BS^{\perp}})\boldsymbol{w} \approx \boldsymbol{w}^T L_{BS^{\perp}} \boldsymbol{w}.$$

In this case, the denominator of Equation (6.9) will be small. Suppose further that $P^T L_B \boldsymbol{w} \approx \mathbf{0}$. This can happen, for example, through "parallel edges" in $L_{BS^{\perp}}$, that is, multiple edges stretching between the same two atomic subsets. Then the numerator of Equation (6.9) is approximately $\boldsymbol{w}^T L \boldsymbol{w} \approx \boldsymbol{w}^T L_{BS^{\perp}} \boldsymbol{w}$, which is by assumption not small. That is, we have a small denominator and a large numerator, resulting in a large κ.

So, we would like to construct the partition hierarchy and L_S to avoid this situation. This leads to the following guidelines:

1. The atomic subsets should be as well-connected as possible, so that $\boldsymbol{w}^T L_W \boldsymbol{w}$ is as large as possible.
2. Often, high-degree nodes have more incident edges than can be captured in L_{SB}. Therefore, as much as possible, high-degree nodes and their neighbors should be placed in the same atomic subset.
3. Suppose u and v are two high-degree nodes, are placed in the same atomic subset, and have many neighbors in other atomic subsets. Then the intersection between u's neighboring atomic subsets and v's neighboring atomic subsets should be as small as possible. Otherwise, a $\boldsymbol{w}$ such that $w_u \approx -w_v$ with zeros everywhere else can result in a large κ.

7. Experiments. In this section we present experiments to demonstrate the efficiency of the algorithms presented here.

7.1. Direct Solver Scaling. In this section we demonstrate the scaling behavior of the direct solver described in Section 3. We recursively construct a graph Laplacian as follows:

1. If constructing a graph of size $N \leq 8$, add 20 edges uniformly at random. If the resulting graph is not connected, try again.
2. If constructing a graph of size $N > 8$, construct two graphs each of size $N/2$ and connect them with 8 edges selected uniformly at random.

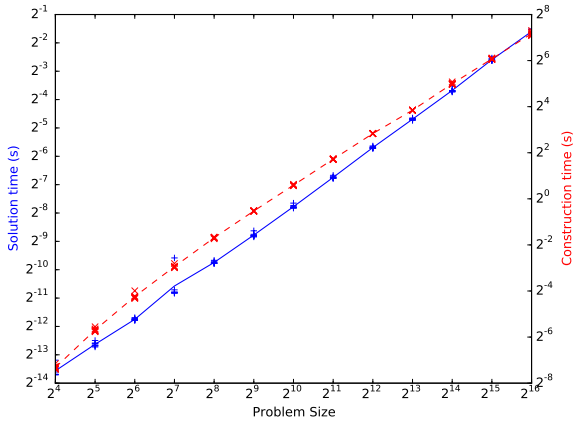


FIG. 7.1. *Time to construct a solver (red X), and time to perform a solve (blue +) using the direct solver of Section 3 on graphs that can be recursively bisected with cuts of size 8. Dotted red and solid blue lines connect sample means of 8 samples each.*

We do this for a range of graph sizes. For each graph size, we perform the test 8 times. The results of this scaling test can be seen in Figure 7.1. Along the x-axis is the total size of the graph being tested. The red line shows the mean time to perform the decomposition to construct the solver, while the blue line shows mean time to perform a solve after the solver has been constructed.

The favorable scaling of this method is clear, as the plot appears nearly linear.

7.2. Preconditioning. In this section we explore the effectiveness of using our method as a preconditioner within the Preconditioned Conjugate Gradient method.

We have not, in this paper, specified methods for computing the recursive bisection on a graph or deciding which edges to keep between atomic subset in the support graph L_S. For 2D and 3D grids, we select the longest axis-aligned dimension and cut along its middle, spreading the support graph edges out evenly along that cutting plane.

For other graphs, we recursively bisect using METIS [8], constrained to give connected partitions, and we select edges to keep in L_S uniformly at random. We believe this to be a reasonable heuristic for guidelines 1 and 2 at the end of Section 6.

First, we test the behavior of our method on four different types of artificial graphs:

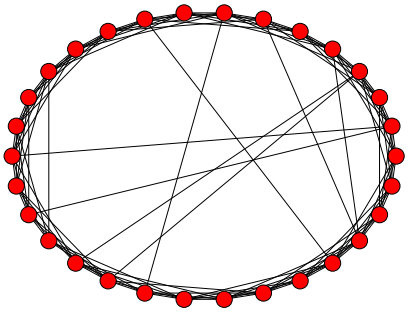
- Two-dimensional square grids.
- Three-dimensional cube grids.
- Watts-Strogatz random graph models [14].
- Barabási-Albert random graph models [3].

The Watts-Strogatz random graph model is a "ring lattice with random rewiring." It a popular random graph model used to capture behavior in which groups of nodes tend to be tightly clustered but still have some "long range" edges. An example can be seen in Figure 7.2(a).

