GRAPH COARSENING: FROM SCIENTIFIC COMPUTING TO MACHINE LEARNING

JIE CHEN*, YOUSEF SAAD†, AND ZECHEN ZHANG†

Abstract. The general method of graph coarsening or graph reduction has been a remarkably useful and ubiquitous tool in scientific computing and it is now just starting to have a similar impact in machine learning. The goal of this paper is to take a broad look into coarsening techniques that have been successfully deployed in scientific computing and see how similar principles are finding their way in more recent applications related to machine learning. In scientific computing, coarsening plays a central role in algebraic multigrid methods as well as the related class of multilevel incomplete LU factorizations. In machine learning, graph coarsening goes under various names, e.g., graph downsampling or graph reduction. Its goal in most cases is to replace some original graph by one which has fewer nodes, but whose structure and characteristics are similar to those of the original graph. As will be seen, a common strategy in these methods is to rely on spectral properties to define the coarse graph.

Key words. Graph Coarsening; Graphs and Networks; Coarsening; Multilevel methods; Hierarchical methods. Graph Neural Networks.

AMS subject classifications. 05C50, 05C85, 65F50, 68T09.

1. Introduction.

The idea of 'coarsening,' i.e., exploiting a smaller set in place of a larger or 'finer' set has had numerous uses across many disciplines of science and engineering. The term 'coarsening' employed here is prevalent in scientific computing, where it refers to the usage of coarse meshes to solve a given problem by, e.g., multigrid (MG) or algebraic multigrid (AMG) methods. On the other hand, the terms 'graph downsampling,' 'graph reduction,' 'hierarchical methods,' and 'pooling' are common in machine learning. Similarly, the related idea of clustering is an important tool in data-based applications. Here, the analogous term employed in scientific computing is 'partitioning.' These notions—graph partitioning, clustering, coarsening—are strongly inter-related. It is possible to use partitioning for the task of clustering data, by first building a graph that models the data which we then partition. Also, coarsening plays an important role in developing effective graph partitioning methods. Further, note that it is possible to partition a graph by just finding some clustering of the nodes, using a method from data sciences such as the K-means algorithm.

In scientific computing, the best known instance of coarsening techniques is in MG and AMG methods [58, 101, 116, 28]. Classical MG methods started with the independent works of Bakhvalov [10] and Brandt [24]. The important discovery revealed by these pioneering articles is that relaxation methods for solving linear systems tend to stall after a few steps, because they have difficulty in reducing high-frequency components of the error. Because the eigenvectors associated with a coarser mesh are direct restrictions of those on the fine mesh, the idea is to project the problem into an 'equivalent' problem on the coarse mesh for error correction and then interpolate the solution back into the fine level. This basic 2-level scheme can be extended to a multilevel one in a variety of ways. MG does not use graph coarsening specifically because it relies on a mesh and it is more natural to define a coarse mesh using processes obtained from the discretization of the physical domain. On the other hand, AMG aims at general problems that do not necessarily have a mesh associated with

*MIT-IBM Watson AI Lab, IBM Research. E-mail: chenjie@us.ibm.com.

†University of Minnesota. E-mail: {saad, zhan5260}@umn.edu. Work supported by NSF grant DMS-2011324. The authors are listed alphabetically.

them. For AMG, the graph representation of the problem at a certain level is explicitly 'coarsened' by using various mechanisms [101, 28, 102]. Since these mechanisms are geared toward a certain class of problems, essentially originating from Poisson-like partial differential equations, researchers later sought to extend AMG ideas in order to define algebraic techniques based on incomplete LU (ILU) factorizations [11, 90, 118, 6, 4, 5, 7, 9, 8, 23].

One can say that the idea of coarsening a graph in data-related applications started with the 1939 article of Kron [72], whose aim was to downsample electrical networks. Kron used his deep intuition to define coarsening techniques that rely on Schur complements, with the goal of obtaining sparse graphs. The justifications for the proposed technique were based on intuition rooted on knowledge about electrical networks. The related technique, widely known as Kron reduction, was revived by Dörfler and Bullo [46] who provided a more rigorous theoretical foundation. Later Shuman et al. [109] extended the Kron reduction into a multilevel framework. In parallel with this line of work, a number of authors developed techniques that bypassed the need to form or approximate the Schur complement relying instead on node aggregation and matching [62, 64, 74, 35, 115, 96, 95, 106].

Applications of graph coarsening in machine learning generally fall in two categories. First, coarsening is instrumental in graph embeddings. When dealing with learning tasks on graphs, it is very convenient to represent a node with a vector in $\mathbb{R}^d$ where d is small. The mapping from a node to the representing vector is termed *node (or vertex) embedding* and finding such embeddings tends to be costly. Hence, the idea is to coarsen the graph first, perform some embedding at the coarse level, and then refine-propagate the embedding back to the upper level; see [35, 74, 42, 94] for examples of such techniques. The second category of applications is when invoking *pooling* on graphs, in the context of graph neural networks (GNNs) [126, 127, 77]. However, in the latest development of GNNs, coarsening is not performed on the given graph at the outset. Instead, coarsening is part of the neural network and it is *learned* from the data. Another class of applications of coarsening is that of graph filtering, as illustrated by the articles [109, 110].

The goal of this paper is to show how the idea of coarsening has been exploited in scientific computing and how it is now emerging in machine learning. While the problems under consideration in scientific computing are fundamentally different from those of machine learning, the basic ingredients used in both methods are striking by their similarity. The paper starts with a discussion of graph coarsening in scientific computing (Section 2), followed by a section on graph coarsening in machine learning (Section 3). We also present some newly developed coarsening methods and results, in the context of machine learning, in Sections 4–5.

1.1. Notation and preliminaries. We denote by $G = (V, E)$ a graph with n nodes and m edges, where V is the node set and E is the edge set. The weights of the edges of G are stored in a matrix A, so a_{ij} is the weight of the edge $(i, j) \in E$. In most cases we will assume that the graph is undirected. We sometimes use $G = (V, E, A)$ to denote the graph, when A is emphasized.

The sum of row i of A is called the degree of node i and the diagonal matrix of the degrees is called the degree matrix:

$$d_i = \sum_{j=1}^{n} a_{ij}; \quad D = \text{Diag}\{d_i\}. \tag{1.1}$$

With this notation, the graph Laplacian matrix L is defined as:

$$L = D - A. \tag{1.2}$$

This definition implies that $L\mathbf{1} = 0$ where $\mathbf{1}$ is the vector of all ones, i.e., $\mathbf{1}$ is an eigenvector associated with the eigenvalue zero. In the simplest case, each weight a_{ij} is either zero (not adjacent) or one (adjacent). A simple, yet very useful, property of graph Laplacians is that for any vector x, we have the relation

$$x^T L x = \sum_{ij} a_{ij} |x_i - x_j|^2 \ . \tag{1.3}$$

The normalized Laplacian is defined as follows:

$$\widehat{L} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2} \ . \tag{1.4}$$

Note that the diagonal entries of $\widehat{L}$ are all ones. The matrix is again singular and has the null vector $D^{1/2}\mathbf{1}$.

We will also make use of the incidence matrix denoted by $B \in \mathbb{R}^{n \times m}$. A column b_e of B represents an edge $e \in E$ between nodes i and j with weight a_{ij}, and its k-th entry is defined as follows:

$$b_e(k) = \begin{cases} +\sqrt{a_{ij}}, & k = i, \\ -\sqrt{a_{ij}}, & k = j, \\ 0, & \text{otherwise.} \end{cases} \tag{1.5}$$

Note that the two nonzero values of $b_e(k)$ have opposite signs, but we have a choice regarding which of i and j is assigned the negative sign. Unless otherwise specified, we simply assign the negative sign to the smaller of i and j. As is well-known, the graph Laplacian can be defined from the incidence matrix through the relation $L = BB^T$.

1.2. Terminology and notation specific to machine learning. In data-related applications, graph nodes are often equipped with feature vectors and labels. We use an $n \times d$ matrix X to denote the feature matrix, whose i-th row is the feature vector of node i. We use an $n \times c$ matrix Y to denote the label matrix, where c is the number of categories. When a node i belongs to category j, $Y_{ij} = 1$ while $Y_{ij'} = 0$ for all $j' \neq j$. Each row of Y is called a 'one-hot' vector. When there are only two categories, the $n \times 2$ matrix Y can equivalently be represented by an $n \times 1$ binary vector y in a straightforward manner.

For example, in a transaction graph, where nodes represent account holders and edges denote transactions between accounts, a node may have $d = 4$ features: account balance, account active days, number of incoming transactions, and number of outgoing transactions; as well as $c = 3$ categories: individual, non-financial institution, and financial institution. A typical task is to predict the account category given the features.

The feature matrix X provides complementary information to a graph $G = (V, E, A)$ that captures relations between data items. One should not confuse the feature matrix with a data matrix, which is often used in the context where a graph structure is not given, but it may be constructed based on the information of the data items. The notation of a data matrix by convention clashes with X; for the moment let us use Z instead to denote it, whose i-th row is denoted by z_i. One may construct

Table 1.1: Commonly used notations.

Notations	Descriptions
G	A graph.
V	The set of nodes in a graph.
E	The set of edges in a graph.
A	The graph adjacency matrix.
a_{ij}	The weight of an edge $(i,j) \in E$.
D	The degree matrix.
L	The graph Laplacian; $L = D - A$.
$L^\dagger$	The pseudoinverse of the graph Laplacian.
S	The Schur complement matrix.
n	The number of nodes in G; $\lvert V\rvert = n$.
m	The number of edges in G; $\lvert E\rvert = m$.
d	The dimension of a node feature vector.
c	The dimension of a label vector/number of classes.
G_c	A coarse graph.
L_c	The coarse graph Laplacian.
n_c	The number of nodes in G_c; $\lvert V_c\rvert = n_c$.
ℓ	The number of coarsening levels.
(λ, u)	Eigenpairs.
$P \in \mathbb{R}^{n\times n_c}$	The interpolation operator.
$P^T \in \mathbb{R}^{n_c\times n}$	The restriction/coarsening operator.
$X \in \mathbb{R}^{n\times d}$	The node feature matrix.
$Y \in \mathbb{R}^{n\times c}$	The ground-truth label matrix.
W	The trainable parameters of a graph neural network.

a graph G from Z. For example, in a k-nearest neighbors (kNN) graph, there is an edge from node i to node j if and only if j is an index of the element among the k smallest elements of $\{r_{ij} = \|z_j - z_i\| \mid j \neq i\}$. One may even define the weighted adjacency matrix A as $a_{ij} = e^{-r_{ij}}$ when there is an edge (i,j) and $a_{ij} = 0$ otherwise. In this case, the constructed graph is entirely decided by the data matrix Z, rather than by holding complementary information to it, as is done with the feature matrix.

2. Graph coarsening in scientific computing. Given a graph $G = (V, E)$, the goal of graph coarsening is to find a smaller graph $G_c = (V_c, E_c)$ with n_c nodes and m_c edges, where $n_c < n$, which is a good approximation of G in some sense. Specifically, we would like the coarse graph to provide a faithful representation of the structure of the original graph. We denote the adjacency matrix of G_c by A_c and the graph Laplacian of G_c by L_c.

We will first elaborate on one of the most important scenarios that invoke coarsening (Section 2.1) and then discuss several representative approaches to it (Sections 2.2 to 2.5). Note that in practice, coarsening often proceeds recursively on the resulting graphs; by doing so, we obtain a hierarchy of approximations to the original graph.

2.1. Multilevel methods for linear systems: AMG and multilevel ILU. Graph coarsening strategies are usually invoked when solving linear systems of equations, by multilevel methods such as (A)MG [58, 116, 101] or Schur-based multilevel techniques [37, 105, 90, 80, 79, 11, 7]. In (A)MG, this amounts to selecting a subset

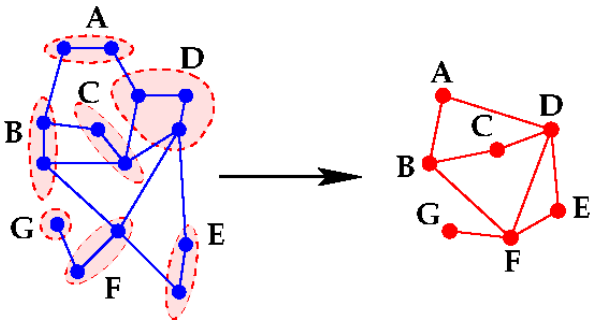


Fig. 2.1: Coarsening a graph.

of the original (fine) grid, known as the 'coarse grid.' In AMG, the selection of coarse nodes is made in a number of different ways. The classical Ruge-Stüben strategy [101] selects coarse nodes based on the number of 'strong connections' that a node has. Here, nodes i and j are strongly connected if a_{ij} has a large magnitude relative to other nonzero off-diagonal elements or row i. The net effect of this strategy is that each fine node is strongly coupled with the coarse set. In other methods, the strength of connection is defined from the speed with which components of a relaxation scheme for solving the homogeneous system $Au = 0$ converge to zero, see [25, 98, 36] and Section 2.4 for additional details. For multilevel Schur-based methods, such as multilevel ILU, the coarsening strategy may correspond to selecting from the adjacency graph of the original matrix, a subset of nodes that form an independent set [105], or a subset of nodes that satisfy good diagonal dominance properties [103] or that limit the growth in the inverse LU factors of the ILU factorization [21, 22].

The coarsening strategy can be expanded into a multilevel framework by repeating the process described above on the graph associated with the nodes in the coarse set. Let G_0 be the original graph G and let $G_1, G_2, \ldots, G_h$ be a sequence of coarse graphs such that $G_\ell = (V_\ell, E_\ell)$ is obtained by coarsening on $G_{\ell-1}$ for $1 \leq \ell < h$. Let $A^{(0)} \equiv A$ and $A^{(\ell)}$ be the matrix associated with the ℓ-th level. The graph G_ℓ admits a splitting into coarse nodes, C_ℓ, and fine nodes, F_ℓ, so that the linear system at the ℓ-th level, which consists of the matrix $A^{(\ell)}$ and the right-hand side $f^{(\ell)}$ can be reordered as follows:

$$A^{(\ell)} = \begin{bmatrix} A^{(\ell)}_{CC} & A^{(\ell)}_{CF} \\ A^{(\ell)}_{FC} & A^{(\ell)}_{FF} \end{bmatrix}, \quad f^{(\ell)} = \begin{bmatrix} f^{(\ell)}_C \\ f^{(\ell)}_F \end{bmatrix}. \tag{2.1}$$

Note that it is also possible to list the fine nodes first followed by the coarse nodes; see [90]. The coarser-level graph $G_{\ell+1}$ as well as the new system consisting of the matrix $A^{(\ell+1)}$ and the right-hand side $f^{(\ell+1)}$ at the next level, are constructed from G_ℓ and $A^{(\ell)}$. These are built in a number of different ways depending on the method under consideration. For the graph, we can for example set two coarse nodes to be adjacent in $G_{\ell+1}$, if their representative children are adjacent in G_ℓ. One common way to do this is to define two coarse nodes to be adjacent in $G_{\ell+1}$ if they are parents of adjacent nodes in G_ℓ. Next we discuss coarsening in the specific context of AMG.

