

GENERALIZING LLOYD'S ALGORITHM FOR GRAPH CLUSTERING*

TAREQ ZAMAN[†], NICOLAS NYTKO[‡], ALI TAGHIBAKHSI[§], SCOTT MACLACHLAN[†], LUKE OLSON[‡], AND MATTHEW WEST[§]

Abstract. Clustering is a commonplace problem in many areas of data science, with applications in biology and bioinformatics, understanding chemical structure, image segmentation, building recommender systems, and many more fields. While there are many different clustering variants (based on given distance or graph structure, probability distributions, or data density), we consider here the problem of clustering nodes in a graph, motivated by the problem of aggregating discrete degrees of freedom in multigrid and domain decomposition methods for solving sparse linear systems. Specifically, we consider the challenge of forming *balanced* clusters in the graph of a sparse matrix for use in algebraic multigrid, although the algorithm has general applicability. Based on an extension of the Bellman-Ford algorithm, we generalize Lloyd's algorithm for partitioning subsets of \mathbb{R}^n to balance the number of nodes in each cluster; this is accompanied by a rebalancing algorithm that reduces the overall *energy* in the system. The algorithm provides control over the number of clusters and leads to “well centered” partitions of the graph. Theoretical results are provided to establish linear complexity and numerical results in the context of algebraic multigrid highlight the benefits of improved clustering.

Key words. clustering, aggregation, multigrid, graph partitioning

AMS subject classifications. 65F50, 65N55, 68R10

1. Introduction. Consider a directed graph $G(V, E, W)$ where V is a set of nodes (or vertices), $V = \{1, \dots, N_{\text{node}}\}$, and where E is a list of edges given by $E = \{(i, j) \mid W_{i,j} \neq 0\}$ for some weight matrix W . The (sparse) weight matrix W is assumed to have non-negative off-diagonal entries and zero diagonal entries. The goal of this work is to define a set of clusters that minimizes a given energy functional with linear complexity in the number of nodes. In subsection 3.1, the maximum cluster diameter is used to define the energy.

There are an array of challenges in clustering; the focus here is twofold: (1) developing efficient algorithms where the number of clusters N_{cluster} can be specified; and (2) generating clusterings that are considered “well balanced”. As a motivating example, we consider a graph generated from a finite-element discretization on a unit disk with 528 vertices¹. Figure 1 illustrates the clustering of nodes using four different methods that underscore these two challenges. Nearest-neighbor (or *Greedy*) clustering [26] yields 63 clusters in this case. While this simple algorithm lacks control of the number of clusters, the clustering offers a clear balance in the number of nodes per cluster and total diameter of each cluster. In contrast, with $N_{\text{cluster}} = 52$ the spectral-based partitioner METIS [10] yields long clusters (and large diameters).

In this work, we focus on shortest-path based clustering algorithms. *Lloyd* clus-

*Submitted to the editors TODO.

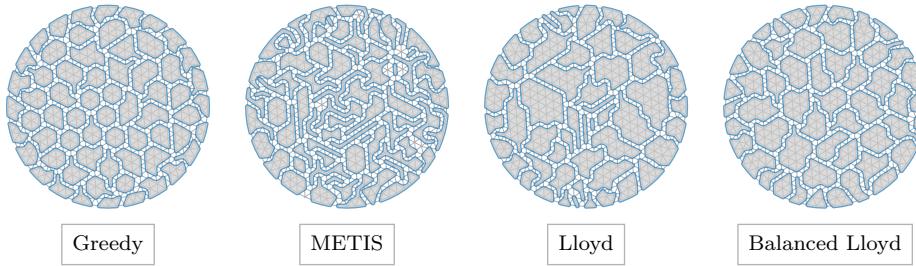
Funding: The work of S.P.M. was partially supported by an NSERC Discovery Grant. This material is based in part upon work supported by the Department of Energy, National Nuclear Security Administration, under Award Number DE-NA0003963.

[†]Interdisciplinary Program in Scientific Computing, Memorial University of Newfoundland, St. John’s, NL, Canada (tzaman@mun.ca, smaclachlan@mun.ca).