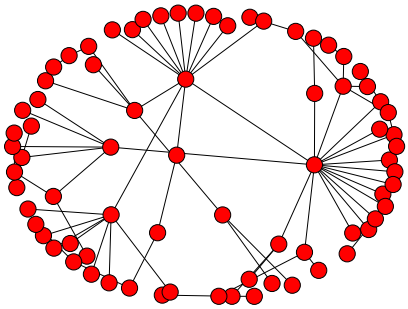
The Barabási-Albert random graph model is a popular "preferential attachment" model in which nodes are added to the graph sequentially. New nodes are connected to other nodes at random, with the probability of connection proportional to the degree of that node. It captures the skewed degree distribution present in many social networks, though there is no clustering in this model. An example can be seen in Figure 7.2(b).

Although artificial graphs are different from real-world graphs, they are useful because they allow us to study the behavior of our method as the size of the problem grows on a given class of graphs. For each of the above four graph types, we examine rates of convergence while varying problem size, acceptable support graph cut size, and coarse graph size.

In the following, let n_A denote the maximum atomic subgraph size and let n_c denote the coarse graph size. Unless stated otherwise, the number of edges per cut in the support graph is also n_A.



(a) A sample Watts-Strogatz graph.



(b) A sample Barabási-Albert graph.

7.2.1. Constant Cut Size, Linear Coarse Graph Size. Here we study the behavior of our method when the maximum atomic subgraph and support graph cut size are kept constant, but the coarse grid size n_c is allowed to grow linearly.

Figures 7.7 and 7.8 show the results of these tests on artificial graphs. These tests show that the coarse-grid correction drastically improves performance of PCG in most cases.

On grids, after an initial rise, the convergence rate of the two-grid method is independent of graph size. Note that some tests only require a single iteration because the cut size is large enough that $L_S = L$ in those cases.

On Watts-Strogatz graphs, we test with mean degree $k = 10$ and rewiring probability $\beta = 0.1$. The rate of growth of iteration count is largely independent of n_A, but increasing n_A tends to improve convergence by a constant.

On Barábasi-Albert graphs, we test with new node degree $m = 10$. The displayed regression lines are less informative here, especially for $n_A = 256$. However, we see that increasing n_A tends to improve convergence for smaller graphs, and slightly hamper convergence for larger graphs.

In Figure 7.2, we test on an array of networks from the Stanford Large Network Database, [10], and the 10th DIMACS Challenge, [2]. For a complete list of the networks tested, see Appendix 9.

As n_A grows, both the one-level and two-level methods improve, but the difference between the two shrink. This suggests that the increasing cut size tends to matter more than the decreasing coarse-grid size. This is different behavior than seen in the Barábasi-Albert model; we hypothesize that this is due to the tendency of real-world networks to cluster, while Barábasi-Albert graphs have no clustering.

7.2.2. Increasing Cut Size, Constant Coarse Grid Size. In this section we examine purely the effect of increasing the acceptable cut size in the support graph without changing n_A or the size of the coarse graph. We study a 2D grid of size 320×320, a 3D grid of size $47 \times 47 \times 47$, a Watts-Strogatz graph of size 103,000, and a Barábasi-Albert graph also of size 103000. We use the same parameters for the random graph models as in the previous section. The results in Figure 7.3 show that while both methods improve with increasing acceptable cut size, the two-level method is significantly less sensitive to these changes.

We also tested increasing cut sizes on a selection of real-world networks. These results can be seen in Figure 7.4. Again we see that while increasing cut size improves convergence, the two-level method is less sensitive to this change than the one-level method.

7.2.3. Constant Cut Size, Increasing Coarse Grid Size. In this section we examine purely the effects of shrinking coarse node size, or equivalently, increasing coarse grid size n_c, without changing n_A or the acceptable cut size. As in the previous section, we study a 2D grid of size 320×320, a 3D grid of size $47 \times 47 \times 47$, a Watts-Strogatz graph of size 103,000, and a Barábasi-Albert graph also of size 103000. We use the same parameters for the random graph models as above. Note that the size of