2.1.1. Algebraic multigrid. AMG techniques are all about generalizing the *interpolation* and *restriction* operations of standard MG. The coarsening process identifies for each fine node a set of nearest neighbors from the coarse set. Using various

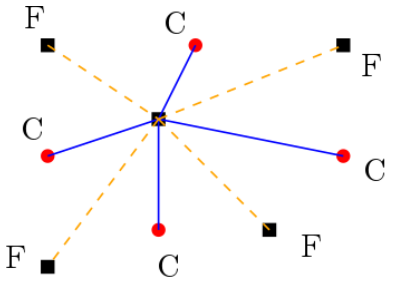


Fig. 2.2: A fine node and its nearest neighbors. Fine nodes are represented by a square and coarse ones by a disk. In the coarsening process the central, fine node is expressed as a combination of its coarse neighbors.

arguments on the strength of connection between nodes, AMG expresses a fine node i as a linear combination of a selected number of nearest neighbors that form a set C_i, see Figure 2.2. To simplify notation, we consider only one level of coarsening and drop the subscript ℓ.

If C is the set of coarse nodes and F is the set of fine nodes, we can define related subspaces $\mathcal{X}_C$ and $\mathcal{X}_F$ of the original space $\mathcal{X} = \mathbb{R}^n$. In fact we can write $\mathcal{X} = \mathcal{X}_C \oplus \mathcal{X}_F$. Then, given a vector x with components in the coarse space $\mathcal{X}_C$, we associate a vector Px in the original space $\mathcal{X}$, whose i-th component is defined as follows [104, 13.6.2]:

$$[Px]_i = \begin{cases} x_i & \text{if } i \in C, \\ \sum_{j \in C_i} p_{ij} x_j & \text{otherwise.} \end{cases} \tag{2.2}$$

The mapping P sends a point of $\mathcal{X}_C$ into a point $y = Px$ of $\mathcal{X}$. The value of y at a coarse point, a node in C, is the same as its starting value. The value at another node, one in F, is defined from interpolated values at a few coarse points. Thus, P is known as the *interpolation* operator.

The transpose of P represents the *restriction*, or coarsening mapping. In the context of AMG, it projects a point in $\mathcal{X}$ into a point in $\mathcal{X}_C$. Each node in C is a linear combination of nodes of the original graph.

If we now return to the multilevel case where P_ℓ denotes a corresponding interpolation operator at the ℓ-th level, then AMG defines the linear system at the next level using *Galerkin projection,* where the matrix and right-hand side are, respectively,

$$A^{(\ell+1)} = P_\ell^T A^{(\ell)} P_\ell, \quad f^{(\ell+1)} = P_\ell^T f^{(\ell)} . \tag{2.3}$$

Recall that we started with the original system $A^{(0)}x = f^{(0)}$, which corresponds to $\ell = 0$. AMG methods rely on a wide variety of iterative procedures that consist of exploiting different levels for building an approximate solution. It is important to note here that *the whole AMG scheme depends entirely on defining a sequence of interpolation operators P_ℓ for $\ell = 0, 1, \ldots$* Once the P_ℓ's are defined, one can design various 'cycles' in which the process goes back and forth from the finest level to the coarsest one in an iterative procedure.

When defining the interpolation operator P_ℓ, there are two possible extremes worth noting, even though these extremes are not used in AMG in practice. On the one end, we find the *trivial interpolation* in which the p_{ij}'s in equation (2.2) are all set to zero. In this case, referring to (2.1), $A^{(\ell+1)}$ is simply $A^{(\ell+1)} = A_{CC}^{(\ell)}$.

The other extreme is the *perfect interpolation* case which yields the Schur complement system. Here the interpolation operator is

$$P_\ell = \begin{bmatrix} -[A_{CC}^{(\ell)}]^{-1} A_{CF}^{(\ell)} \\ I \end{bmatrix}.$$

The right reduced matrix $A^{(\ell)} P_\ell$ involves the Schur complement matrix associated with the coarse block:

$$A^{(\ell)} P_\ell = \begin{bmatrix} O \\ S_\ell \end{bmatrix} \quad \text{where} \quad S_\ell = A_{FF}^{(\ell)} - A_{FC}^{(\ell)} [A_{CC}^{(\ell)}]^{-1} A_{CF}^{(\ell)}.$$

We also clearly have $A^{(\ell+1)} = P_\ell^T A^{(\ell)} P_\ell = S_\ell$. The exact solution of (2.1) can be written in the form $[u_C^{(\ell)}; u_F^{(\ell)}]$ were matlab notation is used, i.e., $[x; y]$ is the concatenation of the vector x followed by the vector y. Then, if we denote by $w_C^{(\ell)}$ the coarse solution $w_C^{(\ell)} = [A_{CC}^{(\ell)}]^{-1} f_C^{(\ell)}$, it can be seen that $u_F^{(\ell)}$ is the solution of the Schur complement system $S_\ell u_F^{(\ell)} = f_F^{(\ell)} - A_{FC}^{(\ell)} w_C^{(\ell)}$. In addition, once $u_F^{(\ell)}$ has been computed, the whole solution of (2.1) can be perfectly reconstructed via substitution since we have:

$$\begin{bmatrix} u_C^{(\ell)} \\ u_F^{(\ell)} \end{bmatrix} = \begin{bmatrix} w_C^{(\ell)} \\ 0 \end{bmatrix} + P_\ell u_F^{(\ell)}.$$

This approach is nothing but a block form of Gaussian elimination and it is generally costly although there are practical alternatives discussed in the literature [21, 118, 11, 82] that are based on Schur complements. However, it is worth pointing out that, viewed from this angle, *the goal of all AMG methods is essentially to find inexpensive approximations to the Schur complement system.*

2.1.2. Multilevel ILU preconditioners based on coarsening. The issue of finding a good ordering for ILU generated a great deal of research interest in the past; see, e.g., [18, 19, 17, 26, 39, 45, 40, 27, 103, 38]. A class of techniques presented in [90] consisted of preprocessing the linear system with an ordering based on coarsening. Thus, for a one-level ordering the matrix is ordered as shown in (2.1). Then in a second level coarsening, $A_{22}^{(0)}$ is in turn reordered and we end up with a matrix like:[1]

$$\left[\begin{array}{c|cc} A_{11}^{(0)} & \multicolumn{2}{c}{A_{12}^{(0)}} \\ \hline A_{21}^{(0)} & A_{11}^{(1)} & A_{12}^{(1)} \\ & \hline A_{21}^{(1)} & A_{22}^{(1)} \end{array}\right].$$

This is repeated with $A_{22}^{(1)}$ and further down for a few levels. Then the idea is simply to perform an ILU factorization of the resulting reordered system. Next, we describe a method based on this general approach.

The first ingredient of the method is to define a weight w_{ij} for each nonzero pair (i, j). This will set an order in which to visit the edges of the graph. The strategies described next are 'static' in that given a certain matrix A (one of the $A^{(\ell)}$'s), these weights are precomputed, in contrast with dynamic ones used in, e.g., [105, 78, 103]. If nnz_i is the number of nonzero entries of row i and nnz_j is the number of nonzero

[1] For notational simplicity, for the subscripts of A we use 1 in place of C and 2 in place of F.

entries of column j, we define the weights as follows:

$$w_{ij} = \min\left\{ \frac{|a_{ij}|}{\delta_r(i)} \ , \ \frac{|a_{ij}|}{\delta_c(j)} \right\} \text{ where:} \tag{2.4}$$

$$\delta_r(i) = \frac{\|A_{i,:}\|_1}{nnz_i} \quad \text{and} \quad \delta_c(j) = \frac{\|A_{:,j}\|_1}{nnz_j} \tag{2.5}$$

where matlab notation is exploited and $\|\cdot\|_1$ is the usual 1-norm. The two terms in the brackets of (2.4) represent the importance of $|a_{ij}|$ relative to the other elements in the same row and column, respectively. If our goal is to put large entries in the (1,2) block of the matrix when it is permuted (block $A_{CF}^{(\ell)}$ in Equation (2.1)), then we need to traverse the graph starting from the largest to smallest w_{ij}.

The above defines an order in which to visit edges. Next, each time an edge (i, j) is visited we need to determine which one of i and j will be selected as a coarse node. This requires a 'preference' measure, or weight, for each node. When $a_{kk} \neq 0$ we define the impact of 'pivot' k as the average potential fill-in created when eliminating unknown k. In the formula $a_{ij} = a_{ij} - a_{ik} \times a_{kj}/a_{kk}$ employed in the k-th step of Gaussian elimination, the term $-a_{ik} \times a_{kj}/a_{kk}$ is a potential fill-in. This is a very crude approximation because it assumes that the entries have not changed. We define the 'impact' of the diagonal entry k as the inverse of the quantity:

$$\phi_k = \frac{|a_{kk}|}{\delta_r(k)\delta_c(k)} \ . \tag{2.6}$$

When visiting edge (k, l), we add k to the coarse set if $\phi_k > \phi_l$ and l otherwise.

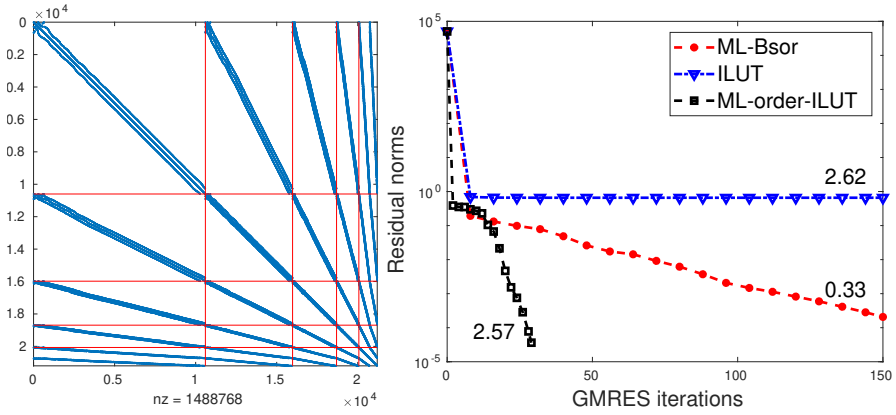


Fig. 2.3: Left: The matrix Raefsky3 after the reordering obtained from four levels of coarsening. Right: Performance of various coarsening based preconditioners for solving a linear system with the matrix.

Here, we show an example on the matrix 'Raefsky3', which is of size 21,200 and has 1,488,768 nonzero elements. It comes from a fluid structure interaction turbulence problem and can be obtained from the suite-sparse collection[2]. Figure 2.3 (left) shows the pattern of the reordered matrix according to the coarsening strategy described

[2] https://sparse.tamu.edu/

above, using four levels of coarsening. The original matrix is not shown but, as expected, its pattern is very similar to that of the (1,1) block of the reordered matrix, and has roughly twice the size.

For this particular matrix, standard ILU-based strategies perform poorly. Using matlab, we applied GMRES(50) to the system, preconditioned with ILU ('crout' version) with a drop tolerance of 0.01. The resulting iterates stagnate as indicated by the top curve in the right side of Figure 2.3. The number 2.62 is the 'fill-factor,' which is the ratio of the number of nonzero elements of the LU factors over the the number of nonzero elements of the original matrix. When comparing preconditioners of this type, we strive to ensure that these fill-factors are about the same for the preconditioners being compared. Next we perform an ILU factorization ('crout' version again) on the reordered system using the coarsening described above. In order to achieve a fill-factor similar to that of the standard factorization we lowered the drop-tolerance to 0.0008. The new method converged in 29 iterations with a fill factor of 2.57. We also tested a more traditional preconditioner based on block Gauss-Seidel, exploiting the block structure shown on the left side of Figure 2.3. Each block Gauss-Seidel step requires solving a system with the diagonal blocks of the reordered matrix. These systems are approximately solved using a simple ILU(0) ('nofill') factorization. Note that the fill-factor here is very low (0.33). Each preconditioning step consists of 10 Gauss-Seidel iterations. As is shown, this also converges, although more slowly.

2.2. Coarsening approach: Pairwise aggregation. The broad class of 'pairwise aggregation' techniques, e.g., [120, 119, 85, 89, 37, 116], is a strategy that seeks to simply coalesce two adjacent nodes in a graph into a single node, based on some measure of nearness or similarity. The technique is based on edge collapsing [63], which is a well known method in the multilevel graph partitioning literature. In this method, the collapsing edges are usually selected using the *maximal matching* method. A *matching* of a graph $G = (V, E)$ is a set of edges $\widetilde{E}$, $\widetilde{E} \subseteq E$, such that no two edges in $\widetilde{E}$ have a node in common. A maximal matching is a matching that cannot be augmented by additional edges to obtain a larger matching in the sense of inclusion. Coarsening schemes based on *edge matching* have been in use in the AMG literature for decades [101]. For each node i, a coarsening algorithm starts from building a set S_i of nodes that are 'strongly connected' to i by using some measure of connection strength. The graph nodes are traversed in a certain order of preference and the next unmarked node in this order, say j, is selected as a *coarse* node. The priority measure of the traversal is updated after each insertion of a coarse node. There are a number of ways to find a maximal matching for coarsening a graph.

The *heavy-edge matching* (HEM) approach, e.g., [67], is a greedy matching algorithm that works with the weight matrix A of the graph. It simply matches a node i with its largest off-diagonal neighbor j_{max}; i.e., we have $|a_{ij_{max}}| = \max_{j \in adj(i), j \neq i} |a_{ij}|$, where $adj(i)$ denotes the adjacency (or nearest-neighbor) set of node i. When selecting the largest neighboring entry, ties are broken arbitrarily. If j_{max} is already matched with some node $k \neq i$ seen before, i.e., $p(k) = j_{max}$, then node i is left unmatched and considered as a singleton. Otherwise, we match i with j and the result is $p(i) = j_{max}$.

A version of HEM is shown in Algorithm 2.1, modified from [90]. The algorithm proceeds by exploiting a greedy approach. It scans all edges (i, j) in decreasing value of their weight a_{ij}. If neither i nor j has defined parents, it creates a new coarse node labeled *new* and sets the parents of i and j to be *new*. After the loop is completed, there will be singletons; i.e., node that have not been assigned a parent in the loop. As shown in lines 10–16, a 'singleton' node is either added as a coarse nodes, if it

Algorithm 2.1 Heavy Edge Matching (HEM)

Input: Weighted graph $G = (V, E, A)$
Output: Coarse nodes; $Prnt$ list
Init: $Prnt(i) = 0\ \forall i \in V$; $new = 0$
for max **to** min edge (i, j) **do**
    if $Prnt(i) == 0$, $Prnt(j) == 0$ **then**
        $new = new + 1$
        $Prnt(i) = Prnt(j) = new$
    end if
end for
for Node v with $Prnt(v) == 0$ **do**
    if v has no neighbor **then**
        $new = new + 1$; $Prnt(v) = new$
    else
        $Prnt(v) = Prnt(j)$ where $j = \text{argmax}_i(a_{iv})$
    end if
end for

is disconnected ('real singleton'), or it is lumped as a child of an already generated coarse node ('left-over singleton'). Figure 2.4 gives an illustration of this step.