[‡]Department of Computer Science, University of Illinois Urbana-Champaign, Urbana, IL 61801, USA (nnytko2@illinois.edu, lukeo@illinois.edu).

[§]Department of Mechanical Science and Engineering, University of Illinois Urbana-Champaign, Urbana, IL 61801, USA (alit2@illinois.edu, mwest@illinois.edu).

¹This example is studied in detail in section 4

FIG. 1. *Example clusterings.*

tering (also known as Lloyd aggregation) [4], for example, uses Bellman-Ford [8] to construct $N_{\text{cluster}} = 52$ clusters based on an initial seeding. Overall clustering quality depends *highly* on the initial seeding; often even costly $\mathcal{O}(N_{\text{node}}^3)$ algorithms for seeding, such as *k-means++* [1], do not dramatically improve the final clustering in this case. Lastly, we highlight clustering based on an algorithm introduced here: a balanced form of Lloyd clustering with *rebalancing* to minimize diameter. While standard Lloyd clustering results in both large and small clusters, the cluster shapes with *balanced* Lloyd clustering and *rebalancing* are more consistent.

While there is a long history of aggregation-based multigrid methods (cf. [15, 25, 26, 17]), surprisingly little attention has been paid to the influence of cluster quality on the performance of the resulting algorithm. The *greedy* clustering algorithm originally proposed in [15] has become a standard approach that is used in many codes. Some variants on this approach have been introduced for massively parallel settings; most notably, approaches based on distance-two maximal independent sets in the graph [24, 2, 11]. Both of these approaches make minimal use of the weight matrix, W , aside from using its nonzero pattern to infer binary connectivity data in the graph. In contrast, in [4], Lloyd's algorithm [12] was extended from computing Voronoi diagrams in \mathbb{R}^n to computing clusters in graphs, using the values in W to define graph distances. It is this approach that we extend here.

In this paper, we introduce a general clustering method for use in graph partitioning and algebraic multigrid that provides control of the number of clusters, yields “centered” clusters, and can be implemented with off-the-shelf codes for Bellman-Ford and Floyd-Warshall algorithms [8]. All algorithms are implemented in and are available through the open source package PyAMG [3]. In section 2, we review aggregation-based algebraic multigrid (AMG) and survey the greedy, maximal independent set, and Lloyd clustering algorithms. Section 3 introduces *balanced* Lloyd clustering and a *rebalance* algorithm, along with theoretical evidence of convergence and complexity. Finally, section 4 provides numerical evidence in support, expanding the example in Figure 1 and others.

Note: throughout the paper and embedded in the algorithms, we make use of the notation listed in Table 2.

2. Clustering in algebraic multigrid. Algebraic multigrid methods are a family of iterative methods for the solution of sparse linear systems of the form $Au = f$, where A is an $N_{\text{node}} \times N_{\text{node}}$ matrix and u and f are vectors of dimension N_{node} . Like all multigrid methods, they achieve their efficiency through the use of two complementary processes, known as relaxation and coarse-grid correction. For algebraic

multigrid methods, we typically consider a fixed relaxation scheme (such as a stationary weighted Jacobi or Gauss-Seidel iteration on the linear system) and seek to compute a coarse-grid correction process that adequately complements relaxation to lead to an efficient solution algorithm. In aggregation-based methods, the coarse-grid correction process takes the form of first computing a clustering of the fine-grid degrees of freedom (nodes in the graph of the sparse matrix, A), and then computing an interpolation operator from the clustered degrees of freedom to those on the fine grid. Rootnode-based aggregation methods additionally make use of a center that is identified for each cluster [14]. For a more thorough review of algebraic multigrid methods, see [Appendix A](#) or [7, 23].

[Figure 1](#) illustrates the wide range of clusters that can arise for a single problem. We next detail three common approaches to clustering (used in the context of AMG), before introducing a balanced method in the next section. First, however, we define a clustering or aggregation of $G(V, E, W)$, as in [Definition 2.1](#). We note that the clustering is a non-overlapping covering.

