LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-779424

Properties of the Graph Modularity Matrix and its Applications

B. G. Quiring, P. S. Vassilevski

June 26, 2019

Disclaimer

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

PROPERTIES OF THE GRAPH MODULARITY MATRIX AND ITS APPLICATIONS

BENJAMIN QUIRING AND PANAYOT S. VASSILEVSKI

Abstract. We study the popular modularity matrix and respective functional ([5]) used in connection with graph clustering and derive some properties useful when performing vertex aggregation of the associated graph. These properties are employed in the derivation of a multilevel parallel pairwise aggregation algorithm. Some illustrative examples which include algebraic multigrid (AMG) coarsening that follows strong direction of anisotropy in finite element problems as well as comparative performance results of the studied algorithm are presented.

1. Introduction

The modularity functional associated with a weighted graph is considered the *state-of-the-art* tool to derive algorithms for graph clustering, which exploit its maximization. One of the best clustering algorithms is of multilevel nature and was originally proposed in [1]. It is known as the *Louvain algorithm*. Since then it has found a number of extensions ([13], [14]) that include implementations on multicore computers, which makes it attractive for very large-scale graphs. Other approaches for graph clustering exploit spectral information of matrices associated with the graph, which may become prohibitively expensive in the large-scale case. Louvain's algorithm is attractive and efficient due to its multilevel nature, where at every step one performs only local operations. It resembles the so-called *relaxation* or *smoothing* iterations in the well-known multigrid method. The second immediate similarity is with the *aggregation*-based multigrid, namely the creation of the coarse graph obtained by forming sets, which we refer to also as aggregates (syn. communities, clusters, coarse vertices), of fine-level vertices and computing weights for the coarse edges that connect any pair of coarse vertices.

The purpose of the present work is to make these connections more explicit and relate them with the change of the modularity matrix at every step of adding vertices (or previous level clusters) to a current cluster.

Our objective is two-fold: on one hand we can use the modularity change matrix as weights in graph clustering (which is basically weighted graph partitioning), and on the other hand, we can use techniques employed in aggregation-based AMG (algebraic multigrid) to more efficiently (including in parallel) compute the coarse-level quantities to hopefully better speed up the resulting clustering algorithm. As a side result, by appropriately assigning weights to the edges of the sparsity graph of a given

Date: August 24, 2018–beginning; Today is June 27, 2019.

1991 *Mathematics Subject Classification.* 65F10, 65N20, 65N30.

Key words and phrases. graphs, multilevel algorithms, graph modularity.

s.p.d. matrix A, we can employ the respective modularity-based coarsening to generate matrix-dependent hierarchy of aggregates for use in AMG algorithms, or simply as physics-based domain/mesh partitioners. The latter topic is only illustrated by some examples to demonstrate its potential, and is left for a more detailed study elsewhere.

The remainder of the present paper is structured as follows. In Section 2 we introduce the modularity matrix and formulate its main properties of our interest. We also study the modularity change matrix after an aggregation step is performed. It contains the properties of the modularity change matrix that are of our main interest. Section 3 contains a simple parallel pairwise aggregation algorithm that is recursive, i.e., its use is to generate a hierarchy of coarse sets of aggregates as well as some heuristics which can speed up and further parallelize the method. Comparisons of this method with Louvain's method is included here. Section 4 illustrates the potential of the proposed method as a coarsening algorithm for AMG which detects communities which follow the dominant anisotropy in finite element matrices. At the end, in Section 5, we draw some conclusions.

2. The modularity matrix: definitions and properties

Consider a symmetric $n \times n$ sparse matrix $A = (a_{ij})$ with positive rowsums

$$r_i = \sum_j a_{ij} > 0 \text{ for all } i = 1,\ 2, \ldots,\ n. \tag{2.1}$$

If we introduce the vector $\mathbf{r} = (r_i) \in \mathbb{R}^n$ and the constant vector $\mathbf{1} = (1) \in \mathbb{R}^n$, we have that

$$\mathbf{r} = A\mathbf{1}.$$

Let the total rowsum be

$$T = \sum_i \sum_j a_{ij} = \sum_i r_i > 0. \tag{2.2}$$

The following matrix is referred to as the modularity matrix, [5],

$$B = A - \frac{1}{T}\mathbf{r}\mathbf{r}^T. \tag{2.3}$$

It is clear that $B = (b_{ij})$ has entries

$$b_{ij} = a_{ij} - \frac{1}{T} r_i r_j.$$

From the definition of B, we have

$$B\mathbf{1} = A\mathbf{1} - \mathbf{r}\frac{\mathbf{r}^T\mathbf{1}}{T} = \mathbf{r} - \mathbf{r} = \mathbf{0}.$$

We see that B has zero rowsums and hence must have both negative and positive entries in each row.

In what follows we need the normalized rowsums

$$\alpha_i = \frac{r_i}{T} = \frac{1}{T}\sum_j a_{ij}. \tag{2.4}$$

2.1. **Performing aggregation: modularity change matrix.** Let $\{\mathcal{A}\}$ be a non-overlapping partition of the vertex set $\{1,\ 2,\ \ldots,\ n\}$.