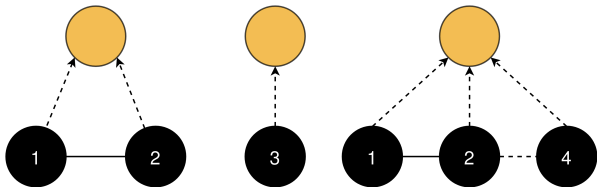


Fig. 2.4: The coarsening process. Original and coarse nodes are colored in black and yellow, respectively. Dashed arrow indicates parent-child relationship. Solid line represents heaviest-weighted edge. *Left:* A coarse node is created from two adjacent nodes 1 and 2. *Middle*: A coarse node is created from a true singleton node 3. *Right*: A left-over singleton node 4 is attached to a coarse, nearest neighbor node 2.

2.3. Coarsening approach: Independent sets. It is also possible to exploit independent sets (see, e.g., [96]) for coarsening graphs. Recall that an independent set $\mathcal{S}$ is a subset of V that consists of nodes that are not adjacent to each other; i.e., no pair (v, w) where $v, w \in \mathcal{S}$ is linked by an edge. An independent set $\mathcal{S}$ is maximal if no (strict) superset of $\mathcal{S}$ forms another independent set. One can use the nodes of a carefully selected independent set to form a coarse graph; i.e., we can define $V_c = \mathcal{S}$. Then, it is relatively easy to determine the edges and weights between these nodes by using information from the original graph. For example, in [96] an edge is inserted between v and w of V_c if there is at least one node y in $V \backslash V_c$ such that $(v, y) \in E$ and $(w, y) \in E$. This will produce the edge set E_c needed to form the coarse graph G_c.

Let L be the graph Laplacian, reordered such that the n_c nodes of V_c are listed

first. Then L will have the following structure, where $D_c \in \mathbb{R}^{n_c \times n_c}$:

$$L = \begin{pmatrix} D_c & -F \\ -F^T & B \end{pmatrix}. \tag{2.7}$$

Since the nodes associated with the (1,1) block in the above matrix form an independent set, it is clear that the matrix D_c is diagonal. Coarsening by independent sets will consist of taking the matrix D_c and adding off diagonal elements to it to obtain the adjacency matrix A_c. An important observation to be made here is that the edges added by the independent set coarsening are those obtained from the Schur complement with respect to B, *where B is replaced by a diagonal matrix*. Let us assume that B is replaced by a matrix D_f. The resulting Schur complement is

$$S_c = D_c - F D_f^{-1} F^T. \tag{2.8}$$

The nonzero pattern of S_c is the same as that of FF^T, which is in turn the sum of the patters of $f_j f_j^T$ where f_j is the j-th column of F. If f_j has nonzero entries in locations i and k, then we will have nonzero entries in the positions $(i,i),(i,k),(k,i)$ and (k,k) of S_c. Next, we will define D_f more specifically. Let D_f be the diagonal of row-sums of F^T and assume for now that these are all nonzero. This is the same diagonal as the one used for the graph Laplacian except that the summation ignores the entries a_{ij} when i and j are both in $V \backslash V_c$. In matlab notation: $D_f = \text{diag}(\text{diag} F^T \mathbf{1})$. In this situation, S_c becomes a Laplacian.

PROPOSITION 2.1. *Let B be replaced by D_f, defined as the diagonal of the row-sums of F^T. Then D_f is invertible. Let $L_c = D_c - F D_f^{-1} F^T$. Then the graph of L_c is G_c, the graph of the independent set coarsening of G. In addition, L_c is a graph Laplacian; specifically, it is the graph Laplacian of G_c.*

Proof. Because the independent set is maximal, we cannot have a zero diagonal element in D_f. Indeed, if the opposite was true, then one row, say row k, of F^T would be zero. This would mean that we could add node k to the independent set, because it is not coupled with any element of $\mathcal{S}$. The result would be another independent set that includes $\mathcal{S}$, contradicting maximality.

It was shown above that the adjacency graph of FF^T, which is the same as that of L_c, is exactly G_c. It is left to show that L_c is a Laplacian. Since D_f and F have nonnegative entries, the off-diagonal of L_c are clearly negative. Next, note that $F^T \mathbf{1} = D_f \mathbf{1}$ and hence

$$(D_c - F D_f^{-1} F^T)\mathbf{1} = D_c \mathbf{1} - F D_f^{-1} D_f \mathbf{1} = (D_c - F)\mathbf{1} = 0.$$

Thus, L_c is indeed a Laplacian. ∎

2.4. Coarsening approach: Algebraic distance. Researchers in AMG methods defined a notion of 'algebraic distance' between nodes based on relaxation procedures. This notion is motivated by the bootstrap AMG (BAMG) method [25] for solving linear systems. AMG creates a coarse problem by trying to exploit some rules of 'closeness' between variables. In BAMG, this notion of closeness is defined from running a few steps of Gauss-Seidel relaxations, starting with some random initial guess for solving the related homogeneous system $Ax = 0$. The speed of convergence of the iterate determines the closeness between variables. This is exploited to aggregate the unknowns and define restriction and interpolation operators [98]. In [36] this general idea was extended for use on graph Laplacians. In the referenced paper, Gauss-Seidel is replaced by Jacobi overrelaxation.

Algorithm 2.2 Algebraic distances for graphs

1: **Input:** Parameter ω, weighted graph $G = (V, E, A)$, initial vector $x^{(0)}$, number of steps k.
2: **Output:** Distances $s_{ij}^{(k)}$ for each pair (i, j).
3: **for** $j = 1, 2, \cdots, k$ **do**
4: $\quad x^{(j)} = (1-\omega)x^{(j-1)} + \omega D^{-1}Ax^{(j-1)}$
5: **end for**
6: Set: $s_{ij}^{(k)} = |x_i^{(k)} - x_j^{(k)}|\ \forall i, j$

Algorithm 2.2 shows how these distances are calculated. They depend on two parameters: the over relaxation parameter ω and the number of steps, k. It can be shown that the distances $s_{ij}^{(k)}$ converge to zero [36] as $k \to \infty$. However, it was argued in [36] that the speed of decay of $s_{ij}^{(k)}$ is an indicator of relative strength of connection between i and j. In other words, the important measure is the magnitude – in relative terms – of $s_{ij}^{(k)}$ for different (i, j) pairs.

Note that the iteration is of the form $x^{(j)} = Hx^{(j-1)}$, where H is the iteration matrix

$$H = (1-\omega)I + \omega D^{-1}A = I - \omega(I - D^{-1}A).$$

Because $D - A$ is a graph Laplacian, the largest eigenvalue of H is $\lambda_1 = 1$. It is then suggested to scale these scalars by λ_2^k, where λ_2 is the second largest eigenvalue in modulus. In general, it is sufficient to iterate for a few steps and stop at a step k when one observes the scaled quantities start to settle.

As can be seen, a coarsening method based on algebraic distances is rather different from the previous two methods. Instead of working on the graph directly we now use our intuition on the iteration matrix to extract intuitive information on what may be termed a relative distance between variables. If two variables are close with respect to this distance, they may be aggregated or merged.

Ultimately, as was shown in [36], what is important is the decay of the component of the vector $x^{(k)} - x^{(k-1)}$ in the second eigenvector. This distance between two vectors is indeed dominated by the component in the second eigenvector.

This brings up the question as to whether or not we can directly examine spectral information and infer from it a notion of distance on nodes. Spectral graph coarsening addresses this and will be examined in Section 4.

2.5. Techniques related to coarsening. Graph coarsening is a *graph reduction* technique, in the sense that it aims at reducing the size of the original graph while attempting to preserve its properties. There exist a number of other techniques in the same category. These include *graph summarization* [86, 75], *graph compression* [51], and *graph sketching* [71]. These are more common in machine learning and the tasks they address are specific to the underlying applications. In the following we discuss methods that are more akin to standard coarsening methods.

2.5.1. Graph reduction: Kron. The Kron reduction of networks was proposed back in 1939 [72], as a means to obtain lower dimensional electrically equivalent circuits in circuit theory. Its popularity gained momentum across fields after the appearance of a thorough analysis of the method in [46]. The method starts with a weighted graph $G = (V, E, A)$ and the associated graph Laplacian L, along with a set V_1 of nodes which is a strict subset of V. Such a subset can be obtained by

downsampling, for example, although in the original application of circuits it is a set of nodes at the boundary of the circuit. The method essentially defines a coarse graph from the Schur complement of the original adjacency graph with respect to this downsampled set.

The goal is to form a reduced graph on V_1. This is viewed from the angle of Laplacians. If we order the nodes of V_1 first, followed by those of the complement set to V_1 in V, the Laplacian can be written in block form as follows:

$$L = \begin{bmatrix} L_{11} & L_{12} \\ L_{12}^T & L_{22} \end{bmatrix}. \tag{2.9}$$

The Kron reduction of L is defined as the Schur complement of the original Laplacian relative to L_{22}; i.e.,

$$L(V_1) = L_{11} - L_{12} L_{22}^{-1} L_{12}^T. \tag{2.10}$$

This turns out to be a proper graph Laplacian as was proved in [46], along with a few other properties. We can therefore associate a set of weights $A_{ij}^{(1)}$ for the reduced graph, defined from $L(V_1)$:

$$a_{ij}^{(1)} = \begin{cases} -[L(V_1)]_{ij} & \text{if } i \neq j, \\ 0 & \text{otherwise.} \end{cases}$$

An example is shown in Figure 2.5.

A multiscale version of the Kron reduction, called the *pyramid transform*, was proposed in [109], specifically for applications that involve signal processing on graphs [110]. It was developed as a multiscale (i.e., 'multilevel' in the scientific computing jargon) extension of a similar scheme invented in the late 1980s for image processing [31]. The extension is from regular data (discrete time signals, images) to irregular data (graphs, networks) as well as from one level to multiple levels.

An original feature of the paper [109] is the use of spectral information for coarsening the graph. Specifically, a departure from traditional coarsening methods such as those described in Sections 2.2 and 2.3 is that the separation into coarse and fine nodes is obtained from the 'polarity' (i.e., the sign of the entries) of the eigenvector associated with the largest eigenvalue of the graph Laplacian. The motivation of the authors is a theorem by Roth [99], which deals with bipartite graphs. The idea of exploiting spectral information was exploited earlier by Aspvall and Gilbert [3] for the problem of graph coloring, an important ingredient of many linear algebra techniques. Another original feature of the paper [109] is the use of spectral methods for sparsifying the Schur complement. As was mentioned earlier, the Schur complement will typically be dense, if not full in most situations. The authors invoke 'sparsification' to reduce the number of edges. We will cover sparsification in Section 2.5.3.

Example. As an example, we return to the illustration of Figure 2.5. Using normalized Laplacians, we find that the largest eigenvector separates the graph in two parts according to its polarity, namely $V_1 = \{1, 5, 6, 9, 10\}$ and $V_2 = \{2, 3, 4, 7, 8, 11\}$. Thus, it is able to discover V_1, the rather natural independent set we selected earlier.

One important question that can be asked is *why resort to the Schur complement as a means of graph reduction?* A number of properties regarding the Kron reduction were established in [46] to provide justifications. Prominent among these is the fact that the resistance distance [49] between nodes of the coarse graph are preserved. The resistance distance involves the pseudo-inverse of the Laplacian.

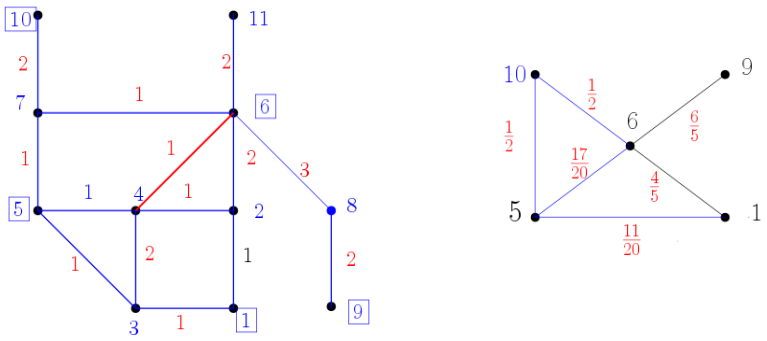


Fig. 2.5: A small graph (left) and its Kron reduction (right). The reduced set of nodes is $V_1 = \{1, 5, 6, 9, 10\}$. These node labels are kept in the Kron reduction. Numbers on the middle of the edges are the weights.

2.5.2. Relationship between Kron reduction and independent set coarsening. Instead of invoking independent sets, the Kron reduction, as it is used in [109], 'downsamples' nodes by means of spectral information. While these samples may form an independent set, this is not guaranteed. Just like independent set coarsening, the coarse graph is built from the Schur complement associated with the complement of the independent set (so-called *node cover*). These two ways of coarsening are illustrated in Figure 2.6.

Unlike independent set coarsening, the matrix L_{22} involved the Schur complement of (2.10) is not approximated by a diagonal before inversion, in effort to reduce the fill-ins introduced. Instead, spectral 'sparsification' is invoked. In the survey paper [46], the set V_1 represents a set of 'boundary points' in an electrical network. Improved sparsity is achieved by other means than those employed in Section 2.3, specifically by eliminating a selection of internal nodes instead of all of them, in the Gaussian elimination process.

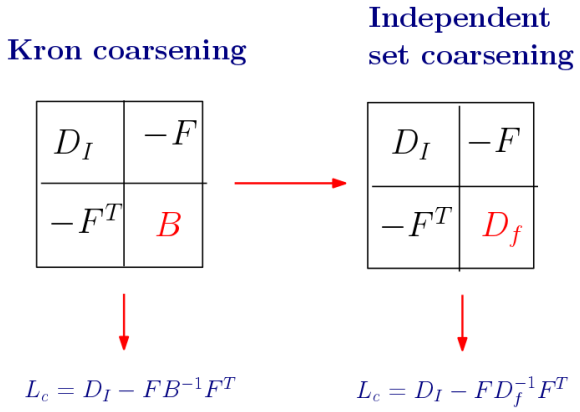


Fig. 2.6: Two ways of using independent sets for coarsening.

2.5.3. Graph sparsification. Graph sparsification methods aim at finding a sparse approximation of the original graph by trimming out edges from the graph [113, 112]. Here, the number of nodes remains the same but the number of edges $|E|$ can be much lower than in the original graph. This is motivated by the argument that in some applications, the graphs that model the data tend to be rather dense and that many edges can be removed without negatively impacting the result of methods that exploit these graphs; see, e.g., [65].