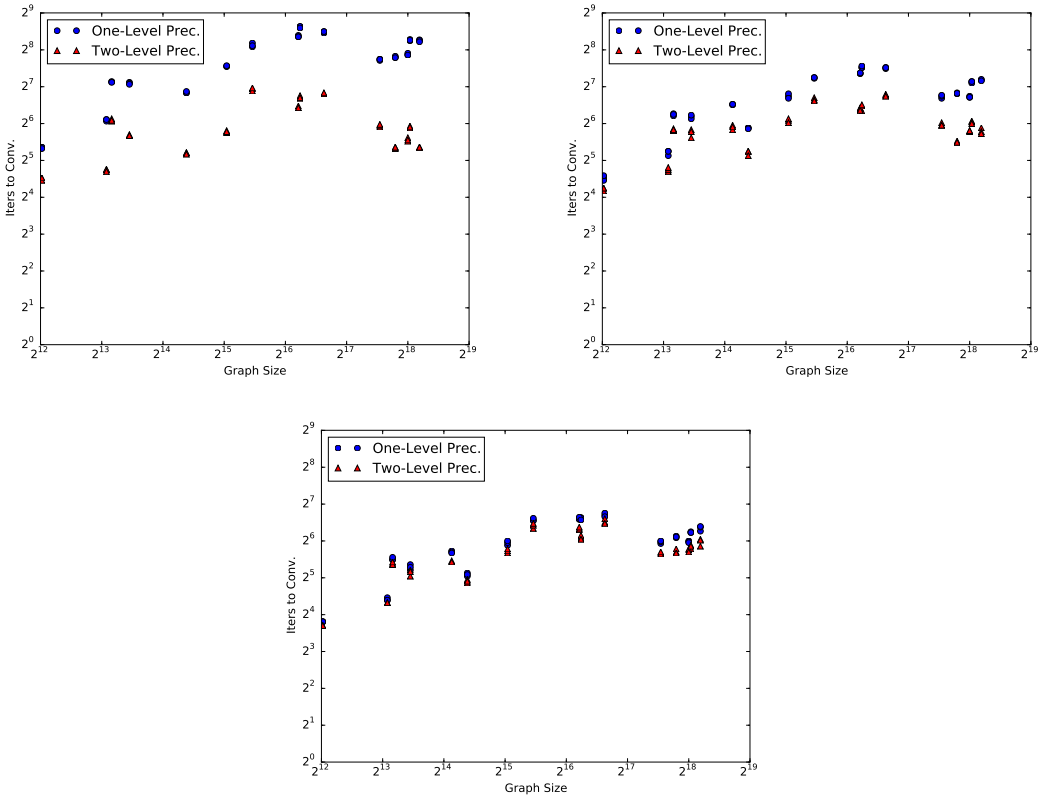


FIG. 7.2. *Iterations required to reach a residual norm of* 10^{-6} *and log-log regression lines for various real-world graphs obtained from SNAP and the 10th DIMACS Challenge. Maximum atomic subgraph size is 16, 64, and 256, respectively.*

a coarse grid scales approximately linearly with the inverse of the coarse node sizes. We choose sets of nodes to aggregate in the creation of a coarse graph as subsets V_i^ν obtained in the partition hierarchy.

The results in Figure 7.5 show that the rate of convergence is most improved on 2D grid, 3D grid, and Watts-Strogatz graphs for smaller coarse nodes, but that the Barábasi-Albert graphs still show significant improvement even for large coarse node sizes. However, the Barábasi-Albert graphs still require the most iterations.

We also tested shrinking coarse node size (increasing n_c) on a selection of real-world networks. These results can be seen in Figure 7.6. Most of the real-world graphs show significant improvements in convergence as the coarse nodes shrink, except for soc-Slashdot0811, which improves but not as drastically.

8. Conclusion. In this paper, we presented a parallelizable method for solving graph Laplacian-based linear systems derived from graphs that can be hierarchically bipartitioned with small edge cuts and showed that this method solves such systems in time $O(n \log n)$. We then used this method to construct a support graph-based preconditioner for graph Laplacian systems that do not have this property. Finally, we augmented this method with a two-grid approach to account for the weaknesses in the one-level preconditioner. We presented an analysis of the two-grid method, as well as a theorem deriving the condition number of two-grid support graph-based preconditioners.

We did not develop methods for partitioning a graph, or for selection of edges to keep in a support graph. Future work will include addressing these issues, especially in the context of graph Laplacians derived from real-world networks. In addition, we will explore the use of recursive coarse-grid corrections in a multigrid method.