For each aggregate $\mathcal{A}$, we define

$$\alpha_{\mathcal{A}} = \frac{1}{T}\sum_{i\in\mathcal{A}} r_i = \sum_{i\in\mathcal{A}} \alpha_i = \frac{1}{T}\sum_{i\in\mathcal{A}}\sum_{j} a_{ij}. \tag{2.5}$$

Consider also the partial rowsums

$$d_{\mathcal{A}} = \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{A}} a_{ij}. \tag{2.6}$$

We also define the aggregate edges (pairs of connected aggregates $\mathcal{A}$ and $\mathcal{B}$). Let

$$a_{\mathcal{A}\mathcal{B}} = \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{B}} a_{ij} \tag{2.7}$$

The popular modularity functional associated with the partitioning $\{\mathcal{A}\}$ reads ([5]),

$$\mathcal{Q} = \frac{1}{T}\sum_{\mathcal{A}}\sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{A}} b_{ij} = \frac{1}{T}\sum_{\mathcal{A}}\sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{A}}\left(a_{ij} - \frac{1}{T}r_i r_j\right) = \sum_{\mathcal{A}}\left(\frac{d_{\mathcal{A}}}{T} - \alpha_{\mathcal{A}}^2\right). \tag{2.8}$$

Lemma 2.1. *Assume that the distinct aggregates $\mathcal{A}$ and $\mathcal{B}$ have merged and formed a new aggregate denoted by $(\mathcal{A}\mathcal{B})$. Then the following formula holds for the change of modularity*

$$\Delta\mathcal{Q}_{\mathcal{A}\mathcal{B}} \triangleq \mathcal{Q}_{new} - \mathcal{Q}_{old} = 2\left(\frac{a_{\mathcal{A}\mathcal{B}}}{T} - \alpha_{\mathcal{A}}\alpha_{\mathcal{B}}\right).$$

Proof. Based on formula (2.8), we have

$$\begin{aligned}
\Delta\mathcal{Q}_{\mathcal{A}\mathcal{B}} &= \mathcal{Q}_{new} - \mathcal{Q}_{old} \\
&= \left[\left(\frac{1}{T}d_{(\mathcal{A}\mathcal{B})} + \frac{1}{T}\sum_{\mathcal{C}\neq\mathcal{A},\mathcal{B},(\mathcal{A}\mathcal{B})} d_{\mathcal{C}}\right) - \left(\alpha_{(\mathcal{A}\mathcal{B})}^2 + \sum_{\mathcal{C}\neq\mathcal{A},\mathcal{B},(\mathcal{A}\mathcal{B})}\alpha_{\mathcal{C}}^2\right)\right] \\
&\quad - \left[\left(\frac{1}{T}(d_{\mathcal{A}} + d_{\mathcal{B}}) + \frac{1}{T}\sum_{\mathcal{C}\neq\mathcal{A},\mathcal{B},(\mathcal{A}\mathcal{B})} d_{\mathcal{C}}\right) - \left(\alpha_{\mathcal{A}}^2 + \alpha_{\mathcal{B}}^2 + \sum_{\mathcal{C}\neq\mathcal{A},\mathcal{B},(\mathcal{A}\mathcal{B})}\alpha_{\mathcal{C}}^2\right)\right] \\
&= \left(\frac{1}{T}d_{(\mathcal{A}\mathcal{B})} - \alpha_{(\mathcal{A}\mathcal{B})}^2\right) - \left(\frac{1}{T}(d_{\mathcal{A}} + d_{\mathcal{B}}) - (\alpha_{\mathcal{A}}^2 + \alpha_{\mathcal{B}}^2)\right).
\end{aligned}$$

Using now that $d_{(\mathcal{A}\mathcal{B})} = d_{\mathcal{A}} + d_{\mathcal{B}} + 2a_{\mathcal{A}\mathcal{B}}$ and $\alpha_{(\mathcal{A}\mathcal{B})} = \alpha_{\mathcal{A}} + \alpha_{\mathcal{B}}$ in the above identity, we obtain

$$\begin{aligned}
\Delta\mathcal{Q}_{\mathcal{A}\mathcal{B}} &= \left[\frac{1}{T}(d_{\mathcal{A}} + d_{\mathcal{B}} + 2a_{\mathcal{A}\mathcal{B}}) - (\alpha_{\mathcal{A}} + \alpha_{\mathcal{B}})^2\right] - \left[\frac{1}{T}(d_{\mathcal{A}} + d_{\mathcal{B}}) - (\alpha_{\mathcal{A}}^2 + \alpha_{\mathcal{B}}^2)\right] \\
&= 2\left(\frac{a_{\mathcal{A}\mathcal{B}}}{T} - \alpha_{\mathcal{A}}\alpha_{\mathcal{B}}\right).
\end{aligned}$$

□

Remark 2.1. *We may think of the original vertices as single-point aggregates. Then, the change of modularity after merging vertex i and vertex j is a scaled version of the entry b_{ij} of the modularity matrix, i.e., we have*

$$\Delta\mathcal{Q}_{ij} = 2\left(\frac{a_{ij}}{T} - \alpha_i\alpha_j\right) = \frac{2}{T}\left(a_{ij} - \frac{1}{T}r_i r_j\right) = \frac{2}{T}b_{ij}.$$

Since $\sum_j b_{ij} = 0$, we always have negative and positive values of $\Delta\mathcal{Q}_{ij}$. We choose to merge i and j only for pairs i, j with positive values of $\Delta\mathcal{Q}_{ij}$.

Additionally, we may regard $\Delta\mathcal{Q}_{ij}$ as the entries of a matrix denoted $\Delta\mathcal{Q}$, and will do so later.