DEFINITION 2.1. *A clustering or aggregation of the connected graph $G(V, E, W)$ is a pair (m, c) , where m_i is the cluster membership of vertex i and c_a is the global index of the center for cluster a . Then m and c have the following properties:*

1. *For each $i \in \{1, \dots, N_{\text{node}}\}$, there exists a unique a with $1 \leq a \leq N_{\text{cluster}}$ such that $m_i = a$;*
2. *For each $a \in \{1, \dots, N_{\text{cluster}}\}$, for every (i, j) with $m_i = m_j = a$, there exists a sequence k_1, \dots, k_p where $m_k = a$ for $k \in \{k_1, \dots, k_p\}$ and with $(i, k_1), (k_q, k_{q+1}), (k_p, j) \in E$ for $q \in \{1, \dots, p-1\}$; and*
3. *For each $a \in \{1, \dots, N_{\text{cluster}}\}$, we have $1 \leq c_a \leq N_{\text{node}}$ and $m_{c_a} = a$.*

The first point ensures that the clustering is a non-overlapping covering, the second requires that the subgraph over the cluster remains connected, and the third confirms that an element of each cluster is identified as the center for that cluster.

2.1. Greedy clustering. Greedy clustering (also known as “greedy aggregation” or “standard aggregation”) was first introduced by Míka and Vaněk [15]; we use a close variant. Greedy clustering consists of two passes over the set of nodes of the graph. In the first pass, for each node, if all neighbors in the graph remain unclustered, then the node becomes a center, forming a cluster from the node and its neighborhood. In the second pass, each unclustered node is included in a neighboring cluster, if possible. If a neighboring cluster is not found, then the unclustered node is considered a center node and the node with its unclustered neighbors form a new cluster. In the case of multiple neighboring clusters, there are several options: arbitrary selection, index, size, or magnitude of the weight can each be used to determine cluster membership. The full greedy algorithm is given in [Algorithm 2.1](#).

2.2. Maximal independent set based clustering. The greedy algorithm is inherently serial, yet there are two immediate observations. First, any two center nodes of two (distinct) clusters must be more than two edges apart. Second, if an unclustered node is more than two edges from any existing center, then the node is eligible to be a center of a new cluster. Hence, the center nodes from the greedy algorithm represent a distance-2 maximal independent set or MIS(2). This leads to the MIS(2) clustering algorithm, where an MIS(2) over the nodes is first constructed, followed by construction of the clustering using the MIS(2) center nodes. This has been shown to exhibit a high degree of parallelism [2]; see [2, Algorithm 5] for details.

Given a distance-2 maximal independent set, the clustering process is straightfor-

Algorithm 2.1 Greedy clustering. See [Table 2](#) for variable definitions.

```

1: function GREEDY-CLUSTERING( $W$ )
2:    $m_i \leftarrow 0$  for all  $i = 1, \dots, N_{\text{node}}$             $\triangleright$  initially all nodes are unclustered
3:    $a \leftarrow 0$                                           $\triangleright$  first cluster index
4:   for  $i \leftarrow 1, \dots, N_{\text{node}}$  do                   $\triangleright$  first pass
5:     if  $m_i = 0$  and  $m_j = 0$  for all  $j$  s.t.  $W_{i,j} \neq 0$  then       $\triangleright$  unclustered
6:        $m_i \leftarrow a$                                       $\triangleright$  add  $i$  and neighbors to cluster  $a$ 
7:        $m_j \leftarrow a$ , for all  $j$  s.t.  $W_{i,j} \neq 0$ 
8:        $c_a \leftarrow i$                                       $\triangleright$  mark cluster center
9:        $a \leftarrow a + 1$                                  $\triangleright$  increment cluster index
10:      for  $i \leftarrow 1, \dots, N_{\text{node}}$  do                  $\triangleright$  second pass
11:        if  $m_i = 0$  then                             $\triangleright$  unclustered
12:          if  $\exists j$  s.t.  $W_{i,j} \neq 0$  and  $m_j > 0$  then       $\triangleright$  clustered neighbor
13:             $j \leftarrow \underset{j : m_j > 0}{\text{argmax}} W_{i,j}$        $\triangleright$  neighbor with largest weight
14:             $m_i \leftarrow m_j$ 
15:          else                                          $\triangleright$  form new cluster
16:             $m_i \leftarrow a$ 
17:            for  $j$  such that  $W_{i,j} \neq 0$  and  $m_j = 0$  do
18:               $m_j \leftarrow a$ 
19:             $a \leftarrow a + 1$                                  $\triangleright$  increment cluster index
20:      return  $m, c$ 