Different measures of closeness between the sparsified and the original graph have been proposed for this purpose, resulting in a wide range of strategies such as spanners [2], cut sparsifiers [66], and spectral sparsifiers [113]. Graph sparsification methods can be beneficial when dealing with high density graphs and come with rigorous theoretical guarantees [13]; see also [16].

We already mentioned at the end of Section 2.5.1 one specific use of graph sparsifiers in the context of multilevel graph coarsening. To sparsify the successive Schur complement obtained by the multilevel scheme, the authors of [109] resorted to a spectral sparsifier developed in [112]. The algorithm exploits a distance measure based on effective resistances [49]. This notion was briefly mentioned in the context of the Kron reduction in Section 2.5.1 and will be discussed in some detail in Section 5.4. The sparsification algorithm of Spielman and Srivastava [112] samples edges according to a probability defined from the original weights of the graph and these effective resistances. It is shown [112] that the graph Laplacian spectrum and resistance distances between nodes are approximately preserved with high probability, if the number of samples is high enough. If $\widetilde{G}$ is the sparsified version of G, and if $\widetilde{L}$ and L are their respective Laplacians, the main goal of spectral sparsifiers is to preserve the quadratic form associated with the Laplacians. The graph $\widetilde{G}$ is said to be a σ-spectral approximation of G if for all $x \in \mathbb{R}^n$,

$$\frac{1}{\sigma} x^T \widetilde{L} x \leq x^T L x \leq \sigma \; x^T \widetilde{L} x. \tag{2.11}$$

A trivial observation for σ-similar ($\sigma > 1$) graphs is that their Rayleigh quotients

$$\mu(x) = \frac{x^T L x}{x^T x}, \qquad \widetilde{\mu}(x) = \frac{x^T \widetilde{L} x}{x^T x}$$

for the same nonzero vector x satisfy the double inequality:

$$\frac{1}{\sigma} \; \widetilde{\mu}(x) \leq \mu(x) \leq \sigma \; \widetilde{\mu}(x). \tag{2.12}$$

Thus, these Rayleigh quotients are, in relative terms, within a factor of $\sigma - 1$ of each other:

$$\left| \frac{\widetilde{\mu}(x) - \mu(x)}{\widetilde{\mu}(x)} \right| \leq \sigma - 1.$$

This has an impact on eigenvalues. If σ is close to one, then clearly the eigenvalues of L and $\widetilde{L}$ will be close to each other, thanks to the Courant–Fisher min-max characterization of eigenvalues [55]. In what follows, S_k represents a generic k-dimensional subspace of $\mathbb{R}^n$ and eigenvalues are sorted decreasingly. In this situation, the theorem states that the k-th eigenvalue of the Laplacian L satisfies:

$$\lambda_k = \max_{\dim(S_k)=k} \; \min_{0 \neq x \in S_k} \mu(x). \tag{2.13}$$

The above maximum is achieved by a set, denote by S_* (which is just the linear span of the set of eigenvectors $u_1, \cdots, u_k$). Then

$$\lambda_k = \min_{0 \neq x \in S_*} \mu(x) \;\leq\; \min_{0 \neq x \in S_*} \sigma\widetilde{\mu}(x) \;\leq\; \sigma \max_{S_k} \min_{0 \neq x \in S_k} \widetilde{\mu}(x) \;\leq\; \sigma\, \widetilde{\lambda}_k.$$

The exact same relation as (2.11) holds if L and $\widetilde{L}$ are interchanged. Therefore, the above relation also holds if λ and $\widetilde{\lambda}$ are interchanged, which leads to the following double inequality, valid for $k = 1, 2, \cdots, n$:

$$\frac{\widetilde{\lambda}_k}{\sigma} \leq \lambda_k \leq \sigma \widetilde{\lambda}_k. \tag{2.14}$$

It is also interesting to note a link with preconditioning techniques. When solving symmetric positive definite linear systems of equations, it is common to approximate the original matrix A by a preconditioner which we denote here by $\widetilde{A}$. Two desirable properties that must be satisfied by a preconditioner $\widetilde{A}$ are that (i) it is inexpensive to apply $\widetilde{A}^{-1}$ to a vector; and (ii) the condition number of $\widetilde{A}^{-1}A$ is (much) smaller than that of A. The second condition translates into the condition that $(x^T Ax)/(x^T \widetilde{A}x)$ be small. If we assume that

$$\frac{1}{\sigma} \leq \frac{x^T Ax}{x^T \widetilde{A}x} \leq \sigma, \tag{2.15}$$

then the condition number of the preconditioned matrix, which is the ratio of the largest to the smallest eigenvalues of $\widetilde{A}^{-1}A$, will be bounded by σ^2. For additional details see [113] where this specific viewpoint was explicitly adopted, as well as [13, 112] among others.

2.5.4. Graph partitioning. The main goal here is to put in contrast the problem of coarsening with that of graph partitioning. To this end, a brief background is needed. In spectral graph partitioning [52, 12, 92], the important equality (1.3) satisfied by any Laplacian L is exploited. If x is a vector of entries $+1$ or -1, encoding membership of node i to one of two subgraphs, then the value of $x^T Lx$ is equal to 4 times the number of edge-cuts between the two graphs with this 2-way partitioning. We could try to find an optimal 2-way partitioning by minimizing the number of edge cuts, i.e., by minimizing $x^T Lx$ subject to the condition that the two subgraphs are of equal size, i.e., subject to $\mathbf{1}^T x = 0$. Since this optimization problem is hard to solve, it is common to 'relax' it by replacing the conditions $x \in \{-1, 1\}^n, x^T\mathbf{1} = 0$ with $x \in \mathbb{R}^n, \|x\| = 1, x^T\mathbf{1} = 0$. This leads to the definition of the Fiedler vector, which is the second smallest eigenvector of the Laplacian. Recall that the smallest eigenvalue of the graph Laplacian is zero and that when the graph is connected, this eigenvalue is simple and the vector $\mathbf{1}$ is a corresponding eigenvector.

It is interesting to note the similarity between spectral graph partitioning and Kron reduction. In both cases, the polarity of an eigenvector is used to partition the graph in two subgraphs. In the case of Kron reduction, it is the vector associated with the largest eigenvalue that defines the partitions; and one of these partitions is selected as the 'coarse', or the 'downsampled' set, according to the terminology in [109]. For graph partitioning, what is done instead is to use the eigenvector at the other end, the one next to the smallest, since the smallest is a constant vector. If we reformulate the problem back in terms of assignment labels of ± 1, then this would lead to the interpretation that in one case, we try to minimize edge cuts (partitioning) and in the other, we try to maximize them (Kron reduction).

An illustration is shown in Figure 2.7 with a small finite element mesh. As can be seen, using the second smallest eigenvector tends to color the graph in such a way that nearest neighbors of a node are mostly of the same color, while using the largest eigenvector tends to color the graph in such a way that nearest neighbors of a node are mostly of a different color. Another way to look at this is that using the second smallest eigenvector gives domains that tend to be 'fat' whereas the largest eigenvector gives domains that tend to be 'thin', like unions of lines separating each other. Another fact shown by this simple example is that neither of the two sets obtained is close to being an independent set.

The following property is straightforward to prove.

PROPOSITION 2.2. *Assume that the graph has no isolated node and that the components $\xi_1, \xi_2, \cdots, \xi_n$ of the largest eigenvector u_1 are nonzero. Let V_+ and V_- be the two subgraphs obtained from the polarities of the largest eigenvector. Then each node of V_+ (resp. V_-) must have at least one adjacent node from V_- (resp. V_+).*

Proof. The i-th row of the relation $Lu_1 = \lambda_1 u_1$ yields: (recall definition (1.2))

$$d_i \xi_i - \sum_{j \in N(i)} a_{ij} \xi_j = \lambda_1 \xi_i \quad \rightarrow \quad (\lambda_1 - d_i)\xi_i = - \sum_{j \in N(i)} a_{ij} \xi_j.$$

Note that $(\lambda_1 - d_i) > 0$ (due to assumption). Then, if $\xi_i \neq 0$ (left side) then at least one of the ξ_j's, $j \neq i$ (right side) must be of the opposite sign. ∎

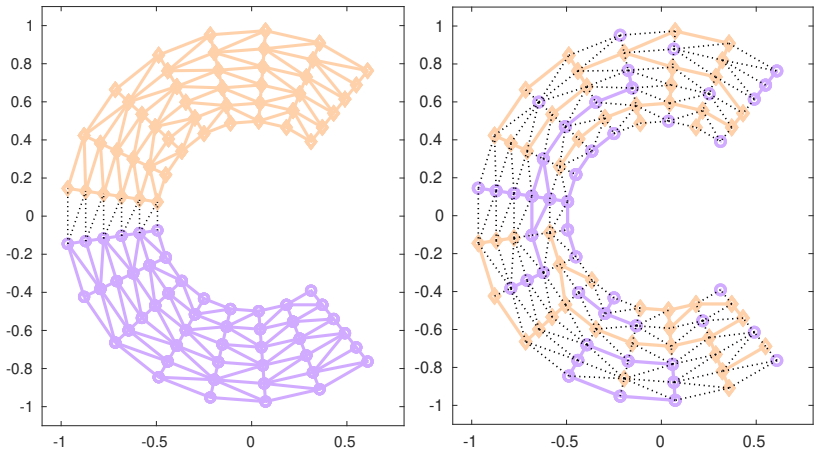


Fig. 2.7: Subgraphs from eigenvectors of the (normalized) Laplacian. Left side: using second smallest eigenvector; Right side: using largest eigenvector. The two sets are colored differently and edge-cuts are shown as dotted lines.

3. Graph coarsening in machine learning. In this section, we discuss existing methods related to graph coarsening in machine learning and discuss how they are employed in typical applications. We begin by defining the types of problems encountered in machine learning and the related terminology. The terms 'graphs' and 'networks' are often used interchangeably in the literature – although the term networks is often employed specifically for certain types of applications, whereas graphs are more general. For example, 'social networks' are common for applications that

model friendship or co-authorship, while one speaks of graphs when modeling chemical compounds. What muddles the terminology is the term 'graph neural networks,' which are neural networks (a type of machine learning model) for graphs. Furthermore, there are 'protein-protein interaction networks,' which are graphs of proteins; that is, each graph node is a protein, which by itself is a molecular graph.

Some commonly encountered graph tasks include: node classification (determine the label of a node in a given graph), link prediction (predict whether or not an edge exists between a pair of nodes), graph classification (determine the label of the graph itself), and graph clustering (group nodes that are most alike together). The task of node classification is often called interchangeably as *semi-supervised learning on graphs*, which mainly addresses the setting where only a small number of data points are equipped with ground-truth labels for learning the classification model, but the graph structure that connects the data points offers useful priors that may help the model predict well. One of the key concepts in a majority of the methods for solving these tasks is *embedding*, which is vector in $\mathbb{R}^d$ used to represent a node $v \in V$ or indeed the entire graph $G = (V, E)$.

3.1. Graph clustering and GraClus. As was seen earlier, graph coarsening is a basic ingredient of multilevel graph partitioning, where each level is a coarse version of the graph in the past level. The same graph partitioning tools can be used in data-related applications (e.g., graph clustering), but the requirement of having partitions of equal size is no longer relevant. This observation led to the development of approaches specifically for data applications; see, e.g., [43]. The method developed in [43] uses a simple greedy graph coarsening approach whereby nodes are visited in a random order. When visiting a node, the algorithm merges it with the unvisited nearest neighbor that maximizes a certain measure based on edge and node weights. The visited node and the selected neighbor are then marked as visited. The algorithm developed in [43], which is known as GraClus, uses different tools from those of graph partitioning. Because it is used for data applications, the refinement phase exploits a kernel K-means technique instead of the usual Kernighan–Lin procedure [68].

3.2. Multilevel graph coarsening for node embedding. In one form of node embedding, one seeks a mapping Φ from the node set V of a graph $G = (V, E)$ to the space $\mathbb{R}^{n\times d}$ where $n = |V|$ and

$$\Phi : v \in V \longrightarrow \Phi(v) \ \in \ \mathbb{R}^d. \tag{3.1}$$

In other words, each node is mapped to a vector in d-dimensional space. Here, the dimension d is usually much smaller than n. Many embedding methods have been developed and used effectively to solve a variety of problems; see, e.g., [100, 14, 1, 115, 33, 57, 123, 35] and [56] for a survey. The idea of applying coarsening to obtain embedding for large graphs has been gaining ground in recent years; see, [35, 74, 42, 50, 94].

The authors of [35] present a method dubbed *hierarchical representation learning for networks* (HARP), which exploits coarsening to facilitate and improve graph embedding. The method starts by performing a sequence of ℓ graph coarsening steps to produce graphs $G_1, G_2, \cdots, G_\ell$ from the initial graph G_0. Then, an embedding is performed on the final level to produce the mapping Φ_ℓ. This embedding is then propagated back to the original level by proceeding as follows. Starting from level $i = \ell$, the mapping Φ_i is naturally prolongated ('extrapolated') from level i to level $i - 1$ to yield a mapping Φ'_{i-1}. An extra step is taken to refine this embedding and

obtain Φ_{i-1}. This specific step is rather reminiscent of the 'post-smoothing' steps invoked in various MG schemes for solving linear systems. Post-smoothing consists of a few steps of smoothing operations, typically a standard relaxation method (e.g., Gauss-Seidel), applied after an approximate solution is prolonged from a coarse level. The MILE method described in [74] is rather similar to the HARP approach described above, the main difference being that the refinement proposed in this method exploits neural networks.

HARP and MILE are general frameworks that use coarsening to improve graph embedding. In the remainder of this section, we give an illustrative example to demonstrate the effectiveness of HARP. We examine the performance improvement of three widely used graph embedding algorithms: DeepWalk [91], LINE [115], and Node2vec [57], each combined with HARP. Furthermore, because HARP is a general framework and we have the freedom to choose the coarsening method it uses, we examine the impact of different coarsening methods on the performance improvement. Three coarsening methods are tested: HEM (Section 2.2), algebraic distance (Section 2.4), and the LESC method to be introduced in Section 5 (it is similar to HEM but uses spectral information to define the visiting order of nodes).

We evaluate the HARP framework on a node classification task with the Citeseer graph [107]. Given a graph G where some nodes are labeled, the task of node classification amounts to predicting the labels of the remaining nodes. Citeseer is a citation graph of computer science publications, consisting of 3.3K nodes and 4.5K edges. The label of each node indicates the subject area of the paper. We first generate the node embedding for each node using the HARP method. Then, a fraction of the nodes are randomly sampled to form the training set and the remaining is used for testing. We train a logistic regression model [20] by using the training data and evaluate the classification performance on the test data. We use the macro-average F1 score [93] as the performance metric, which is the mean of the F1 score for each label category.[3] Figure 3.1 shows the score under different training set sizes.