Lemma 2.2. *Consider three aggregates $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$. Assume that $\mathcal{B}$ and $\mathcal{C}$ have merged to a new aggregate denoted by $(\mathcal{BC})$. The following* additive *property holds for the change of modularity.*

$$\Delta\mathcal{Q}_{\mathcal{A}(\mathcal{BC})} = \Delta\mathcal{Q}_{\mathcal{AB}} + \Delta\mathcal{Q}_{\mathcal{AC}}.$$

Proof. We have

$$\begin{aligned}
\Delta\mathcal{Q}_{\mathcal{AB}} &= 2\left(\frac{a_{\mathcal{AB}}}{T} - \alpha_{\mathcal{A}}\alpha_{\mathcal{B}}\right)\\
\Delta\mathcal{Q}_{\mathcal{AC}} &= 2\left(\frac{a_{\mathcal{AC}}}{T} - \alpha_{\mathcal{A}}\alpha_{\mathcal{C}}\right)\\
\Delta\mathcal{Q}_{\mathcal{A}(\mathcal{BC})} &= 2\left(\frac{a_{\mathcal{A}(\mathcal{BC})}}{T} - \alpha_{\mathcal{A}}\alpha_{(\mathcal{BC})}\right).
\end{aligned}$$

Next, we use the properties

$$\begin{aligned}
\alpha_{(\mathcal{BC})} &= \alpha_{\mathcal{B}} + \alpha_{\mathcal{C}},\\
a_{\mathcal{A}(\mathcal{BC})} &= \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{B}\cup\mathcal{C}} a_{ij}\\
&= \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{B}} a_{ij} + \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{C}} a_{ij}\\
&= a_{\mathcal{AB}} + a_{\mathcal{AC}}.
\end{aligned}$$

The latter shows that

$$\alpha_{\mathcal{A}}\alpha_{(\mathcal{BC})} = \alpha_{\mathcal{A}}\alpha_{\mathcal{B}} + \alpha_{\mathcal{A}}\alpha_{\mathcal{C}}.$$

Therefore,

$$\begin{aligned}
\Delta\mathcal{Q}_{\mathcal{A}(\mathcal{BC})} &= 2\left(\frac{a_{\mathcal{A}(\mathcal{BC})}}{T} - \alpha_{\mathcal{A}}\alpha_{(\mathcal{BC})}\right)\\
&= 2\left(\frac{a_{\mathcal{AB}} + a_{\mathcal{AC}}}{T} - \alpha_{\mathcal{A}}(\alpha_{\mathcal{B}} + \alpha_{\mathcal{C}})\right)\\
&= 2\left(\frac{a_{\mathcal{AB}}}{T} - \alpha_{\mathcal{A}}\alpha_{\mathcal{B}}\right) + 2\left(\frac{a_{\mathcal{AC}}}{T} - \alpha_{\mathcal{A}}\alpha_{\mathcal{C}}\right)\\
&= \Delta\mathcal{Q}_{\mathcal{AB}} + \Delta\mathcal{Q}_{\mathcal{AC}}.
\end{aligned}$$

□

Lemma 2.2 enables us to prove the following result.

Theorem 2.1. *Consider a fine partitioning $\{i\}$ of size n and a coarse one $\{\mathcal{A}\}$ of size n_c, obtained by merging elements from the fine one. I.e., each $\mathcal{A}$ consists of some fine-level vertices and the set $\{\mathcal{A}\}$ provides their nonoverlapping partitioning. Let $\mathbf{1}_{\mathcal{A}}$ for each coarse vertex $\mathcal{A}$ be a vector of size n (the number of fine vertices) with nonzero entries equal to 1 at positions $i \in \mathcal{A}$ and zero elsewhere. Let $\mathbf{1}_{\mathcal{A}}$ form the $\mathcal{A}$th column of the $n \times n_c$ rectangular matrix P referred to as the piecewise constant interpolation (or prolongation) matrix. Finally, consider the modularity change matrix $\Delta\mathcal{Q} = (\Delta\mathcal{Q}_{ij})$ and the coarse counterpart $\Delta\mathcal{Q}_c = (\Delta\mathcal{Q}_{\mathcal{AB}})$. Then, the following relation holds*

$$\Delta\mathcal{Q}_c = P^T(\Delta\mathcal{Q})P.$$

Proof. Using the symmetry of the $\Delta\mathcal{Q}$ matrices and breaking each aggregate $\mathcal{A}$ into the union of its individual fine vertices via the additivity proven in Lemma 2.2, we have

$$\Delta\mathcal{Q}_{\mathcal{AB}} = \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{B}} \Delta\mathcal{Q}_{ij} \tag{2.9}$$

$$= \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{B}} 2\left(\frac{1}{T}a_{ij} - \alpha_i\alpha_j\right) \tag{2.10}$$

$$= 2\left(\frac{1}{T}a_{\mathcal{AB}} - \alpha_{\mathcal{A}}\alpha_{\mathcal{B}}\right). \tag{2.11}$$

Consider now the $(\mathcal{A},\mathcal{B})$-entry of the triple matrix product $P^T(\Delta\mathcal{Q})P$. Letting $P = (P_{i\mathcal{A}})$ where $P_{i\mathcal{A}} = 1$ if $i \in \mathcal{A}$ and $P_{i\mathcal{A}} = 0$ otherwise, we have

$$\begin{aligned}
\left(P^T(\Delta\mathcal{Q})P\right)_{\mathcal{AB}} &= \sum_i\sum_j P_{i\mathcal{A}}(\Delta\mathcal{Q})_{ij}P_{j\mathcal{B}} \\
&= \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{B}}(\Delta\mathcal{Q})_{ij} \\
&= \sum_{i\in\mathcal{A}}\sum_{j\in\mathcal{B}} 2\left(\frac{1}{T}a_{ij} - \alpha_i\alpha_j\right) \\
&= \Delta\mathcal{Q}_{\mathcal{AB}},
\end{aligned}$$

which shows the desired result. □

Remark 2.2. *We remark that the triple-matrix product $P^T(\Delta\mathcal{Q})P$ is performed in practice exploiting the sparsity of $A = (a_{ij})$ as follows*

$$P^T\Delta\mathcal{Q}P = \frac{2}{T}P^T\left(A - \frac{\mathbf{r}}{T}\mathbf{r}^T\right)P = \frac{2}{T}\left(P^TAP - \frac{P^T\mathbf{r}}{T}(P^T\mathbf{r})^T\right).$$