```

ward. In the first step, the index of the cluster representing the center is propagated to its neighbors. This continues in the second step, where the index of the cluster is propagated to the second layer of neighbors; if there are multiple clusters adjacent to an unclustered node, the choice is made arbitrarily (or by index). The algorithm is shown in [Algorithm 2.2](#).

Algorithm 2.2 MIS(2) clustering. See [Table 2](#) for variable definitions.

```

1: function MIS(2)-CLUSTERING( $W$ )
2:    $c \leftarrow \text{MIS}(W, 2)$                                 $\triangleright$  distance-2 independent set
3:    $m_i \leftarrow 0$  for  $i = 1, \dots, N_{\text{node}}$ 
4:    $N_{\text{cluster}} \leftarrow |c|$ 
5:   for  $a = 1, \dots, N_{\text{cluster}}$  do                 $\triangleright$  pass 1: distance-1
6:      $i \leftarrow c_a$                                       $\triangleright$  index of center for cluster  $a$ 
7:      $m_i \leftarrow a$                                      $\triangleright$  set cluster number for center
8:     for  $j$  s.t.  $W_{i,j} \neq 0$  do
9:        $m_j \leftarrow a$                                  $\triangleright$  set cluster number for neighbors
10:      for  $i$  s.t.  $m_i > 0$  do                       $\triangleright$  pass 2: distance-2
11:        for  $j$  s.t.  $W_{i,j} \neq 0$  and  $m_j = 0$  do
12:           $m_j \leftarrow m_i$                             $\triangleright$  set cluster number for neighbors
13:      return  $m, c$ 

```

With an appropriate ordering, the first pass of MIS-based and greedy clustering can yield identical clusters. With only minor differences in the second pass, we expect the clustering patterns to be similar. Indeed, the convergence factors of AMG based on these two clustering strategies are shown to be close in practice [[2](#), Appendix].

2.3. Standard Lloyd clustering. A shortcoming of the previous two clustering strategies is the inability to control the coarsening rate: the number of clusters is an outcome of the algorithm, rather than an input. In contrast, Lloyd clustering, introduced in [4], is based on an initial seeding of centers (of any length). Lloyd clustering can be viewed as an extension of Lloyd's algorithm [12] applied to graphs, where an initial random seeding of centers yields Voronoi cells (or a set of nodes closest to each center), followed by a recentering of center locations.

A full algorithm is given in [Algorithm 2.3](#), where a subset of N_{cluster} nodes are randomly selected as the initial centers, input as c . A standard Bellman-Ford algorithm (see [Algorithm 2.4](#) and [8, Section 8.7]) is used to find the distance and index of the closest center; the set of points closest to each center form the initial clustering. Next, the border nodes of each cluster are selected and a modified form of the Bellman-Ford algorithm then identifies the (new) center — see [Algorithm 2.5](#) — by selecting the node of maximum distance to the cluster boundary (with ties selected arbitrarily). The steps are repeated until the algorithm has converged or a maximum number of iterations (given as T_{\max}) is reached.