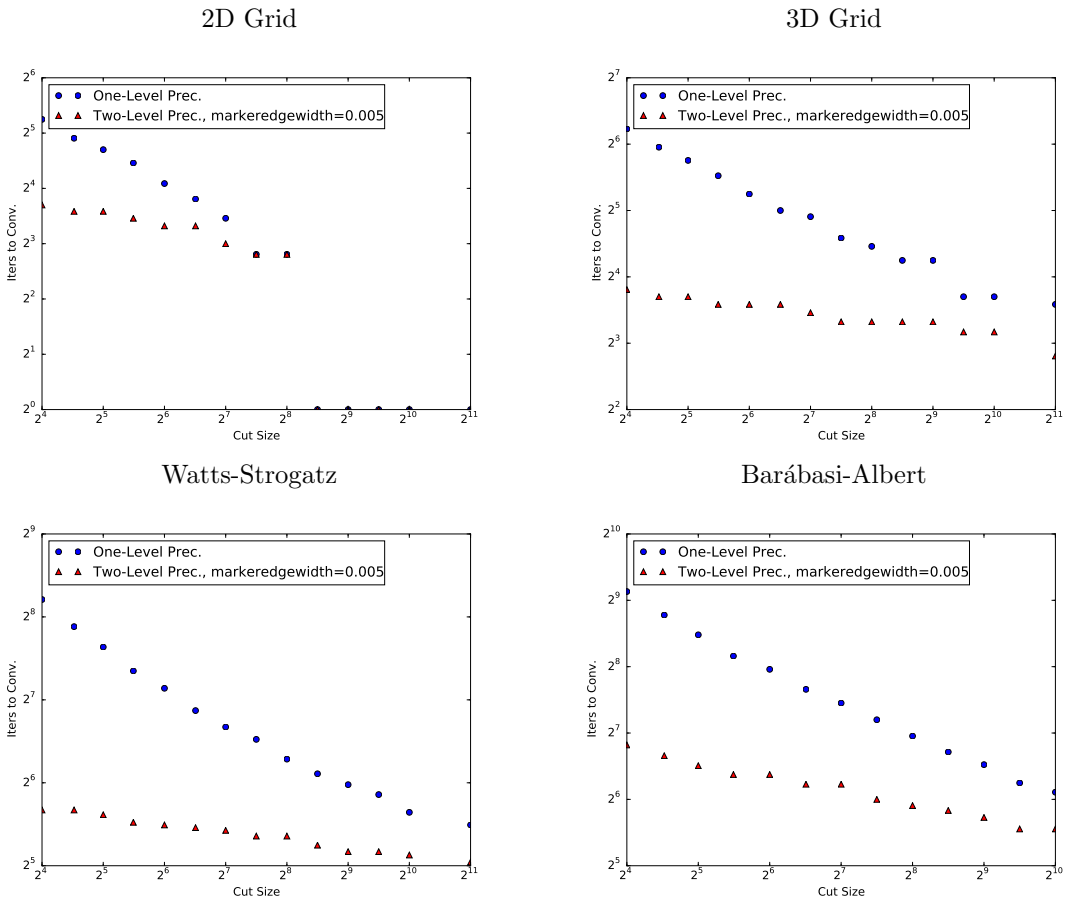


FIG. 7.3. *Maximum cut size in L_S vs. iterations required to reach a residual norm of 10^{-6} for several artificial graphs. All graphs have approximately 103,000 nodes. While both one-level and two-level methods improve with increasing cut size, the two-level method is significantly less sensitive.*

REFERENCES

[1] O. AXELSSON, *A survey of preconditioned iterative methods for linear systems of algebraic equations*, BIT Numerical Mathematics, 25 (1985), pp. 165–187. 11

[2] D. A. BADER, A. KAPPES, H. MEYERHENKE, P. SANDERS, SCHULZ C., AND WAGNER D., *Benchmarking for graph clustering and partitioning*, in Encyclopedia of Social Network Analysis and Mining, R. Alhajj and J. Rokne, eds., Springer-Verlag, New York, 2014. 17

[3] A.-L. BARABÁSI AND R. ALBERT, *Emergence of scaling in random networks*, Science, 286 (1999), pp. 509–512. 16

[4] A. BRANDT, S. F. MCCORMICK, AND J. W. RUGE, *Algebraic multigrid (AMG) for sparse matrix equations*, in Sparsity and its Application, David J. Evans, ed., Cambridge, 1985, Cambridge University Press, pp. 257–284. 2

[5] C. CHEVALIER AND F. PELLEGRINI, *Pt-scotch: A tool for efficient parallel graph ordering*, Parallel Matrix Algorithm and Applications, 34 (2008), pp. 318–331. 7

[6] R. D. FALGOUT, P. S. VASSILEVSKI, AND L. T. ZIKATANOV, *On two-grid convergence estimates*, Numerical Linear Algebra Appl., 12 (2005), pp. 471–494. 12, 13

[7] A. GEORGE, *Nested dissection of a regular finite element mesh*, Siam Journal on Numerical Analysis, 10 (1973), pp. 345–363. 1

[8] G. KARYPIS AND V. KUMAR, *A fast and highly quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1999), pp. 359–392. 7, 16

[9] I. KOUTIS, G. L. MILLER, AND R. PENG, *A nearly-m log n time solver for sdd linear systems*, in IEEE 52nd Annual Symposium on Foundations of Computer Science, 2011, pp. 590–598. 2

[10] JURE LESKOVEC AND ANDREJ KREVL, *SNAP Datasets: Stanford large network dataset collection.* `http://snap.stanford.edu/data`, June 2014. 9, 17

[11] R. J. LIPTON, D. J. ROSE, AND R. E. TARJAN, *Generalized nested dissection*, SIAM Journal on Numerical Analysis, 16 (1979), pp. 346–358. 1

[12] D. A. Spielman and S.-H. Teng, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in 36th Annual ACM Symposium on Theory of Computing, 2004, pp. 81–90. 2
[13] P. S. Vassilevski, *Multilevel Block Factorization Preconditioners*, Springer, Livermore, California, 2008. 5
[14] D. Watts and S. Strogatz, *Collective dynamics of 'small-world' networks*, Nature, 393 (1998), pp. 440–442. 16

9. Appendix: List of Real-World Graphs Tested.

- **SNAP Networks**
  - ca-CondMat
  - ca-GrQc
  - soc-Epinions1
  - soc-Slashdot0811
  - amazon0302
  - ca-HepTh
  - ca-HepPh
  - email-Enron
- **10th DIMACS Networks**
  - citationCiteseer
  - caidaRouterLevel
  - coAuthorsCiteseer
  - coAuthorsDBLP