That is, we perform separately the triple-matrix product of sparse matrices $A_c = P^TAP$ and the matrix-vector product $\mathbf{r}_c = P^T\mathbf{r}$. Note that the coarse modularity change matrix $(\Delta\mathcal{Q})_c$ has the same rank-one modified form as $\Delta\mathcal{Q} = \frac{2}{T}A - \frac{2}{T^2}\mathbf{r}\mathbf{r}^T$, i.e.,

$$(\Delta\mathcal{Q})_c = \frac{2}{T}A_c - \frac{2}{T^2}\mathbf{r}_c\mathbf{r}_c^T.$$

This means, that in practice we can keep the sparse matrix A and the vector $\mathbf{r}$, from which we can easily get any entry of $\Delta\mathcal{Q}$. The same holds at any coarse level; i.e., we can form and keep the sparse matrix $A_c = P^T AP$ and the coarse vector $\mathbf{r}_c = P^T\mathbf{r}$, from which we can obtain any required entry of $(\Delta\mathcal{Q})_c$.

Remark 2.3. *Consider the coarse ones vector $\mathbf{1}_c = (1) \in \mathbb{R}^{n_c}$. Since $P\mathbf{1}_c = \mathbf{1}$–the fine-level unit vector, we have that $(\Delta\mathcal{Q})_c\mathbf{1}_c = P^T(\Delta\mathcal{Q})(P\mathbf{1}_c) = P^T(\Delta\mathcal{Q})\mathbf{1} = 0$, i.e., the zero rowsum property of $\Delta\mathcal{Q}$ is maintained at coarse levels.*

Corollary 2.1. *Remark 2.3 implies that a recursive multilevel aggregation algorithm which monotonically increases the modularity can stop only when we reach a situation when all $\Delta\mathcal{Q}_{\mathcal{AB}} \leq 0$ for $\mathcal{A} \neq \mathcal{B}$ with positive entries being only on its diagonal, i.e., $\Delta\mathcal{Q}_{\mathcal{AA}} \geq 0$. The latter means that at the final coarse level $\Delta\mathcal{Q}$ is actually a (dense) weighted graph Laplacian matrix, where the modularity $\mathcal{Q}$ is related to the sum of the diagonal diagonal entries of $\Delta\mathcal{Q}$,*

$$\mathcal{Q} = \frac{1}{2}\sum_{\mathcal{A}} \Delta\mathcal{Q}_{\mathcal{AA}}.$$

The latter formula actually holds at all stages of aggregation.

3. Aggregating vertices based on edge weights: a parallel algorithm

We are given a graph G with a vertex set $\{1, 2, \ldots, n\}$ and an edge set $E = \{e = (i, j)\} \subset V \times V$. Each edge e also comes with a weight w_e. Although the algorithm can be written for general weights, here our focus on the choice $w_e = \Delta\mathcal{Q}_{ij},\ e = (i, j)$, where $\Delta\mathcal{Q}$ is the modularity change matrix from the previous section and where $a_{ij} = 1$, for example. Our goal is to create coarse graph by aggregating vertices into aggregates $\mathcal{A}$ giving priority when merging for edges with a larger weight.

A simple parallel algorithm (similar to [6]) can be formulated as follows.

Algorithm 3.1 (Parallel pairwise aggregation using $\Delta\mathcal{Q}$ weights)**.**

- *Given A and $\mathbf{r}$, compute a vector $\mathbf{p} = (p_i)$, which stands for pointer. For each vertex i,*

  $$p_i = \underset{j\,:\,i\neq j}{\operatorname{argmax}}\{\Delta\mathcal{Q}_{ij} \mid \Delta\mathcal{Q}_{ij} > 0\}.$$

  Vertex i "points" to vertex p_i; this is the neighbor it would like to merge with most. If no j gives $\Delta\mathcal{Q}_{ij} > 0$, then $p_i = -1$. (Here, we exploit the sparsity of A as if there is not an edge between i and j then $\Delta\mathcal{Q}_{ij}$ is negative, and we will not merge this pair. That is, we only need to consider$\Delta\mathcal{Q}_{ij}$ for j adjacent to i.) *Note that $\mathbf{p}$ may be computed in parallel.*
- *Based on computed $\mathbf{p}$, find the pairs i and j such that $p_i = j$ and $p_j = i$. Each such pair gives a locally maximal edge: an edge whose $\Delta\mathcal{Q}_{ij}$ value is maximal among all neighboring edges. Each pair (i, j) is to be merged into a single aggregate $\mathcal{A} = \{i, j\}$, and vertices not in such pairs are left alone as singleton aggregates.*
- *Construct the coarsening matrix $R = P^T$ via $\mathbf{p}$. R has a row for each aggregate $\mathcal{A}$, with non-zero values, 1, only in columns $i \in \mathcal{A}$.*
- *Compute the triple matrix product $A_c = RAP$ and the vector $\mathbf{r}_c = R\mathbf{r}$.*

- *If no more positive weights in the current $\Delta\mathcal{Q}$ matrix are available (i.e., $\Delta\mathcal{Q}$ is a weighted graph Laplacian), the hierarchy is complete, then Exit. Otherwise recur.*
- *On Exit, each $(A, \mathbf{r}, P)$, $(A_c, \mathbf{r}_c, P_c)$, etc. correspond to a level in the hierarchy.*

The algorithm may be modified also to go into the Exit state if the coarsening factor is not good enough or if the size of the coarsest level is small enough. If A is $n \times n$ and A_c is $m \times m$ then the coarsening factor is $\frac{m}{n}$.