As can be seen the HARP framework consistently improves all these embedding methods, especially LINE and DeepWalk. Each coarsening approach used inside the HARP framework improves the performance to a different degree, with the LESC approach generally outperforming the others.

3.3. Graph neural networks and graph pooling. GNNs have recently attracted a great deal of interest in various disciplines, including biology [44], chemistry [122], and social networks [125], where data is modeled by graphs. A major class of GNNs are those of a convolution style that generalize lattice convolutions in convolutional neural networks (CNNs). A standard convolution applies a filter on a signal; in CNNs, the signal is a 2-dimensional image and the filter has a very small support—say, a 3×3 window. Convolution-style GNNs generalize the regularity of such a filter to irregularly connected node pairs [29, 60, 83]. Specifically, the regular window is replaced by the 1-hop neighborhood of a node.

One such representative GNN is the *graph convolutional network* (GCN) [69]. A 2-layer GCN maps a feature matrix X to the label matrix Y (see notation introduced in Section 1.2) by including the graph adjacency matrix A in the mapping:

$$\widetilde{Y} = \mathrm{softmax}(\widehat{A} \cdot \mathrm{ReLU}(\widehat{A}XW^0) \cdot W^1), \tag{3.2}$$

[3] An alternative definition, which is less used, is that the macro-average F1 score is the harmonic mean of the macro-average precision and the macro-average recall, where the macro-average precision/recall is the mean of the precision/recall for each label category.

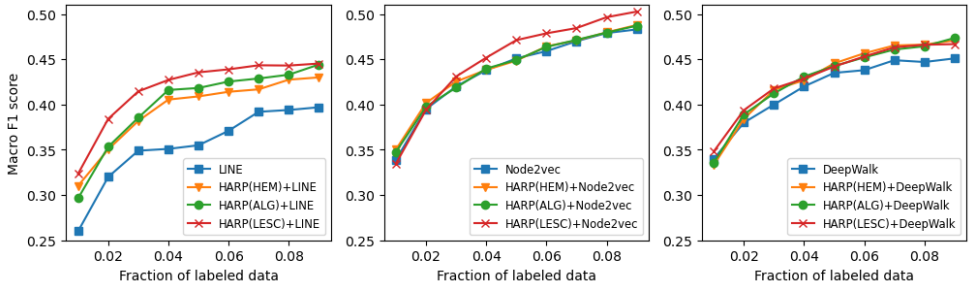


Fig. 3.1: Classification results on the Citeseer graph. The x-axis shows the portion of nodes used for training. The y-axis shows the macro-average F1 score. Each reported score is an average over five random repetitions.

where $\widehat{A}$ is a normalization of A through $\widehat{A} = \widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}$, $\widetilde{A} = A+I$, $\widetilde{D} = \mathrm{diag}(\widetilde{d}_i)$, and $\widetilde{d}_i = \sum_j \widetilde{a}_{ij}$. Therefore, matrix products such as $\widehat{A}X$ denote the convolution by using a 1-hop neighborhood filter.

The following concepts are for neural networks. The functions ReLU and softmax are nonlinear *activation functions*: $\mathrm{ReLU}(x) = \max\{x, 0\}$ is an elementwise function, while $\mathrm{softmax}(x) = [e^{-x_1}/c, \ldots, e^{-x_d}/c]$ with $c = \sum_j e^{-x_j}$ is a vector function; it acts on each row if the input is a matrix. The matrices W^0 and W^1 are called *weight matrices*. Their contents are not manually specified but *learned* through minimizing the discrepancy between $\widetilde{Y}$ and the ground truth label matrix Y.

Besides GCN, the literature has seen a large number of generalizations of lattice convolution to convolutions in the graph context, including for example, spectral [30, 61, 69, 41] and spatial [88, 59, 121, 127, 124, 84] schemes.

GNNs such as (3.2) essentially produce a mapping $\Phi : v \in V \longrightarrow \Phi(v) \in \mathbb{R}^c$ for every node v in the graph, if we read only one row of $\widetilde{Y}$ in (3.2). This mapping is almost identical to the form (3.1) discussed in the context of node embedding; the only nominal difference is that the output is in a c-dimensional space while the node embedding is in d-dimensional space. This difference is caused by the need for the output $\widetilde{Y}$ to be matched with the ground truth Y that has c columns (c label categories); while for node embedding, the embedding dimension d may be arbitrary. However, if our purpose is not to match $\widetilde{Y}$ with Y, we can adjust the number of columns of W^1 so that the right-hand side of (3.2) can be repurposed for producing a node embedding. More significantly, we may take the elementwise minimum, the average, or the weighted average of the node embedding vectors to form an embedding vector for the entire graph $G = (V, E)$ [48]. In other words, a GNN, with a slight modification, can produce a mapping

$$\Psi : G \longrightarrow \Psi(G) \in \mathbb{R}^d. \tag{3.3}$$

Now that we see how GNNs can be utilized to produce a graph embedding through the mapping Ψ, a straightforward application of coarsening is to use $\Psi(G_c)$ in place of $\Psi(G)$ for the classification of G. This simple idea can be made more sophisticated in two ways. First, suppose we perform a multilevel coarsening resulting in a sequence of increasingly coarser graphs $G = G_0, G_1, G_2, \ldots, G_\ell$. We may concatenate the vectors $\Psi(G_0), \Psi(G_1), \ldots, \Psi(G_\ell)$ and treat the resulting vector as the embedding of G. We

use the concatenation result (say ψ_G) to classify G, though building a linear classifier that takes ψ_G as input and outputs the class label.

Second, we introduce the concept of *pooling*. Pooling in neural networks amounts to taking a min/max or (weighted) average of a group of elements and reducing it to a single element. In the context of graphs, pooling is particularly relevant to coarsening, since if we recall the Galerkin projection $f^{(\ell+1)} = P_\ell^T f^{(\ell)}$ in AMG (see (2.3)), the interpolation matrix P_ℓ plays the role of pooling: each column of P_ℓ defines the weights in the averaging of nodes in the last graph into a node in the coarse graph. Hence, in the context of GNNs, we call P_ℓ a *pooling matrix*. This matrix can be obtained directly through the definition of a coarsening method [41, 111], or it can be unspecified but learned through training [126, 53, 73]. By using successive poolings that form a hierarchy, recent work [126, 73, 53, 70] has shown that hierarchical pooling improves graph classification performance.

4. Spectral coarsening. While the terms 'spectral graph partitioning' and 'spectral clustering' are quite well-known, the term 'spectral graph coarsening' is less explored in the literature. There are two aspects, and therefore possible directions, to spectral coarsening. First, it may be desirable in various tasks to preserve the spectral properties of the original graph, as is the case in the local variation method proposed by [76]. The second aspect is that one may wish to apply spectral information for coarsening. The method proposed by [109] falls in this category. It uses the eigenvector of the graph Laplacian to select a set of nodes for Kron reduction [46] discussed earlier.

In this section, we present an approach for the first aspect; whereas in Section 5, we develop an approach related to the second aspect. Regarding the first aspect, eigenvectors of the graph Laplacian encapsulate much information on the structure of the graph. For example, the first few eigenvectors are often used for partitioning the graph into more or less equal partitions. Therefore, the first question we will ask is whether or not it is possible to coarsen a graph in such a way that eigenvectors are 'preserved.' Of course, the coarse graph and the original one have different sizes so we will have to clarify what is meant by this.

4.1. Coarsening and lifting. Recall from AMG that coarsening is represented by the matrix $P^T \in \mathbb{R}^{n_c \times n}$. It helps to view a coarse node as a linear combination of a set of nodes in the original graph. Let the k-th coarse node be denoted by $v_k^{(c)}$ and the set be S_k. The weights are p_{ik} for each $v_i \in S_k$:

$$v_k^{(c)} = \sum_{v_i \in S_k} p_{ik} v_i \quad \Leftrightarrow \quad v^{(c)} = P^T v. \tag{4.1}$$

The coarse adjacency matrix is then defined as:

$$A_c = P^T A P. \tag{4.2}$$

A similar framework of writing a coarsened matrix in the form of (4.2) is adopted in [76]. Note that in general, A_c is not binary. Here, we assign the diagonal entries of A_c to 0 and all non-zero entries to 1.

The original graph G and its coarse graph G_c have a different number of nodes. If we wish to compare the properties of these two graphs, it is necessary to 'lift' the graph Laplacian of G_c into a matrix that has the same size as that of G. Let L and L_c denote the Laplacian of the original graph and the coarse graph, respectively.

One way to construct a matrix L_c that is guaranteed to be a graph Laplacian is as follows [76, 64]:

$$L_c = Q^T L Q, \tag{4.3}$$

where $Q \in \mathbb{R}^{n \times n_c}$ is a sparse matrix with

$$Q_{ij} = \begin{cases} 1, & \text{node } i \text{ in } S_j, \\ 0, & \text{otherwise.} \end{cases} \tag{4.4}$$

In the simplest case, the entries P in (4.1) can be defined as $p_{ik} = 1/|S_k|$ for all $v_i \in S_k$. Then, the resulting P is the pseudoinverse of Q^T, with $Q^T P = I_{n_c}$. Therefore, the lifted counterpart of L_c, denoted by L_l, is defined as

$$L_l = P L_c P^T, \tag{4.5}$$

because PQ^T is a projector.

4.2. The projection method viewpoint. If we extend the matrix P as an orthonormal matrix, then PP^T is a projector and the lifted Laplacian defined in (4.5) becomes $L_l = PP^T LPP^T$. It is useful to view spectral coarsening from the *projection method* [104] angle.

Consider an orthogonal projector π and a general (symmetric) matrix A. In an orthogonal projection method on a subspace $\mathcal{V}$, we seek an approximate eigenpair $\tilde{\lambda}, \tilde{u}$, where $\tilde{u} \in \mathcal{V}$ such that

$$\pi(A - \tilde{\lambda} I)\tilde{u} = 0.$$

If $V = [v_1, v_2, \cdots, v_k]$ is an orthonormal basis of $\mathcal{V}$ and the approximate eigenvector is written as $\tilde{u} = Vy$, then the previous equation immediately leads to the problem

$$V^T A V y = \lambda y. \tag{4.6}$$

The eigenvalue $\tilde{\lambda}$ is known as a Ritz value and $\tilde{u}$ is the associated Ritz vector.

Recall the orthogonal projector π onto the columns of P; that is, $\pi = PP^T$. If we look at the specific case under consideration, we notice that this is precisely what is being done and that the basis vectors V are just the columns of P.

When analyzing errors for projection methods, the orthogonal projector π represented by the matrix PP^T plays a particularly important role. Specifically, a number of results are known that can be expressed based on the distance $\|(I - \pi)u\|$ where u is an eigenvector of A; see, e.g., [104]. The norm $\|(I - \pi)u\|$ represents the distance of u to the range of π in $\mathbb{R}^n$.

4.3. Eigenvector preserving coarsening. Based on the interpretation of projection methods, it is desirable to construct a projector that preserves eigenvectors. We say that a given eigenvector u is exactly preserved or just 'preserved' by the coarsening if $(I - \pi)u = 0$. If this is the case then when we solve the projected problem (4.6), we will find that $y = P^T u$ is an eigenvector of $V^T AV$ associated with the eigenvalue λ:

$$V^T AV(V^T u) = V^T A \pi u = V^T A u = \lambda V^T u.$$

The Ritz vector is $\tilde{u} = Py = PP^T u = \pi u = u$ which is clearly an eigenvector.

What might be more interesting is the more practical situation in which $(I - \pi)u$ is not zero but just small. In this case, there are established bounds [104] for the

angle between the exact and approximate eigenvectors based on the quantity $\epsilon = \|(I - \pi)u\|_2$.

In what follows, we instantiate the general matrix A by the graph Laplacian matrix L and consider the preservation of its eigenvectors. From many machine learning methods (e.g., the Laplacian eigenmap [15]), the bottom eigenvectors of L carry the crucial information of a dataset. Thus, they are the ones that we want to preserve.

4.4. Preserving one eigenvector. If we want to coarsen the graph into k nodes, we partition an eigenvector u into k parts. Up to permutations of the elements of u, let us write, in matlab notation

$$u = \begin{bmatrix} u_1 \ ; \ u_2 \ ; \ \cdots \ ; \ u_k \end{bmatrix}.$$

Then, we let

$$P = \begin{bmatrix} u_1/\|u_1\| & & & \\ & u_2/\|u_2\| & & \\ & & \ddots & \\ & & & u_k/\|u_k\| \end{bmatrix}.$$

Clearly, P is orthonormal and satisfies $u = PP^T u$. In other words, the matrix P so defined preserves the eigenvector u of the graph Laplacian in coarsening.

The square of an element of u is called the *leverage score* of the corresponding node (see Section 5.1). Then, each $\|u_i\|^2$ is the leverage score of the i-th coarse node. In other words, if a collection of nodes of the original graph is grouped into a coarse node, then the sum of their leverage scores is the leverage score of the coarse node.

4.5. Preserving m eigenvectors. The one-eigenvector case can be easily extended to m eigenvectors. Let U be the matrix of these eigenvectors; that is, U has m columns, each of which is an eigenvector. We partition U similarly to the preceding subsection, as

$$U = \begin{bmatrix} U_1 \ ; \ U_2 \ ; \ \cdots \ ; \ U_k \end{bmatrix}.$$

Then, we define the matrix P in the following way:

$$P = \begin{bmatrix} P_1 & & & \\ & P_2 & & \\ & & \ddots & \\ & & & P_k \end{bmatrix} \equiv \begin{bmatrix} U_1 R_1^{-1} & & & \\ & U_2 R_2^{-1} & & \\ & & \ddots & \\ & & & U_k R_k^{-1} \end{bmatrix},$$

where for each partition i, $U_i = P_i R_i$. The equality $U_i = P_i R_i$ can be any factorization that results in orthonormal matrices P_i (so that P is orthonormal). A straightforward choice is the QR factorization. Alternatively, one may use the polar factorization, where P_i and R_i are the unitary polar factor and the symmetric positive definite polar factor, respectively. This factorization is conceptually closer to the one-eigenvector case.

In contrast with the one-eigenvector case, now a collection of nodes of the original graph is grouped into m coarse nodes, which are all pairwise connected in the coarse graph. The total number of nodes in the coarse graph is mk. Because the Frobenius norm of U_i is equal to that of R_i, we see that the sum of the leverage scores of the original nodes in a partition is the same as that of the m resulting coarse nodes.