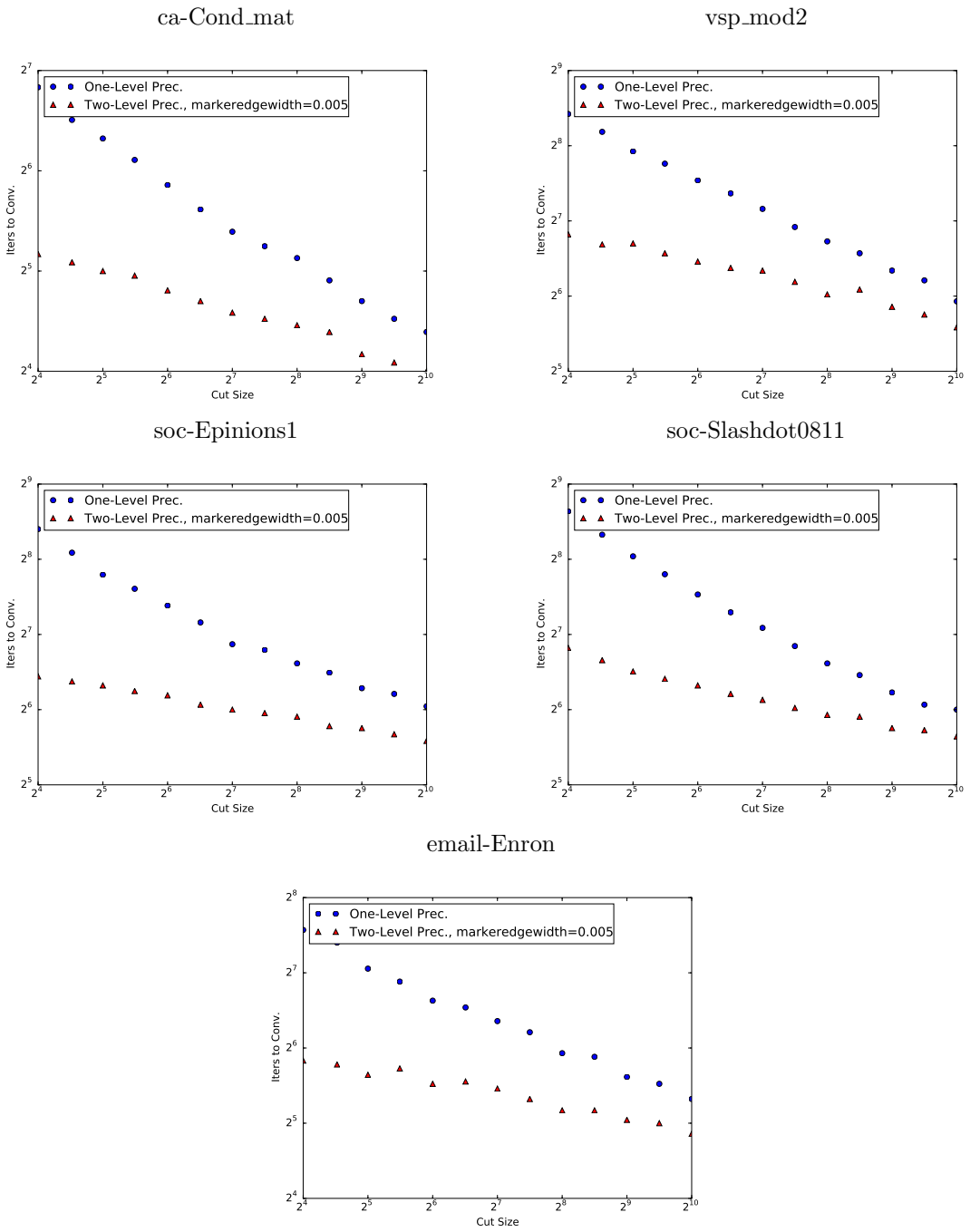


FIG. 7.4. *Maximum cut size in L_S vs. iterations required to reach a residual norm of 10^{-6} for several real-world networks. While both one-level and two-level methods improve with increasing cut size, the two-level method is less sensitive.*