3.1. **Some heuristics.** The number of merges at each coarsening step is what determines the speed of the algorithm. To improve this, we introduce a heuristic parametrized over a natural number k. In the above method we do one pass over each vertex to determine the neighbor it would like to merge with most. We modify this slightly:

- If $p_i = -1$ (i has not found a match to merge with) then we take
$$p_i = \underset{j\,:\,i\neq j,\, p_j=-1}{\operatorname{argmax}} \{\Delta\mathcal{Q}_{ij} \mid \Delta\mathcal{Q}_{ij} > 0\}.$$
If the set is empty, then p_i remains set to -1.
- Find the pairs where $p_i = j$ and $p_j = i$.
- Any i that does not belong to such a pair has p_i reset to -1. If i does belong to such a pair then p_i is left alone.
- repeat k times

Algorithm 3.1 is the case where $k = 1$.

This is the same as keeping a 'liveness' value for each edge, any any edge found to be maximal compared with its live neighbors is taken as a coarsening pair and it and all its (edge) neighbors are remove from examination. A similar heuristic can be performed where only the locally maximal edges are killed, and not their neighbors as well, though this leads to a method which does not perform *pairwise* merging.

These heuristics tend to only very slightly hurt the maximal modularity (for small k), but can significantly improve the coarsening factor. In cases where the coarsening factor is significantly poor, then partitioning may have to be halted early, thus not achieving a good $\mathcal{Q}$ value, so improving the coarsening factor can actually increase the maximal $\mathcal{Q}$ when actually partitioning.

Additionally, changing the random weights at every iteration appears to improve the coarsening factor without hurting the modularity. That is, we compute $\Delta\mathcal{Q}_{ij}$ as

$$\Delta\mathcal{Q}_{ij} = 2(\frac{a_{ij}}{T} - \alpha_i\alpha_j)x_ix_j \qquad \text{or} \qquad \Delta\mathcal{Q}_{ij} = 2(\frac{a_{ij}}{T}x_ix_j - \alpha_i\alpha_j)$$

where $\mathbf{x} = (x_i)$ is a vector with entries $x_i \in (1-\epsilon, 1+\epsilon)$ for some $\epsilon > 0$.

Other options such as aggregating leaves as a first step can also help improve the coarsening factor.

3.2. **Results.** We compare our proposed method in both a parallel and serial setting to an implementation of Louvain's method ([12]) by examining timings, maximum modularity scores, and the size of the coarsest level.

Dataset	$\|V\|$, $\|E\|$	max $\mathcal{Q}$	L. max $\mathcal{Q}$	# coarse	L. # coarse	time (s)	parallel time (s)	L. time (s)
ENZYMES_g479 [11]	28, 49	0.518	0.548	5	5	0.0	0.0	0.0
ENZYMES8 [11]	88, 113	0.640	0.665	6	6	0.0	0.0	0.0
ca-netscience [11]	379, 914	0.843	0.843	18	19	0.0	0.0	0.0
GD00_c [11]	638, 1025	0.710	0.714	18	18	0.0	0.0	0.0
G11 [11]	800, 1600	0.837	0.833	13	14	0.0	0.0	0.0
fe-4elt2 [11]	11K, 33K	0.908	0.910	30	30	0.0	0.0	0.0
road-usroads [11]	129K, 160K	0.979	0.979	191	190	0.6	0.4	0.8
roadNet-CA [11]	2M, 2.2M	0.992	0.992	410	385	10.1	5.0	19.2
road-germany-osm [11]	12M, 12M	0.962	0.963	1M	847K	122.7	35.3	145.1
socfb-B-anon [11]	3M, 21M	0.128	0.729	2.1M	321	28.4	25.1	73.5
soc-livejournal [11]	4M, 23M	0.210	0.770	1.2M	1702	80.8	66.2	221.9
soc-orkut [11]	3M, 106M	0.032	0.687	1.6M	45	180.9	119.3	327.4
soc-sinaweibo [11]	59M, 272M	0.000	0.554	58M	161	105.3	62.9	1016.8

TABLE 1. Comparison of the proposed method with an implementation of Louvain's algorithm. L. stands for Louvain. Coarsening was halted once the coarsening factor was 0.99. Parallel timings were done with 8 threads.

Table 1 summarizes the modularity of the presented algorithm (without any heuristics or optimizations), the modularity for a Louvain implementation [12], the size of the coarsest levels, and the times taken to compute the partitions.

The coarsening factor for the proposed method typically increases monotonically until hitting 1, though sometimes it hits a bottleneck, decreases, and increases until hitting 1. The results indicate that if the graph is well-suited to the proposed method (the coarsening factor does not approach 1 too quickly), then the computed partition is of high quality, and can be computed quickly. With the datasets `socfb-B-anon`, `soc-livejournal`, `soc-orkut`, and `soc-sinaweibo` the proposed method was not able to coarsen enough in order to obtain a good modularity before hitting a coarsening factor of 0.99. To improve the performance of the algorithm on graphs such as these as well as networks in general, heuristics can be added to quicken the coarsening, such as those in Section 3.1.