Algorithm 2.3 Lloyd clustering algorithm. See [Table 2](#) for variable definitions.

```

1: function LLOYD-CLUSTERING( $W, c, T_{\max}$ )
2:    $t = 0$ 
3:   repeat
4:      $m, d \leftarrow \text{BELLMAN-FORD}(W, c)$                                  $\triangleright$  find closest centers
5:      $c \leftarrow \text{MOST-INTERIOR-NODES}(W, m)$                              $\triangleright$  recenter
6:      $t = t + 1$ 
7:   until  $t = T_{\max}$  or no change in  $c$  and  $m$ 
8:   return  $m, c$ 
```

Algorithm 2.4 Bellman-Ford algorithm to compute distance and index of closest center. See [Table 2](#) for variable definitions.

```

1: function BELLMAN-FORD( $W, c$ )
2:    $d_i \leftarrow \infty$  for all  $i = 1, \dots, N_{\text{node}}$                                  $\triangleright$  initial distance
3:    $m_i \leftarrow 0$  for all  $i = 1, \dots, N_{\text{node}}$                                  $\triangleright$  initial membership undefined
4:   for  $a \leftarrow 1, \dots, N_{\text{cluster}}$  do
5:      $i \leftarrow c_a$                                                $\triangleright$  cluster  $a$  has center node  $i$ 
6:      $d_i \leftarrow 0$                                                $\triangleright$  distance of a center node to itself is zero
7:      $m_i \leftarrow a$                                                $\triangleright$  center node  $i$  belongs to its own cluster
8:   repeat
9:      $\text{done} \leftarrow \text{true}$ 
10:    for  $i, j$  such that  $W_{i,j} > 0$  do                                 $\triangleright$  all pairs of adjacent nodes
11:      if  $d_i + W_{i,j} < d_j$  then                                 $\triangleright$  node  $j$  is closer to  $i$ 's center
12:         $m_j \leftarrow m_i$                                  $\triangleright$  switch node  $j$  to the same cluster as  $i$ 
13:         $d_j \leftarrow d_i + W_{i,j}$                                  $\triangleright$  use the shorter distance via node  $i$ 
14:         $\text{done} \leftarrow \text{false}$                                  $\triangleright$  change was made; do not terminate
15:    until  $\text{done}$ 
16:    return  $m, d$ 
```

2.3.1. Theoretical observations. A significant advantage of standard Lloyd clustering, as in [Algorithm 2.3](#), is the dependence on *off-the-shelf* algorithms such as

Algorithm 2.5 Find the most-interior node (furthest from boundary) for each cluster. See [Table 2](#) for variable definitions.

```

1: function MOST-INTERIOR-NODES( $W, m$ )
2:    $B \leftarrow \{\}$                                  $\triangleright$  border nodes
3:   for  $i, j$  such that  $W_{i,j} > 0$  do           $\triangleright$  all pairs of adjacent nodes
4:     if  $m_i \neq m_j$  then                   $\triangleright$  are nodes  $i$  and  $j$  in different clusters?
5:        $B \leftarrow B \cup \{i, j\}$              $\triangleright$  if so, add both of them to the border set
6:    $\cdot, d \leftarrow \text{BELLMAN-FORD}(W, B)$        $\triangleright d$  is distance from cluster borders
7:   for  $i \leftarrow 1, \dots, N_{\text{node}}$  do
8:      $a \leftarrow m_i$                            $\triangleright a$  is the cluster index for node  $i$ 
9:      $c_a \leftarrow i$                           $\triangleright$  assign the highest-index node as cluster center
10:    for  $i \leftarrow 1, \dots, N_{\text{node}}$  do
11:       $a \leftarrow m_i$                          $\triangleright a$  is the cluster index for node  $i$ 
12:       $j \leftarrow c_a$                        $\triangleright j$  is the current cluster center
13:      if  $d_i > d_j$  then                 $\triangleright$  is node  $i$  further from the border than  $j$ ?
14:         $c_a \leftarrow i$                      $\triangleright$  if so, node  $i$  is the new cluster center
15:   return  $c$ 

```

Bellman-Ford. This allows us to establish key properties that will carry over to more advanced algorithms in the next section.

To begin, we note that standard Bellman-Ford terminates (in [Theorem 2.2](#)), an important property to maintain as we seek more balanced clusters.