It is not hard to see that each eigenvector in U is preserved. Indeed, if u is the j-th column of U, then $u = Ue_j$ and, using matlab notation,

$$\begin{aligned} u &= [U_1 e_j \ ; \ U_2 e_j \ ; \ \cdots \ ; \ U_k e_j] \\ &= [P_1 R_1 e_j \ ; \ P_2 R_2 e_j \ ; \ \cdots \ ; \ P_k R_k e_j] \\ &\equiv [P_1 \rho_1 \ ; \ P_2 \rho_2 \ ; \ \cdots \ ; \ P_k \rho_k] \\ &= P[\rho_1 \ ; \ \rho_2 \ ; \ \cdots \ ; \ \rho_k], \end{aligned}$$

where we have set $\rho_i = R_i e_j$ for $i = 1, \cdots, k$. Therefore, u is in the range of P and as such it will be left invariant by the projector π: If $\rho = [\rho_1 \ ; \ \rho_2 \ ; \ \cdots \ ; \ \rho_k]$ then $\pi u = PP^T(P\rho) = P\rho = u$.

Clearly, it is not necessary to use a regular fixed and equal-sized splitting for the rows of U (and u); i.e., we can select any grouping of the rows that can be convenient for, say, preserving locality, or reflecting some clustering.

5. Coarsening based on leverage scores. Spectral coarsening may have desirable qualities when considering spectral properties, but these methods face a number of practical difficulties. Among them is the fact that the coarsened graph tends to be dense. For this reason, spectral methods will be invoked mostly as a tool to provide an ordering of the importance of the nodes—which will in turn be used for defining a traversal order in other coarsening approaches (e.g., HEM). This has been a common theme in the literature [35, 64, 76].

5.1. Leverage scores. Let A be a general matrix and let U be an orthonormal matrix, whose range is the same as that of A. The leverage score [47] of the i-th row of A is defined as the squared norm of the i-th row of U:

$$\eta_i = \|U_{i,:}\|_2^2. \tag{5.1}$$

Clearly, the leverage score is invariant to the choice of the orthonormal basis of the range of A.

Leverage scores defined in the form (5.1) have been used primarily for (rectangular) matrices A that represent data. In these cases, the columns of U are the dominant singular vectors of A and the matrix $\pi = UU^T$ is an orthogonal projector that projects a vector in $\mathbb{R}^n$ onto the span of U. The vector of leverage scores, η, is the diagonal of this projector. This quantity appears also in a different context in quantum physics, where the index i represents a location in space, η_i represents the electron density in position i, and the projector π is the density matrix in the idealistic case of zero temperature. See Section 3.4 of [81].

5.2. Leverage-score coarsening (LESC). For coarsening methods such as Algorithm 2.1, the traversal order in the coarsening process can have a major impact on the quality of the results. Instead of the heavy edge matching strategy, we now consider using leverage scores to measure the importance of a node. Exploiting what we know from spectral graph theory, we will use the bottom eigenvectors of the graph Laplacian L to form U. However, we find that in many cases, the traversal order is sensitive to the number of eigenvectors, r. To lessen the impact of r on the outcome, we weigh individual entries U_{ik} in (5.1) by using the eigenvalues $\lambda_1, \ldots, \lambda_r$ of L. Specifically, we define

$$\eta_i = \sum_{k=1}^{r} (e^{-\tau \lambda_k} U_{ik})^2, \tag{5.2}$$

where τ represents a decay factor of the weights. This leads to a modification of HEM that is based on leverage scores (5.2) which we call *leverage score coarsening*, or LESC for short.

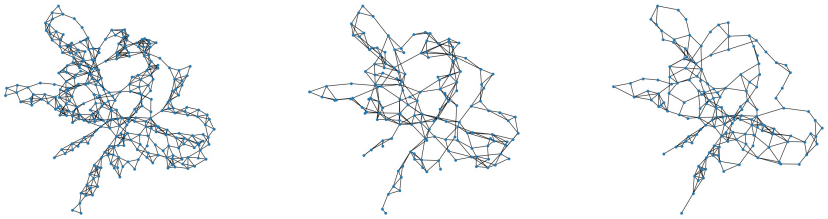
Algorithm 5.1 describes the LESC procedure. Its main differences from HEM (Algorithm 2.1) lie in (i) the traversal order and (ii) the handling of singletons. While HEM proceeds by the heaviest edge, LESC scans nodes in decreasing η values. At the beginning of each for-loop, LESC selects the next unvisited node i with the highest leverage score and selects from its unvisited neighbors a node j, where the edge (i, j) has the heaviest edge weight, to create a coarse node.

The way in which LESC handles the singletons is elaborated in lines 14–25 of Algorithm 5.1. During the traversal, we append a singleton to a set named *Single*. Depending on the desired coarse graph size n_c, there are two ways to assign parents to the singletons: lines 20–22 handle a real singleton and lines 23–25 handle a leftover singleton. This extra step outside HEM better preserves the local structure surrounding high-degree nodes, as well as the global structure of the graph.

Algorithm 5.1 Leverage-Score Coarsening (LESC)

1: **Input:** Weighted graph $G = (V, E, A)$; leverage score η; coarse graph size n_c.
2: **Output:** Coarse nodes; $Prnt$ list
3: **Init:** $Prnt(i) = 0\ \forall i \in V$, $new = 0$, $Single = \emptyset$
4: **for** max-score **to** min-score node i **do**
5: **if** i has no neighbor **then**
6: $new = new + 1$; $Prnt(i) = new$
7: **else**
8: **for** max **to** min edge (i, j) **do**
9: **if** $Prnt(j) == 0$ **then**
10: $new = new + 1$
11: $Prnt(i) = Prnt(j) = new$
12: **end if**
13: **end for**
14: **if** $Prnt(i) == 0$ **then**
15: Add i to *Single* set
16: **end if**
17: **end if**
18: **end for**
19: Randomly shuffle *Single* set
20: **for** the first $n_c - new$ nodes v **in** *Single* **do**
21: $new = new + 1$; $Prnt(v) = new$
22: **end for**
23: **for** remaining nodes v **in** *Single* **do**
24: $Prnt(v) = Prnt(j)$ where $j = \text{argmax}_i(A_{i,v})$
25: **end for**

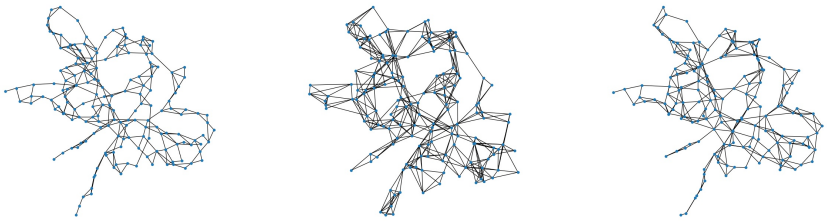
For an illustration, we visualize the coarse graphs produced by the following five coarsening methods: HEM, local variation (LV) [76], algebraic distance, Kron reduction, and LESC. The original graph is selected from the D&D protein data set; see the Section 5.5 for a detailed description. The coarse graphs are shown in Figure 5.1. We observe from the figure that the global structure of the graph is well preserved in each coarse graph. Moreover, the connection chains of the nodes are well preserved.



(a) Original
312 nodes, 761 edges

(b) HEM
156 nodes, 340 edges

(c) LV
156 nodes, 321 edges



(d) algebraic distance
171 nodes, 327 edges

(e) Kron
156 nodes, 485 edges

(f) LESC
156 nodes, 362 edges

Fig. 5.1: Visualization of a graph and the coarsened graphs obtained by different methods.

Note also that, as expected, the coarse graph from Kron reduction is denser than the other coarse graphs.

5.3. Interpretations. Leverage scores defined in (5.1) have been used primarily for matrices that represent data. The form used by us, (5.2), stabilizes the ordering of the scores when the number r of dominant eigenvectors varies. When τ or r is large, there is little difference between the value (5.2) and the following one that uses all eigenvectors:

$$\eta_i = \sum_{k=1}^{n} (e^{-\tau\lambda_k} U_{ik})^2. \tag{5.3}$$

The form (5.3) can lead to interesting interpretations and results.

First, observe that if we denote by e_i the i-th column of the identity matrix, then

$$\eta_i = \sum_{k=1}^{n} e^{-2\tau\lambda_k} |U_{ik}|^2 = e_i^T \exp(-2\tau L) e_i. \tag{5.4}$$

That is, η_i is nothing but the i-th diagonal entry of the matrix $H \equiv \exp(-2\tau L)$. If the adjacency matrix A is doubly stochastic, then $L = I - A$ and $H = \exp(-2\tau L) =$

$\exp(-2\tau I + 2\tau A) = e^{-2\tau} \cdot \exp(2\tau A)$. Therefore, since A has nonnegative entries, so does H. Then, H is a stochastic matrix (in fact, doubly stochastic because of symmetry). To see this, we have $L\mathbf{1} = 0$ and thus by the Taylor series of the matrix exponential, $H\mathbf{1} = \mathbf{1}$. Now that H is a stochastic matrix, the leverage score η_i (i-th diagonal entry of H) represents the self-probability of state i.

There exists another interpretation from the *transient solutions of Markov chains*; see Chapter 8 of [114]. In the normalized case, the negative Laplacian $-L$ plays the role of the matrix Q in the notation of continuous time Markov chains (see Section 1.4 of [114]). Given an initial probability distribution $\pi(0) \in \mathbb{R}^{1\times n}$, the transient solution of the chain at time t is $\pi(t) = \pi(0)\exp(tQ) = \pi(0)\exp(-tL)$. If $\pi(0) = e_i^T$, then $\pi(t)$ carries the probabilities for each state at time t. In particular, the i-th entry (which coincides with the leverage score η_i if $t = 2\tau$) is the probability of remaining in state i.

5.4. Alternative definition. The definition of η_i in (5.2) modifies the standard leverage score by using decaying weights, to reduce sensitivity of the number of eigenvectors used. In principle, any decreasing function of eigenvalues can be used to get distinguishable leverage scores of a Laplacian. We consider the following alternative, which is related to the pseudoinverse of the Laplacian:

$$\eta_i = \sum_{j=2}^{n} \left(\frac{1}{\sqrt{\lambda_j}} U_{ij} \right)^2 . \tag{5.5}$$

Several points are worth noting. First, weighted leverage scores emphasize eigenvectors corresponding to small non-zero eigenvalues. Hence, weighted leverage scores reveal the contribution of nodes to the global structure. Second, a smaller weighted leverage score indicates a higher topological importance of a node. Third, calculating the complete set of eigenvectors of L is expensive. Given a parameter r, we can further define r-truncated weighted leverage scores using only r eigenpairs:

$$\eta_i = \sum_{j=2}^{r} \left(\frac{1}{\sqrt{\lambda_j}} U_{ij} \right)^2 . \tag{5.6}$$

For simplicity, we refer to these numbers as leverage scores of L, and use $\eta = [\eta_1, \cdots, \eta_n]$ to denote them. A visual example of using η to define the traversal order in Algorithm 5.1 is given in Figure 5.2.

The definition (5.5) has a direct connection with the pseudoinverse of the Laplacian. In particular, the vector η is equal to the diagonal of $L^\dagger$. To see this, we first notice that L and $L^\dagger$ have the same set of eigenvectors, and nontrivial eigenvalues are reciprocals of each other. We then write $L^\dagger$ as $L^\dagger = U\Sigma^\dagger U^T$, where $\Sigma^\dagger$ is a diagonal matrix with $0 < 1/\lambda_n \leq \cdots \leq 1/\lambda_2$ on the diagonal. Diagonal entries of $\Sigma^\dagger$ are non-negative, so we can write

$$L^\dagger = U\sqrt{\Sigma^\dagger}\left(U\sqrt{\Sigma^\dagger}\right)^T , \tag{5.7}$$

from which we get

$$L^\dagger_{ii} = \sum_{j=2}^{n} \frac{U_{ij}}{\sqrt{\lambda_j}} \frac{U_{ij}}{\sqrt{\lambda_j}} = \eta_i .$$

The pseudoinverse of L has long been used to denote node importance. The article [117] provides a rather detailed description of the link between $L^\dagger$ and the various

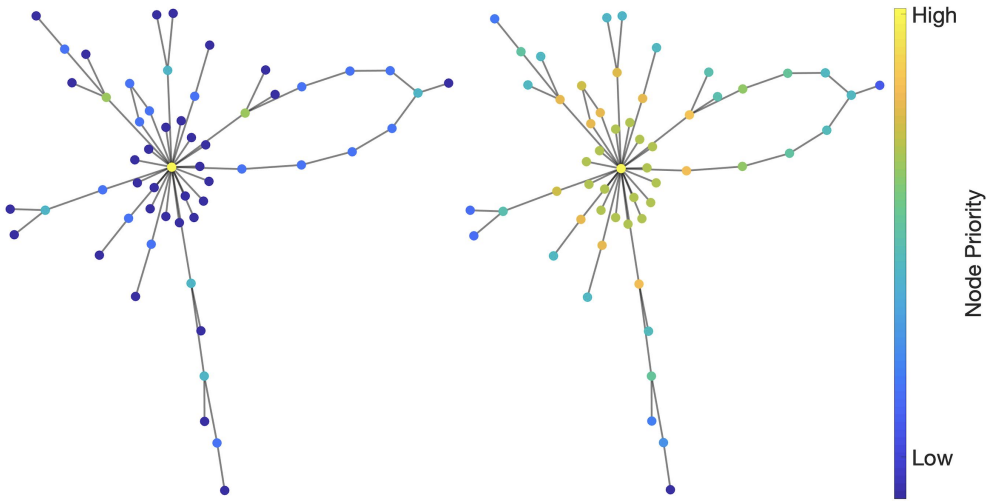


Fig. 5.2: Traversal order of HEM (left) and LESC (right) on an unweighted graph.

definitions of network properties. The effective resistance distance between nodes a and b is given by $\omega_{ab} = L^\dagger_{aa} + L^\dagger_{bb} - 2L^\dagger_{ab}$. The trace of $L^\dagger$ defines a graph metric called effective graph resistance [49], which is related to random walks [34] and the betweenness centrality [87]. In addition, the columns s_j of the matrix $U\sqrt{\Sigma^\dagger}$ in (5.7) have a particular significance. The squared distance between two columns s_a and s_b is equal to ω_{ab} and based on this notion of distance, [97] propose to use $(L^\dagger_{ii})^{-1}$ as a measure of the topological centrality of a node i. The smaller the distance, the higher its topological centrality. This metric has been used to quantify the roles of nodes in independent networks [108]. Therefore, by using η, LESC prioritizes nodes with high importance with respect to this metric.