However, the social networks were still unable to be coarsened quickly (other graphs did get a speed up) despite the heuristics. This is likely due to the small number of outliers with high degree. The union of these nodes' neighborhoods make up a good portion of the network, and so prevent good coarsening. Although the pairwise method fails to run fast enough, it is able to approach the modularity of Louvain's method after running to completion, though not quite reach it. Despite failing timewise on large social networks, the proposed method appears to work quite well (both modularity-wise and time-wise) for simpler networks such as road networks or graphs given by problems on meshes of any size. On these type of graphs, our method achieves a similar modularity to the Louvain procedure and attains a speed-up as well, which can be further improved with the heuristics discussed.

4. Applications

4.1. **Using edge weights coming from adaptive AMG.** Consider a system $\widehat{A}\mathbf{x} = \mathbf{b}$ with a s.p.d. (symmetric positive definite) matrix $\widehat{A} = (\widehat{a}_{ij})$. In the so-called

adaptive AMG ([4], [3], [8]), also called *bootstrap* AMG ([2]), a vector $\mathbf{w} = (w_i)$ is constructed such that $\widehat{A}\mathbf{w} \approx 0$. This is done by performing iterations on the trivial system $\widehat{A}\mathbf{x} = 0$ starting with a random initial iterate. The iteration process is based on a convergent smoother (like Gauss-Seidel) or on an already available convergent solver to test its convergence properties. Note that the iterates are the actual errors (since the exact solution is 0), so one can monitor the error decay (in $\|.\|_{\widehat{A}}$-norm) during the iteration process. If the convergence is deemed unsatisfactory, the current iterate (after normalization) is chosen as $\mathbf{w}$. We note that in such a case we will have $\widehat{A}\mathbf{w} \approx 0$. Such near-null components of $\widehat{A}$ are referred to as *algebraically smooth* vectors associated with the solver we test.

Note that $\widehat{A}\mathbf{w} \approx 0$ means that for each i, we have $\sum_j \widehat{a}_{ij} w_j \approx 0$, which implies

$$-\sum_{j \neq i} w_i \widehat{a}_{ij} w_j \approx \widehat{a}_{ii} w_i^2 \geq 0.$$

Therefore, if we consider the matrix $A = (a_{ij})$ where

$$a_{ij} = -w_i \widehat{a}_{ij} w_j, \tag{4.1}$$

its rowsums will be non-negative. That is, we can use A as the adjacency matrix of the sparsity graph of the original matrix $\widehat{A}$ where a_{ij} can serve as edge weights, which can be positive and negative. Nevertheless, the positive rowsum property (2.1) of A is ensured, which was the only requirement for our modularity-based coarsening algorithm. For some PDE-discretization matrices one can simply take $\mathbf{w} = (1)$ which makes $a_{ij} = -\widehat{a}_{ij}$.

4.1.1. *Detecting dominant anisotropy in finite element matrices.* In our tests, we used the matrix $\widehat{A}$ coming from finite element discretization of the highly anisotropic diffusion equation

$$-\operatorname{div}(\epsilon I + \mathbf{b}\mathbf{b}^T)\nabla u) = f(\mathbf{x}),\ \mathbf{x} \in \Omega \subset \mathbb{R}^d\ (d = 2 \text{ or } 3).$$

For small $\epsilon > 0$, the dominant direction of the $d \times d$ diffusion tensor $\epsilon I + \mathbf{b}\mathbf{b}^T$ is $\mathbf{b} \in \mathbb{R}^d$. We choose

$$\mathbf{b} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix},$$

for some angle θ in two dimensions ($d = 2$), and

$$\mathbf{b} = \begin{bmatrix} \cos\theta\cos\varphi \\ \sin\theta\cos\varphi \\ \sin\varphi \end{bmatrix},$$

for some angles θ and φ in three dimensions ($d = 3$). We let $\epsilon = 0.001$.

The polyhedral domains Ω shown in Fig. 1, is covered by an unstructured tetrahedral mesh and the above PDE is discretized using continuous piecewise linear elements. From the computed stiffness matrix $\widehat{A}$, using symmetric Gauss-Seidel iterations applied to $\widehat{A}\mathbf{x} = 0$, and starting with a random initial iterate, we generate the corresponding *algebraically smooth* vector $\mathbf{w}$. Then, we apply our modularity based coarsening algorithm to generate aggregates of mesh vertices.