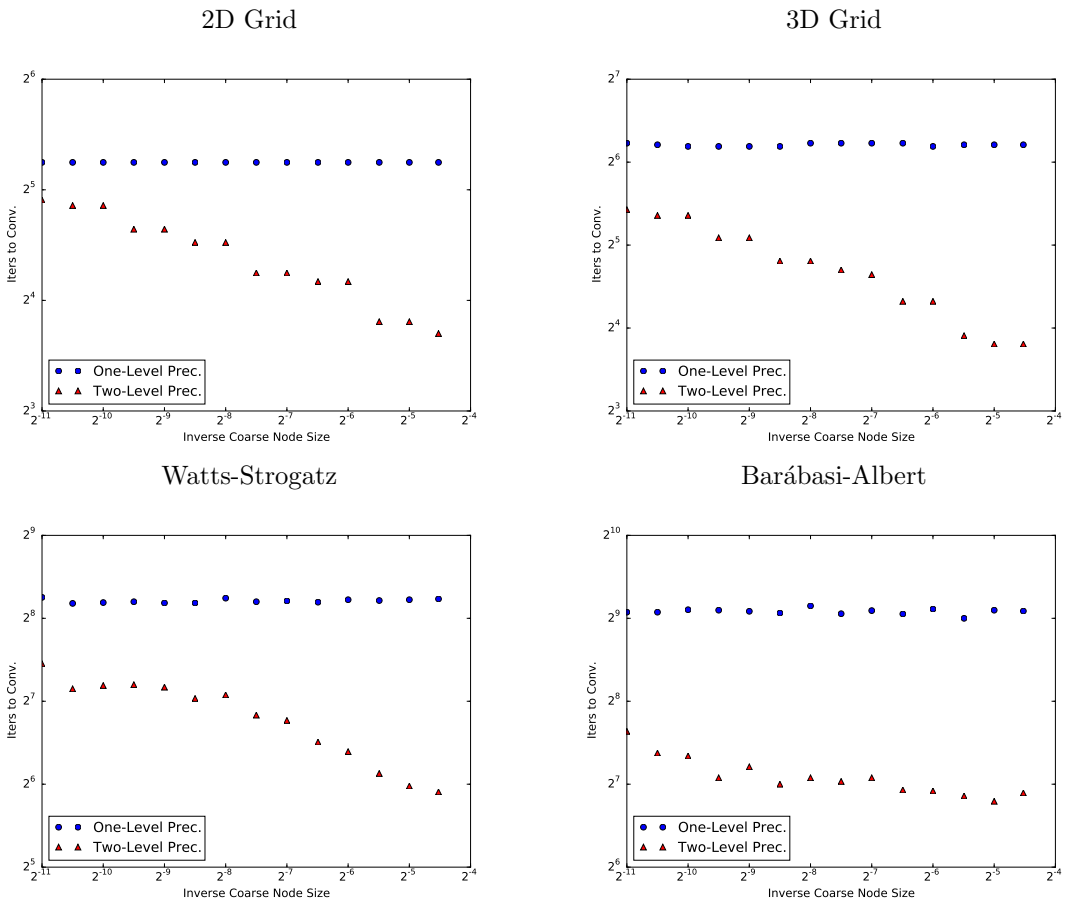


FIG. 7.5. *Inverse coarse node ($\propto n_c$) size vs. iterations required to reach a residual norm of 10^{-6} for several artificial graphs. All graphs have approximately 103,000 nodes. While most graph types show improved convergence with increasing coarse grid size (i.e. shrinking coarse node size), the Barábasi-Albert graphs show significantly improved convergence even for small coarse grids.*