The leverage score vector η is also related to the change of the Laplacian pseudoinverse when merging a pair of nodes in coarsening. We follow the work by [62] to elucidate this. To simplify the comparison between two matrices with different sizes, consider the following perspective: during coarsening, instead of merging a pair of nodes and reducing the graph size by one, we assign the corresponding edge with an edge weight $+\infty$. To avoid possible confusion with L_c, we use L_∞ to denote the coarse graph Laplacian (the Laplacian of the $+\infty$-weighted graph). Suppose we assign an edge $e(v_i, v_j)$ with the $+\infty$ weight, then the difference between L and L_∞ is

$$\Delta L = L_\infty - L = w b_e b_e^T, \quad (w = +\infty) \tag{5.8}$$

where b_e is defined in (1.5). Then, the change in $L^\dagger$ is given by the Woodbury matrix identity [54]:

$$\Delta L^\dagger = -\frac{w}{1 + w b_e^T L^\dagger b_e} L^\dagger b_e b_e^T L^\dagger = -\frac{1}{b_e^T L^\dagger b_e} L^\dagger b_e b_e^T L^\dagger. \quad (w = +\infty) \tag{5.9}$$

Thus, the magnitude of $\Delta L^\dagger$ can be defined by the Frobenius norm:

$$||\Delta L^\dagger||_F^2 = \frac{b_e^T L^\dagger L^\dagger b_e}{b_e^T L^\dagger b_e}. \tag{5.10}$$

The following result bounds the magnitude of $\Delta L^\dagger$ by using leverage scores.

PROPOSITION 5.1. *Let the graph be connected. The magnitude of the difference between $L^\dagger$ and $L^\dagger_\infty$ caused by assigning the $+\infty$ edge weight to an edge $e(i,j)$ is bounded by*

$$||\Delta L^\dagger||_F^2 \leq \kappa(L)(L^\dagger_{ii} + L^\dagger_{jj}),$$

where κ denotes the effective condition number (i.e., the largest singular value divided by the smallest nonzero singular value).

Proof. Let $L^\dagger = U\Lambda U^T$ be the spectral decomposition of $L^\dagger$ with eigenvalues sorted nonincreasingly: $\mu_1 \geq \cdots \geq \mu_{n-1} > 0 = \mu_n$. Further, let $x = \Lambda^{1/2}U^T b_e$. Then,

$$\frac{b_e^T L^\dagger L^\dagger b_e}{b_e^T L^\dagger b_e} = \frac{x^T \Lambda x}{x^T x} \leq \mu_1.$$

Now consider a lower bound and an upper bound of $b_e^T L^\dagger b_e$. Since b_e is orthogonal to the vector of all ones (an eigenvector associated with μ_n), we have

$$\left(\frac{b_e}{\sqrt{2}}\right)^T L^\dagger \left(\frac{b_e}{\sqrt{2}}\right) \geq \min_{\|y\|=1} \{y^T L^\dagger y \mid y^T \mathbf{1} = 0\} = \mu_{n-1}.$$

On the other hand, note that $b_e^T L^\dagger b_e = L^\dagger_{ii} + L^\dagger_{jj} - 2L^\dagger_{ij}$ and that $(L^\dagger_{ij})^2 \leq L^\dagger_{ii} L^\dagger_{jj}$ (by positive semidefiniteness). Then,

$$b_e^T L^\dagger b_e \leq \left(\sqrt{L^\dagger_{ii}} + \sqrt{L^\dagger_{jj}}\right)^2 \leq 2(L^\dagger_{ii} + L^\dagger_{jj}).$$

Invoking both the lower bound and the upper bound, we obtain

$$\frac{b_e^T L^\dagger L^\dagger b_e}{b_e^T L^\dagger b_e} \leq \mu_1 \leq \mu_1 \frac{2(L^\dagger_{ii} + L^\dagger_{jj})}{2\mu_{n-1}}.$$

Then, by noting that μ_1/μ_{n-1} is the effective condition number of $L^\dagger$ (as well as L), we conclude the proof. ∎

5.5. Experimental results. Here, we show an experiment to demonstrate the effective use of LESC to speed up the training of GNNs. We use the leverage scores defined in (5.6) to conduct the experiments.

We use three data sets for evaluation: D&D [44], REDDIT-BINARY (REBI), and REDDIT-MULTI-5K (RE5K) [125]. The first is a protein data set and the label categories are enzymes and non-enzymes. Each protein is represented by a graph, where nodes represent amino acids and they are connected if the two acids are less than six Angstroms apart. The last two are are social network data sets collected from the online discussion forum Reddit. Each discussion thread is treated as one graph, in which a node represents a user and there is an edge between two nodes if either of the two users respond to each other's comment. The label categories are community types and discussion topics, respectively. Statistics of the data sets are given in Table 5.1.

We focus on the task of *graph classification*. Given a collection of graphs, where some graphs are labeled, this task is to predict the labels of the remaining graphs. For example, protein graphs in D&D will be classified as enzymes or non-enzymes. As motivated in Section 3.3, we classify the coarsened graphs, which are smaller

Table 5.1: Dataset statistics.

	D&D	REBI	RE5K
#graphs	1178	2000	4999
#classes	2	2	5
Avg.#nodes	284.32	429.63	508.52
Avg.#edges	715.66	497.75	594.87

but retain the structural information of the original graphs. We use four GNNs (SortPool [127], DiffPool [126], TopKPool [53, 32], and SAGPool [73]) to perform the task and investigate the change of training time and prediction accuracy under three coarsening schemes (HEM, local variation (LV) [76], and LESC).

Figure 5.3 summarizes the time (bars) and accuracy (percentages) results. Each column is for one GNN and each row is for one data set. Inside a panel, three coarsening methods are compared, each using three coarsening levels.

When comparing times, note that applying coarsening to graph classification incurs two costs: the time to perform coarsening (as preprocessing) and the time for training. Therefore, we normalize the overall time by that of training a GNN without using coarsening. Hence, a relative time < 1 indicates improvement. In fact, the relative time is just the reciprocal of the speedup.

Here are a few observations regarding training times. First, for all three data sets, time reduction is always observed for HEM and LESC. Second, for these two methods, the coarsening time is almost negligible compared with the training time, whereas LV incurs a substantial overhead for coarsening in several cases. In LV, the coarsening time may even dominate the training time (see REBI and RE5K). Discounting coarsening, however, LV produces coarse graphs that lead to reduced training times. Third, in general, the deeper the levels, the more significant is the time reduction. The highest speedup for LESC, which is approximately 30x, occurs for REBI in three levels of coarsening.

To compare prediction accuracies, we compute the relative change (in %) in accuracy and annotate it on the right side of each panel in Figure 5.3. We see that LESC achieves the best performance among all three coarsening methods on all three datasets. It also improves accuracy on a few of the GNNs while for the others it produces an accuracy that is quite close to that achieved by not using coarsening. In other words, coarsening does not negatively impact graph classification overall.

6. Concluding remarks. This survey (with new results; see Section 4.4 onward) focused on graph coarsening techniques with a goal of showing how some common ideas have been utilized in two different disciplines while also highlighting methods that are specific to some applications. The recent literature on methods that employ graphs to model data clearly indicates that the general method is likely to gain importance. This is only natural because the graphs encountered today are becoming large and experiments show that if employed with care, coarsening does not cause a big degradation in the performance of the underlying method. As researchers in numerical linear algebra and scientific computing are increasingly turning their attention to problems related to machine learning, graph based methods, and graph coarsening in particular, are likely to play a more prominent role.

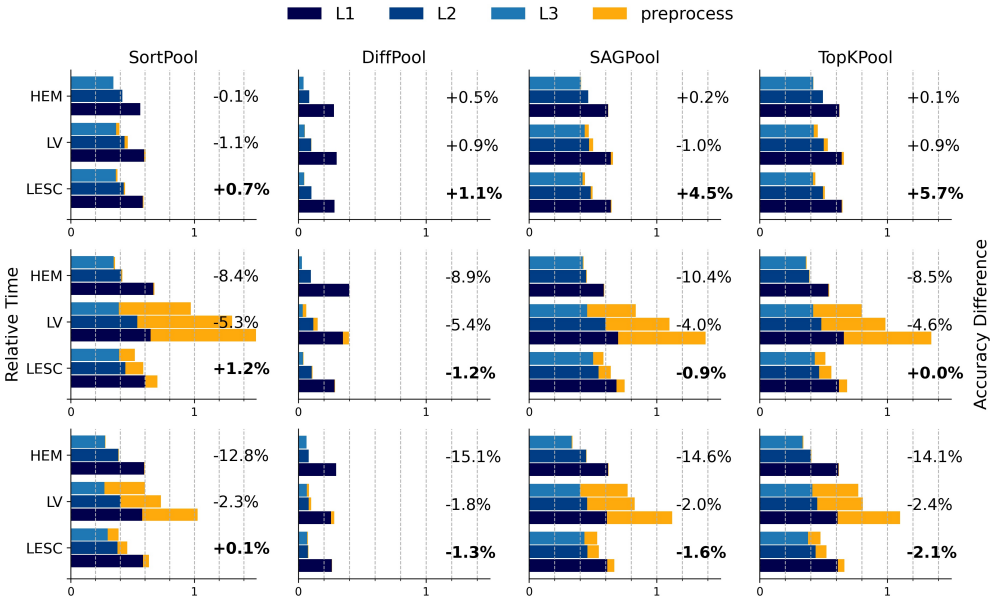


Fig. 5.3: Relative time and accuracy difference on D&D (top row), REBI (middle row), and RE5K (bottom row). Four classification methods (SortPool, DiffPool, SAGPool, and TopKPool) and three coarsening methods (HEM, LV, and LESC) are considered. For each classification method, we normalize the run time by that of the original method. Each run time consists of two parts: coarsening and GNN training with 10-fold cross validation. For each coarsening method, run times are reported for a number of coarsening levels. The highest accuracy achieved by each coarsening method is annotated on the right of the bar chart through differencing from that of the original graph (coarsening minus original). Positive values indicate accuracy increase. The best case for each chart is highlighted in boldface.

REFERENCES

[1] A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, *Distributed large-scale natural graph factorization*, in Proceedings of the 22Nd International Conference on World Wide Web, WWW '13, New York, NY, USA, 2013, ACM, pp. 37–48.

[2] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, *On sparse spanners of weighted graphs*, Discrete & Computational Geometry, 9 (1993), pp. 81–100.

[3] B. Aspvall and J. R. Gilbert, *Graph coloring using eigenvalue decomposition*, SIAM Journal on Algebraic Discrete Methods, 5 (1984), pp. 526–538.

[4] O. Axelsson and M. Larin, *An algebraic multilevel iteration method for finite element matrices*, J. Comput. Appl. Math., 89 (1997), pp. 135–153.

[5] O. Axelsson and M. Neytcheva, *Algebraic multilevel iteration method for Stieltjes matrices*, Numer. Linear Algebra Appl., 1 (1994), pp. 213–236.

[6] O. Axelsson and B. Polman, *A robust preconditioner based on algebraic substructuring and two-level grids*, in Robust multigrid methods. Proc., Kiel, Jan. 1988., W. Hackbusch, ed., Notes on Numerical Fluid Mechanics, Volume 23, Vieweg, Braunschweig, 1988, pp. 1–26.

[7] O. Axelsson and P. Vassilevski, *Algebraic multilevel preconditioning methods. I*, Numer. Math., 56 (1989), pp. 157–177.

[8] ———, *A survey of multilevel preconditioned iterative methods*, BIT, 29 (1989), pp. 769–793.

[9] ———, *Algebraic multilevel preconditioning methods. II*, SIAM J. Numer. Anal., 27 (1990), pp. 1569–1590.

[10] N. S. Bakhvalov, *On the convergence of a relaxation method with natural constraints on the elliptic operator*, U.S.S.R Computational Math. and Math. Phys., 6 (1966), pp. 101–135.
[11] R. E. Bank and C. Wagner, *Multilevel ILU decomposition*, Numerische Mathematik, 82 (1999), pp. 543–576.
[12] S. T. Barnard and H. D. Simon, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Concurrency: Practice and Experience, 6 (1994), pp. 101–107.
[13] J. Batson, D. A. Spielman, N. Srivastava, and S.-H. Teng, *Spectral sparsification of graphs: Theory and algorithms*, Commun. ACM, 56 (2013), p. 87–94.
[14] M. Belkin and P. Niyogi, *Laplacian eigenmaps and spectral techniques for embedding and clustering*, in Advances in Neural Information Processing Systems 14, Cambridge, MA, 2002. MIT Press., G. Dietterich, S. Becker, and Z. Ghahramani, eds., 2002.
[15] M. Belkin and P. Niyogi, *Laplacian eigenmaps for dimensionality reduction and data representation*, Neural Computation, 15 (2003), pp. 1373–1396.
[16] A. A. Benczúr and D. R. Karger, *Approximating s-t minimum cuts in $\tilde{O}$(n2) time*, in Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, STOC '96, New York, NY, USA, 1996, Association for Computing Machinery, p. 47–55.
[17] M. Benzi, W. Joubert, and G. Mateescu, *Numerical experiments with parallel orderings for ILU preconditioners*, Electronic Transactions on Numerical Analysis, 8 (1999), pp. 88–114.
[18] M. Benzi, D. B. Szyld, and A. van Duin, *Orderings for incomplete factorizations of nonsymmtric matrices*, SIAM Journal on Scientific Computing, 20 (1999), pp. 1652–1670.
[19] M. Benzi and Tůma, *Orderings for factorized sparse approximate inverse preconditioners*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1851–1868.
[20] C. M. Bishop, *Pattern Recognition and Machine Learning*, Information Science and Statistics, Springer, 2006.
[21] M. Bollhöfer, *A robust ILU with pivoting based on monitoring the growth of the inverse factors*, Linear Algebra and its Applications, 338 (2001), pp. 201–213.
[22] M. Bollhöfer, M. J. Grote, and O. Schenk, *Algebraic multilevel preconditioner for the Helmholtz equation in heterogeneous media*, SIAM Journal on Scientific Computing, 31 (2009), pp. 3781–3805.
[23] E. Botta and F. Wubs, *Matrix Renumbering ILU: an effective algebraic multilevel ILU*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 1007–1026.
[24] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Mathematics of Computation, 31 (1977), pp. 333–290.
[25] ———, *Multiscale scientific computation: review 2001*, in Multiscale and multiresolution methods, T. Barth, T. Chan, and R. Haimes, eds., vol. 20 of Lect. Notes Comput. Sci. Eng., Springer-Verlag, 2002, pp. 3–95.
[26] R. Bridson and W. P. tang, *Ordering, anisotropy, and factored sparse approximate inverses*, SIAM Journal on Scientific Computing, 21 (1999).
[27] ———, *A structural diagnosis of some IC orderings*, SIAM Journal on Scientific Computing, 22 (2001).
[28] W. L. Briggs, V. E. Henson, and S. F. Mc Cormick, *A multigrid tutorial*, SIAM, Philadelphia, PA, 2000. Second edition.
[29] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, *Geometric deep learning: Going beyond euclidean data*, IEEE Signal Processing Magazine, 34 (2017), pp. 18–42.
[30] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, *Spectral networks and locally connected networks on graphs*, 2013.
[31] P. J. Burt and E. H. Adelson, *The Laplacian pyramid as a compact image code*, in Readings in Computer Vision, M. A. Fischler and O. Firschein, eds., Morgan Kaufmann, San Francisco (CA), 1987, pp. 671–679.
[32] C. Cangea, P. Velickovic, N. Jovanovic, T. Kipf, and P. Liò, *Towards sparse hierarchical graph classifiers*, ArXiv, abs/1811.01287 (2018).
[33] S. Cao, W. Lu, and Q. Xu, *Grarep: Learning graph representations with global structural information*, in Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM '15, New York, NY, USA, 2015, ACM, pp. 891–900.
[34] A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, and P. Tiwari, *The electrical resistance of a graph captures its commute and cover times*, Computational Complexity, 6 (1996), pp. 312–340.
[35] H. Chen, B. Perozzi, Y. Hu, and S. Skiena, *HARP: Hierarchical representation learning for networks*, Proceedings of the AAAI Conference on Artificial Intelligence, 32 (2018).