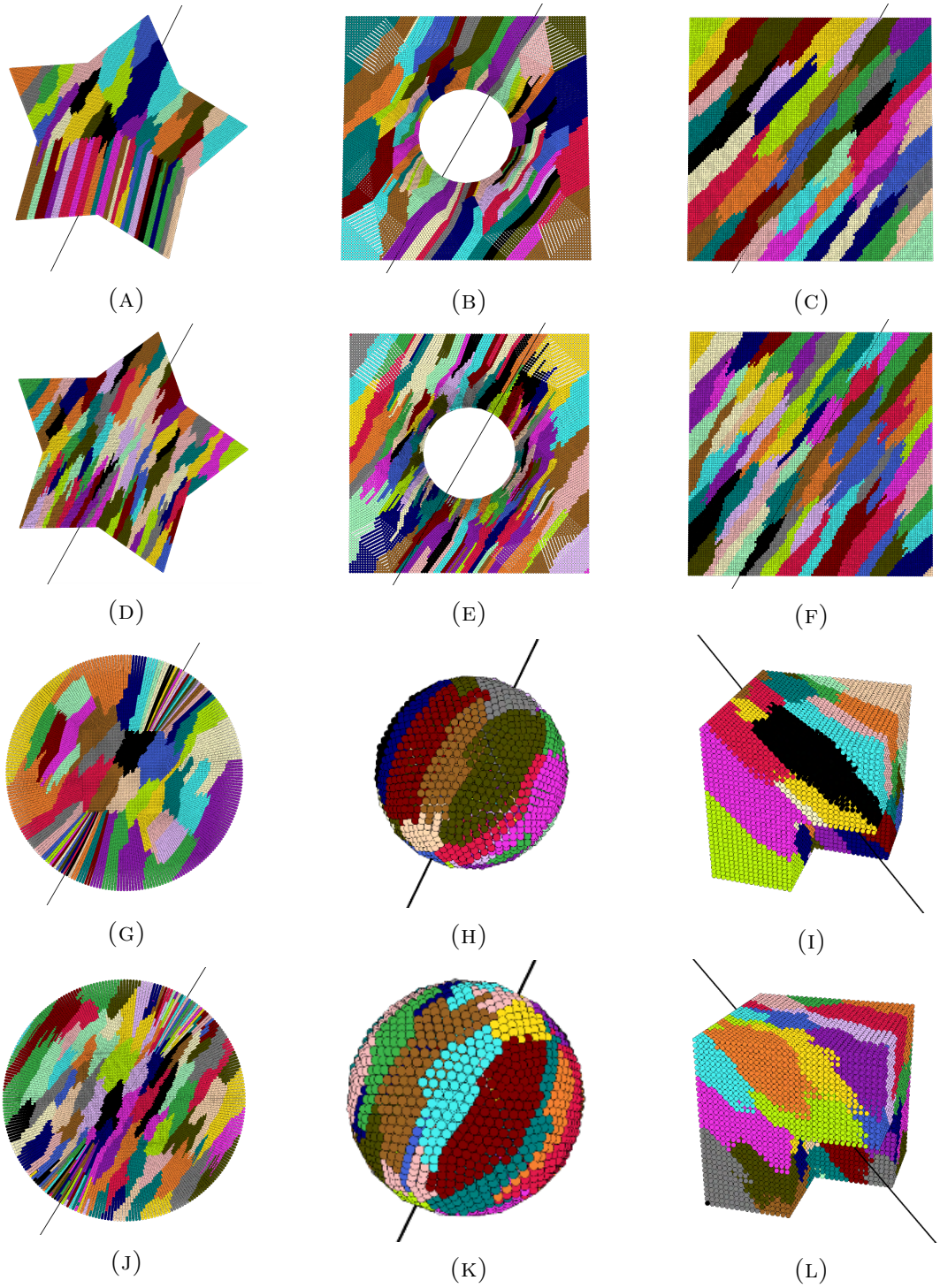

FIGURE 1. Communities found on the `star`, `square-disc`, `inline-tri`, `disc-nurbs`, and `fischera` meshes from MFEM [15] along with a sphere mesh, discretized using MFEM with no boundary conditions. The top versions of the coarsening used weights determined by taking $\mathbf{w} = (1)$ while the bottom versions had weights determined by the computed $\mathbf{w}$.

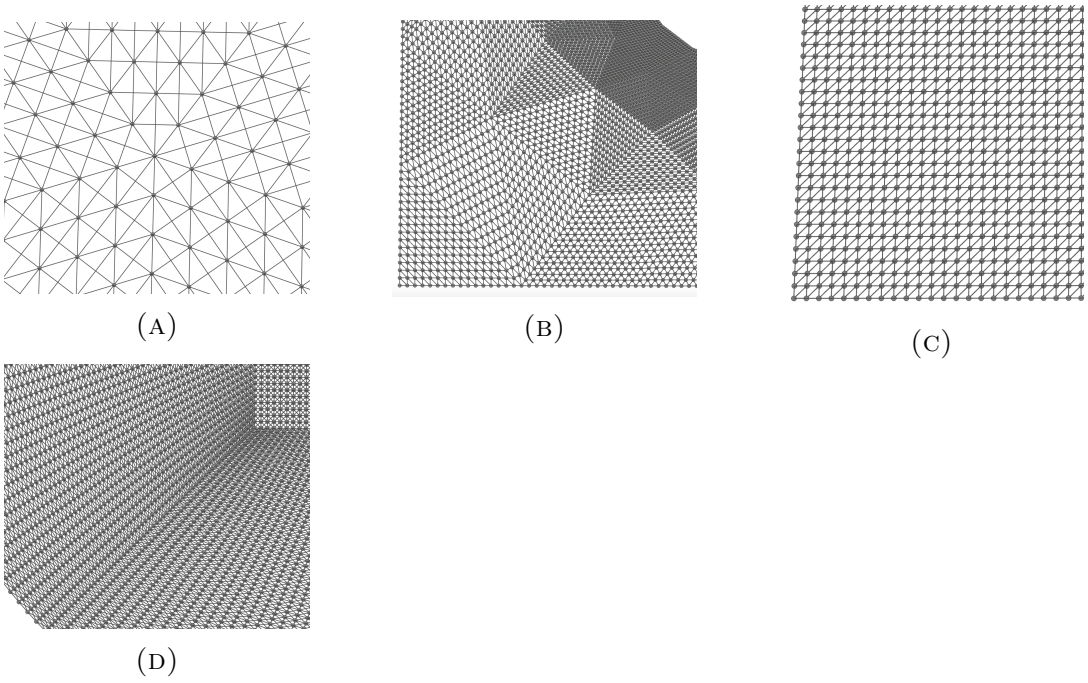

FIGURE 2. Portions of the two dimensional meshes used in Figure 1.