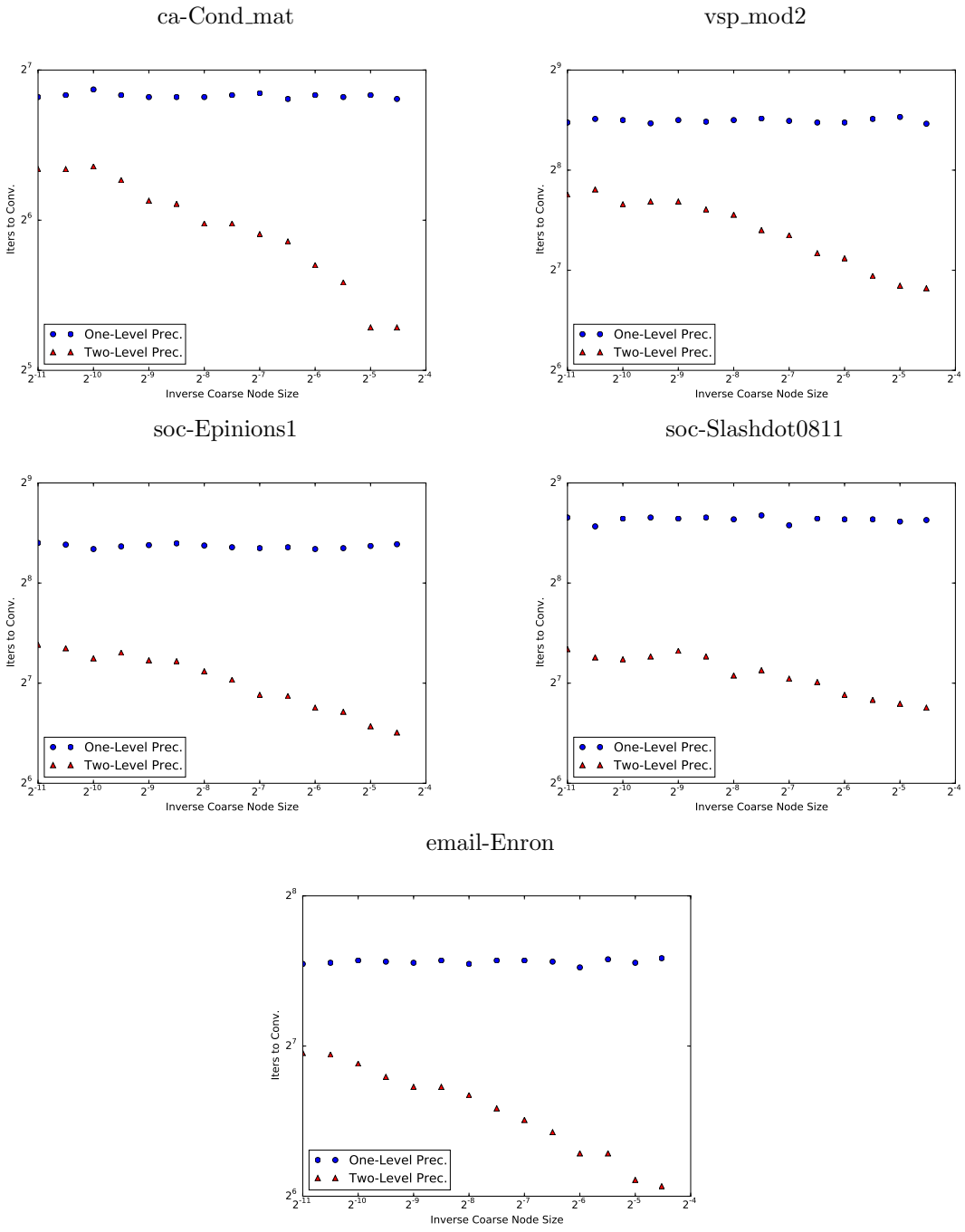


FIG. 7.6. *Inverse coarse node size vs. iterations required to reach a residual norm of* 10^{-6} *for several real-world networks. Some graphs behave like the Watts-Strogatz grahs of Figure 7.5, others behave like the Barábasi-Albert graphs.*