[36] J. Chen and I. Safro, *Algebraic distance on graphs*, SIAM J. Sci. Comput., 33 (2011), p. 3468–3490.
[37] K. Chen, *Matrix Preconditioning Techniques and Applications*, Cambridge University Press, Cambridge, UK,, 2005.
[38] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, Journal of Computational and Applied Mathematics, 86 (1997), pp. 387–414.
[39] S. Clift and W. Tang, *Weighted graph based ordering techniques for preconditioned conjugate gradient methods*, BIT, 35 (1995), pp. 30–47.
[40] E. D'Azevedo, P. Forsyth, and W. Tang, *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, SIAM J. Matrix Anal. Applic., 13 (1992), pp. 944–961.
[41] M. Defferrard, X. Bresson, and P. Vandergheynst, *Convolutional neural networks on graphs with fast localized spectral filtering*, in Proceedings of the 30th International Conference on Neural Information Processing Systems, NIPS'16, Red Hook, NY, USA, 2016, Curran Associates Inc., p. 3844–3852.
[42] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng, *Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding*, in International Conference on Learning Representations, 2020.
[43] I. S. Dhillon, Y. Guan, and B. Kulis, *Weighted graph cuts without eigenvectors a multilevel approach*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 29 (2007), pp. 1944–1957.
[44] P. D. Dobson and A. J. Doig, *Distinguishing enzyme structures from non-enzymes without alignments*, Journal of Molecular Biology, 330 (2003), pp. 771–783.
[45] S. Doi and A. Lichnewsky, *A graph theory approach for analyzing the effects of ordering on ILU preconditioning*, Tech. Rep. 1452, INRIA, Rocquencourt, France, 1991.
[46] F. Dorfler and F. Bullo, *Kron reduction of graphs with applications to electrical networks*, IEEE Transactions on Circuits and Systems I: Regular Papers, 60 (2013), pp. 150–163.
[47] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff, *Fast approximation of matrix coherence and statistical leverage*, J. Mach. Learn. Res., 13 (2012), p. 3475–3506.
[48] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, *Convolutional networks on graphs for learning molecular fingerprints*, in Advances in neural information processing systems, 2015, pp. 2224–2232.
[49] W. Ellens, F. Spieksma, P. Van Mieghem, A. Jamakovic, and R. Kooij, *Effective graph resistance*, Linear algebra and its applications, 435 (2011), pp. 2491–2506.
[50] M. Fahrbach, G. Goranci, R. Peng, S. Sachdeva, and C. Wang, *Faster graph embeddings via coarsening*, in Proceedings of the 37th International Conference on Machine Learning, 2020.
[51] W. Fan, J. Li, X. Wang, and Y. Wu, *Query preserving graph compression*, in Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 2012, pp. 157–168.
[52] M. Fiedler, *Algebraic connectivity of graphs*, Czechoslovak Mathematical Journal, 23 (1973), pp. 298–305.
[53] H. Gao and S. Ji, *Graph u-nets*, in ICML, 2019.
[54] D. Goldfarb, *Modification methods for inverting matrices and solving systems of linear algebraic equations*, Mathematics of Computation, 26 (1972), pp. 829–852.
[55] G. H. Golub and C. F. V. Loan, *Matrix Computations, 4th edition*, Johns Hopkins University Press, Baltimore, MD, 4th ed., 2013.
[56] P. Goyal and E. Ferrara, *Graph embedding techniques, applications, and performance: A survey*, Knowledge-Based Systems, 151 (2018), pp. 78–94.
[57] A. Grover and J. Leskovec, *Node2vec: Scalable feature learning for networks*, in Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, New York, NY, USA, 2016, ACM, pp. 855–864.
[58] W. Hackbusch, *Multi-Grid Methods and Applications*, vol. 4 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, 1985.
[59] W. Hamilton, Z. Ying, and J. Leskovec, *Inductive representation learning on large graphs*, in Advances in neural information processing systems, 2017, pp. 1024–1034.
[60] W. L. Hamilton, R. Ying, and J. Leskovec, *Representation learning on graphs: Methods and applications*, 2017.
[61] M. Henaff, J. Bruna, and Y. LeCun, *Deep convolutional networks on graph-structured data*, 2015.
[62] G. B. Hermsdorff and L. Gunderson, *A unifying framework for spectrum-preserving graph*

sparsification and coarsening, in Advances in Neural Information Processing Systems, 2019, pp. 7736–7747.

[63] Y. F. Hu and R. J. Blake, *Load balancing for unstructured mesh applications*, Nova Science Publishers, Inc., Commack, NY, USA, 2001, pp. 117–148.

[64] Y. Jin, A. Loukas, and J. JaJa, *Graph coarsening with preserved spectral properties*, in International Conference on Artificial Intelligence and Statistics, PMLR, 2020, pp. 4452–4462.

[65] M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford, *Single pass spectral sparsification in dynamic streams*, in Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on, IEEE, 2014, pp. 561–570.

[66] D. R. Karger, *Random sampling in cut, flow, and network design problems*, in Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, STOC '94, New York, NY, USA, 1994, Association for Computing Machinery, p. 648–657.

[67] G. Karypis and V. Kumar, *Multilevel graph partitioning schemes*, in Proc. 24th Intern. Conf. Par. Proc., III, CRC Press, 1995, pp. 113–122.

[68] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal, 29 (1970), pp. 291–307.

[69] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, 2016.

[70] B. Knyazev, G. W. Taylor, and M. R. Amer, *Understanding attention and generalization in graph neural networks*, 2019.

[71] R. Krauthgamer, *Sketching graphs and combinatorial optimization (invited talk)*, in 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[72] G. Kron, *Tensor Analysis of Networks*, Wiley, New York, 1939.

[73] J. Lee, I. Lee, and J. Kang, *Self-attention graph pooling*, in ICML, 2019.

[74] J. Liang, S. Gurukar, and S. Parthasarathy, *MILE: a multi-level framework for scalable graph embedding*, ArXiv preprint arXiv:1802.09612 [cs.AI], (2018).

[75] Y. Liu, T. Safavi, A. Dighe, and D. Koutra, *Graph summarization methods and applications: A survey*, ACM Computing Surveys (CSUR), 51 (2018), pp. 1–34.

[76] A. Loukas, *Graph reduction with spectral and cut guarantees*, Journal of Machine Learning Research, 20 (2019), pp. 1–42.

[77] T. Ma and J. Chen, *Unsupervised learning of graph hierarchical abstractions with differentiable coarsening and optimal transport*, in Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, 2021.

[78] S. MacLachlan, D. Osei-Kuffuor, and Y. Saad, *Modification and compensation strategies for threshold-based incomplete factorizations*, SIAM Journal on Scientific Computing, 34 (2012), pp. A48–A75.

[79] S. MacLachlan and Y. Saad, *Greedy coarsening strategies for non-symmetric problems*, SIAM Journal on Scientific Computing, 29 (2007), pp. 2115–2143.

[80] ———, *A greedy strategy for coarse-grid selection*, SIAM Journal on Scientific Computing, 29 (2007), pp. 1825–1853.

[81] R. M. Martin, *Electronic Structure, basic theory and practical methods*, Cambridge University Press, Cambridge, UK, 2004.

[82] J. Mayer, *A multilevel Crout ILU preconditioner with pivoting and row permutation*, Numerical Linear Algebra with Applications, 14 (2007), pp. 771–789.

[83] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, *Geometric deep learning on graphs and manifolds using mixture model cnns*, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[84] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, *Weisfeiler and leman go neural: Higher-order graph neural networks.*, in AAAI, 2019, pp. 4602–4609.

[85] A. C. Muresan and Y. Notay, *Analysis of aggregation-based multigrid*, SIAM Journal on Scientific Computing, 30 (2008), pp. 1082–1103.

[86] S. Navlakha, R. Rastogi, and N. Shrivastava, *Graph summarization with bounded error*, in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008, pp. 419–432.

[87] M. E. Newman, *A measure of betweenness centrality based on random walks*, Social networks, 27 (2005), pp. 39–54.

[88] M. Niepert, M. Ahmed, and K. Kutzkov, *Learning convolutional neural networks for graphs*, in Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48, ICML'16, JMLR.org, 2016, p. 2014–2023.

[89] Y. Notay, *An aggregation-based algebraic multigrid method*, Electronic Transactions on Numerical Analysis, 37 (2010), pp. 123–146.
[90] D. Osei-Kuffuor, R. Li, and Y. Saad, *Matrix reordering using multilevel graph coarsening for ILU preconditioning*, SIAM Journal on Scientific Computing, 37 (2015), pp. A391–A419.
[91] B. Perozzi, R. Al-Rfou, and S. Skiena, *Deepwalk: Online learning of social representations*, in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, 2014, pp. 701–710.
[92] A. Pothen, H. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 430–452.
[93] D. Powers, *Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation*, Mach. Learn. Technol., 2 (2008).
[94] B. Quiring and P. S. Vassilevski, *Multilevel graph embedding*, Numerical Linear Algebra with Applications, 28 (2021), p. e2326.
[95] H. r. Fang and Y. Saad, *Hypergraph-based multilevel matrix approximation for text information retrieval*, in Proceedings of the ACM International Conference on Information and Knowledge Management, 2010, J. H. et al., ed., 2010, pp. 1597–1600.
[96] H. r. Fang, S. Sakellaridi, and Y. Saad, *Multilevel manifold learning with application to spectral clustering*, in Proceedings of the ACM International Conference on Information and Knowledge Management, 2010, J. H. et al., ed., 2010, pp. 419–428.
[97] G. Ranjan and Z.-L. Zhang, *Geometry of complex networks and topological centrality*, Physica A: Statistical Mechanics and its Applications, 392 (2013), pp. 3833–3845.
[98] D. Ron, I. Safro, and A. Brandt, *Relaxation-based coarsening and multiscale graph organization*, SIAM Multiscale Modelling and Simulations, 9 (2011), pp. 407–423.
[99] R. Roth, *On the eigenvectors belonging to the minimum eigenvalue of an essentially nonnegative symmetric matrix with bipartite graph*, Linear Algebra and its Applications, 118 (1989), pp. 1–10.
[100] S. Roweis and L. Saul, *Nonlinear dimensionality reduction by locally linear embedding*, Science, 290 (2000), pp. 2323–2326.
[101] A. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid Methods, S. McCormick, ed., vol. 3 of Frontiers in Applied Mathematics, SIAM, 1987, ch. 4, pp. 73–130.
[102] Y. Saad, *Iterative Methods for Sparse Linear Systems, 2nd edition*, SIAM, Philadelpha, PA, 2003.
[103] ———, *Multilevel ILU with reorderings for diagonal dominance*, SIAM Journal on Scientific Computing, 27 (2005), pp. 1032–1057.
[104] Y. Saad, *Numerical Methods for Large Eigenvalue Problems-classics edition*, SIAM, Philadelphia, 2011.
[105] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numerical Linear Algebra with Applications, 9 (2002).
[106] S. Sakellaridi, H. R. Fang, and Y. Saad, *Graph-based multilevel dimensionality reduction with applications to eigenfaces and latent semantic indexing*, in Proceedings of Int. Conf. Mach. Learn. Appls. (ICMLA), 2008, M. A. Wani, ed., IEEE comp. Soc., 2008, pp. 194–200.
[107] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, *Collective classification in network data*, AI magazine, 29 (2008), pp. 93–93.
[108] D.-H. Shin, D. Qian, and J. Zhang, *Cascading effects in interdependent networks*, IEEE Network, 28 (2014), pp. 82–87.
[109] D. I. Shuman, M. J. Faraji, and P. Vandergheynst, *A multiscale pyramid transform for graph signals*, IEEE Transactions on Signal Processing, 64 (2016), pp. 2119–2134.
[110] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, *The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains*, IEEE Signal Processing Magazine, 30 (2013), pp. 83–98.
[111] M. Simonovsky and N. Komodakis, *Dynamic edge-conditioned filters in convolutional neural networks on graphs*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 3693–3702.
[112] D. A. Spielman and N. Srivastava, *Graph sparsification by effective resistances*, SIAM Journal on Computing, 40 (2011), pp. 1913–1926.
[113] D. A. Spielman and S.-H. Teng, *Spectral sparsification of graphs*, SIAM Journal on Computing, 40 (2011), pp. 981–1025.
[114] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Princeton, NJ, 1994.

[115] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, *Line: Large-scale information network embedding*, in Proceedings of the 24th International Conference on World Wide Web, WWW '15, Republic and Canton of Geneva, CHE, 2015, International World Wide Web Conferences Steering Committee, p. 1067–1077.

[116] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, San Diego, CA, 2001.

[117] P. Van Mieghem, K. Devriendt, and H. Cetinay, *Pseudoinverse of the Laplacian and best spreader node in a network*, Physical Review E, 96 (2017), p. 032311.

[118] N. Vannieuwenhoven and K. Meerbergen, *IMF: An incomplete multifrontal LU-factorization for element-structured sparse linear systems*, SIAM Journal on Scientific Computing, 35 (2013), pp. A270–A293.

[119] P. Vaněk, M. Brezina, and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation*, Numerische Mathematik, 88 (2001), pp. 559–579.

[120] P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems*, Computing, 56 (1996), pp. 179–196.

[121] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2017.

[122] N. Wale and G. Karypis, *Comparison of descriptor spaces for chemical compound retrieval and classification*, in Sixth International Conference on Data Mining (ICDM'06), Dec 2006, pp. 678–689.

[123] X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, *Community preserving network embedding*, in Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), 2017, pp. 203–209.

[124] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, *How powerful are graph neural networks?*, 2018.

[125] P. Yanardag and S. Vishwanathan, *Deep graph kernels*, in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, New York, NY, USA, 2015, Association for Computing Machinery, p. 1365–1374.

[126] R. Ying, J. You, C. Morris, X. Ren, W. L. Hamilton, and J. Leskovec, *Hierarchical graph representation learning with differentiable pooling*, in Proceedings of the 32Nd International Conference on Neural Information Processing Systems, NIPS'18, USA, 2018, Curran Associates Inc., pp. 4805–4815.

[127] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, *An end-to-end deep learning architecture for graph classification*, in AAAI, 2018.