Figure 1 shows the results of the proposed clustering method applied to different meshes for different angles, where $\mathbf{w}$ is as above (giving $a_{ij} = -w_i\widehat{a}_{ij}w_j$ for $i \neq j$ and $a_{ii} = 0$), as well as when $\mathbf{w} = (1)$ (giving $a_{ij} = -\widehat{a}_{ij}$ for $i \neq j$ and $a_{ii} = 0$. The gray line in the subfigures indicates the direction of anisotropy. Figure 2 has pictures of important subsections each of the two dimensional meshes for reference. Meshes 2(a), 2(c), 2(b), 2(d) correspond to Figures 1(a) and 1(d), Figures 1(b) and 1(e), Figures 1(c) and 1(f), and Figures 1(g) and 1(j), respectively.

We found that using $\mathbf{w} = (1)$ can allow the proposed clustering method to aggregate in the direction of anisotropy, as shown in Figures 1(a), 1(b), 1(c), 1(g), 1(h), and 1(i). However, using $\mathbf{w}$ corresponding to an approximation to $\widehat{A}\mathbf{x} = 0$ (as explained above) allows us to coarsen in the direction of anisotropy even more, as depicted in Figures 1(d), 1(e), 1(f), 1(j), 1(k), and 1(l).

Taking $\mathbf{w} = (1)$ can cause coarsening which is highly dependent on the structure of the mesh, while the proposed weights are not dependent. For example, in Figure 1(a) the community structure follows the mesh structure nearly exactly on the lower half. This happens similarly in Figures 1(c) and 1(g) but to a lesser extent. Using the other weight choice on these meshes remedies this defect. The correct $\mathbf{w}$ weights also appear to form smaller aggregates, which end up aligning more closely with the direction of anisotropy.

Figures 1(h), 1(i), 1(k), and 1(l) are three dimensional meshes which we can similarly detect anisotropy in. This detection occurs throughout the mesh, not just on the boundary.

5. Concluding remarks

In this paper, we demonstrated that a parallel pairwise aggregation algorithm for community detection optimizes modularity as well as the Louvain method on certain classes of graphs and is also faster than the Louvain method on these classes. We also found that the social networks class of graphs do not allow the proposed method, even with heuristics, to work well due to the high-degree nodes. Finally, we applied the aggregation algorithm to the problem of finding communities aligning with the dominant direction of anisotropy in finite element discretizations and found that the method indeed detects the direction of anisotropy in both two and three dimensions on various meshes.

References

[1] V. D. Blondel, J.-L. Guillaume , R. Lambiotte, E. Lefebvre, "*Fast unfolding of communities in large networks,*" Journal of Statistical Mechanics: Theory and Experiment. 2008 (10): P10008, doi:10.1088/1742-5468/2008/10/P10008.

[2] A. Brandt, J. Brannick, K. Kahl, and I. Livshitz, "Bootstrap AMG", *SIAM J. Sci. Comput.* **33** (2011), pp. 612–632.

[3] M. Brezina, R.D. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, "Adaptive smoothed aggregation αSA multigrid", *SIAM Rev.* **47** (2005), pp. 317–346.

[4] M. Brezina, R.D. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, "Adaptive algebraic multigrid". *SIAM J. Sci. Comput.* **27** (2006), pp. 1261–1286.

[5] M.E.J. Newman, "*Networks. An Introduction*", Oxford University Press, New York, 2010.

[6] M. T. Jones and P. E. Plassmann, "*A Parallel Graph Coloring Heuristic,*" SIAM Journal on Scientific Computing, **14**(3), (1993), pp. 654669.

[7] P. D'Ambra, S. Filippone, and P. S. Vassilevski, "BootCMatch: A Software Package for Bootstrap AMG Based on Graph Weighted Matching, ACM Transactions on Mathematical Software (TOMS) 44(4) (2018) Article No. 39.

[8] P. D'Ambra and P. S. Vassilevski, "*Adaptive AMG with coarsening based on compatible weighted matching*", Computing and Visualization in Science, **16**, (2013), pp. 59–76.

[9] P.S. Vassilevski, *Multilevel block factorization preconditioners, matrix-based analysis and algorithms for solving finite element equations*, Springer, New York, 2008.

[10] J. Leskovec and A. Krevl, SNAP Datasets: Stanford Large Network Dataset Collection, 2014. snap.stanford.edu/data.

[11] R. Rossi and N. Ahmed, The Network Data Repository with Interactive Graph Analytics and Visualization, 2015. networkrepository.com

[12] V. D. Blondel, J.-L. Guillaume , R. Lambiotte, E. Lefebvre, Implementation of the Louvain method. sites.google.com/site/findcommunities/.

[13] S. Ghosh, et al., "*Distributed Louvain Algorithm for Community Detection*".

[14] J. Zeng and H. Yu, "*A Scalable Distributed Louvain Algorithm for Large-scale Graph Community Detection*"

[15] MFEM: Modular Finite Element Methods Library. mfem.org.

Acknowledgment

This work is performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This work of the second author was partially supported by NSF under Grant DMS-1619640. The LLNL IM release number is LLNL-TR-779424.

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551, U.S.A.

E-mail address: bq2367@gmail.com, quiring1@llnl.gov

Department of Mathematics and Statistics, Portland State University, Portland, Oregon, USA, and, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA, USA.

E-mail address: panayot@pdx.edu, vassilevski1@llnl.gov