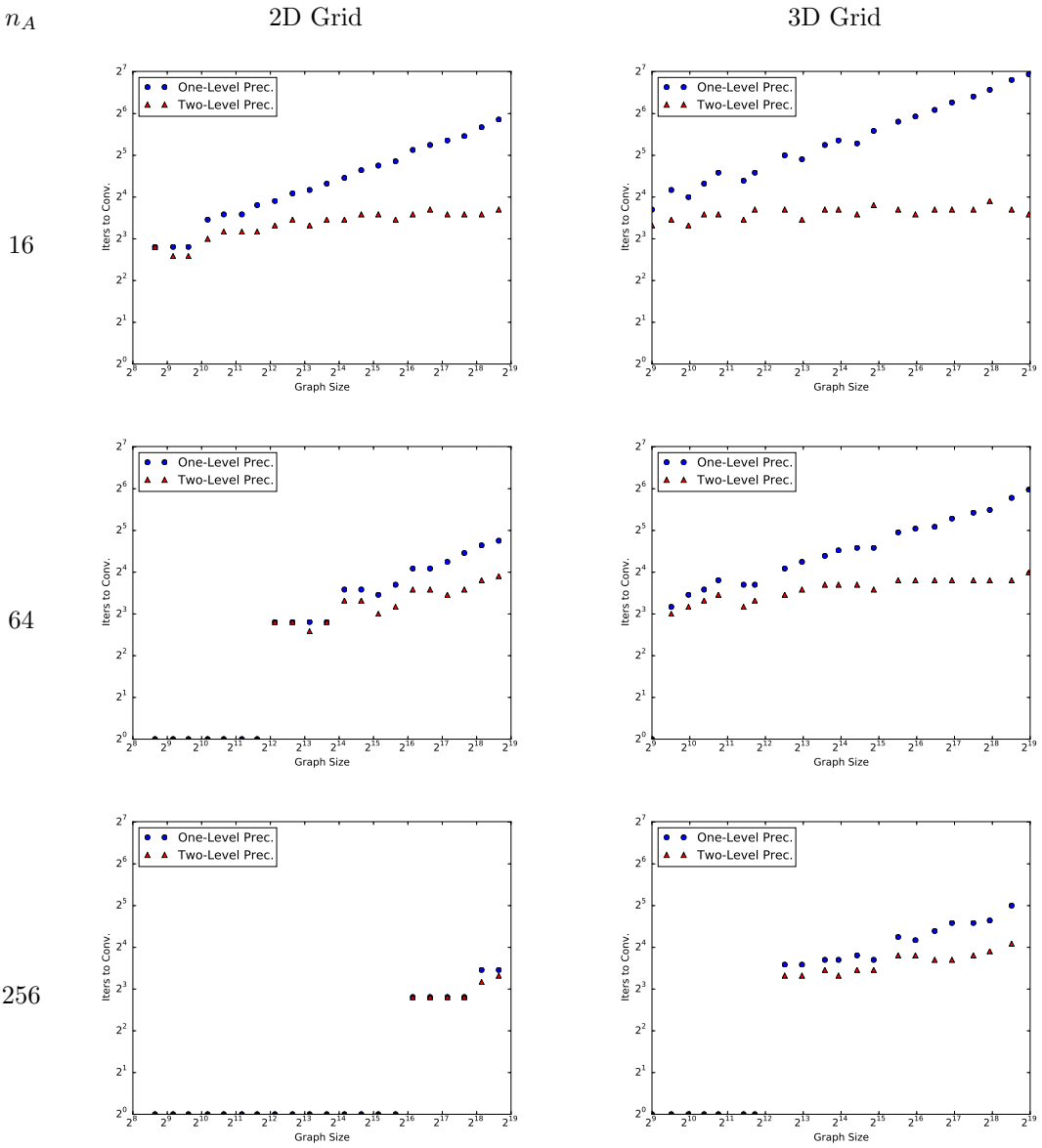


FIG. 7.7. *Iterations required to reach a residual norm of* 10^{-6} *for 2D and 3D grids using both our one-level and two-level preconditioners with maximum atomic subgraph size* n_A*, support graph cut size* n_A*, and coarse grid size* $n_c \approx n/n_A$*. After an initial rise, iteration count is independent of graph size.*

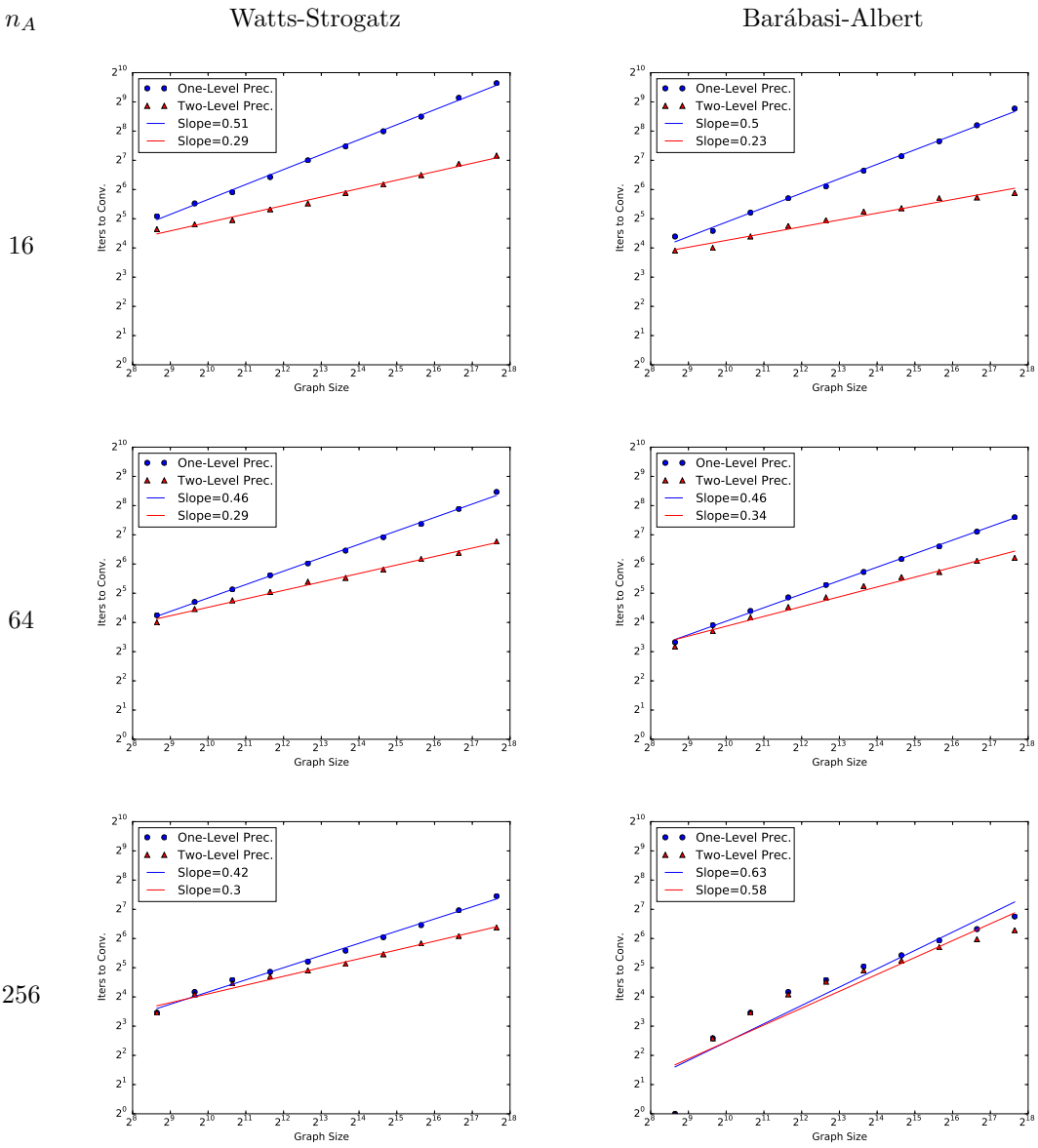


FIG. 7.8. *Iterations required to reach a residual norm of* 10^{-6} *for Watts-Strogatz and Barábasi-Albert graphs using both our one-level and two-level preconditioners with maximum atomic subgraph size* n_A*, support graph cut size* n_A*, and coarse grid size* $n_c \approx n/n_A$*. Two-grid performance on larger graphs is robust to variations in* n_A*.*