THEOREM 2.2. *Algorithm 2.4 terminates.*

Proof. This is a standard result [8, Section 8.7]. \square

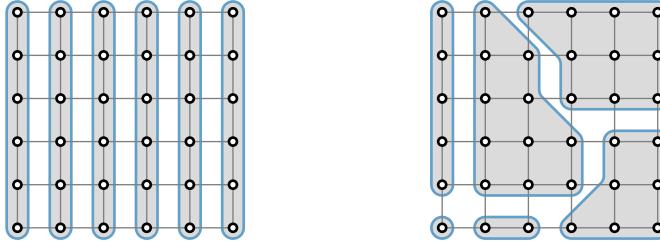
Likewise, while we assume the initial graph is connected, [Definition 2.1](#) requires each of the clusters to be connected. Bellman-Ford provides this, as summarized in [Theorem 2.3](#).

THEOREM 2.3. *The clusters returned by Algorithm 2.4 are connected.*

Proof. This follows from the proof of [Theorem 3.1](#), using only the first case in the proof corresponding to Line 9 in [Algorithm 3.4](#). \square

3. Balanced Lloyd clustering. Lloyd clustering in [subsection 2.3](#) enables the construction of a *variable* number of clusters, based on the initial seeding. Yet, the method can result in poor *quality* clusters (cf. [Figure 1](#)). As an example, consider a nearest-neighbor weight matrix W based on distance and on a 6×6 structured mesh. [Figure 2](#) illustrates two common scenarios in standard Lloyd clustering. The first is the emergence of long, narrow clusters. This is, in part, due to the method of finding boundaries in [MOST-INTERIOR-NODES](#); in this case, the entire cluster (left figure) is comprised of boundary nodes, leaving no opportunity to re-center. The second artifact of standard Lloyd is that of disparate cluster sizes. Here, we observe both large clusters and clusters of a single point (right figure). An immediate goal in the algorithms of this section is to address these two points.

The Bellman-Ford algorithm is the central component of standard Lloyd, finding the shortest path for each seed and to find distal points for each boundary node in a cluster. In the following, we introduce a new method for centering nodes (see [Algo-](#)

FIG. 2. Two example clusterings from Lloyd clustering on a 6×6 mesh.

rithm 3.5), where we seek to minimize the total energy defined by

$$(3.1) \quad H = \sum_{i=1}^{N_{\text{node}}} d_i^2,$$

where d_i is defined to be the distance from node i to the center of the cluster for node i , namely c_a where cluster a satisfies $a = m_i$. This requires the computation of the shortest path for each pair of nodes in the cluster, a.k.a. the all-pairs shortest path problem. For this we turn to a per-cluster use of Floyd-Warshall [8] as detailed in Algorithm 3.1.

Algorithm 3.1 Floyd-Warshall algorithm [8, Section 9.8] to find inter-node distances within each cluster. See Table 2 for variable definitions.

```

1: function CLUSTERED-FLOYD-WARSHALL( $W, m$ )
2:    $V_a \leftarrow \{i \mid m_i = a\}$  for all  $a = 1, \dots, N_{\text{cluster}}$             $\triangleright$  nodes in cluster  $a$ 
3:   for  $a \leftarrow 1, \dots, N_{\text{cluster}}$  do
4:     for  $i, j \in V_a$  do
5:        $D_{i,j} \leftarrow \infty$                                  $\triangleright$  initial distance  $i \rightarrow j$ 
6:        $P_{i,j} \leftarrow 0$                                  $\triangleright$  initial predecessor node for  $i \rightarrow j$ 
7:       if  $W_{i,j} > 0$  then
8:          $D_{i,j} \leftarrow W_{i,j}$            $\triangleright$  adjacent nodes have the adjacency distance
9:          $P_{i,j} \leftarrow i$              $\triangleright$  the predecessor is the tail node for adjacent pairs
10:        if  $i = j$  then
11:           $D_{i,i} \leftarrow 0$            $\triangleright$  nodes are distance zero from themselves
12:           $P_{i,i} \leftarrow i$            $\triangleright$  nodes are their own predecessors to themselves
13:        for  $k \in V_a$  do           $\triangleright$  potential intermediate node on the path  $i \rightarrow j$ 
14:          for  $i, j \in V_a$  do           $\triangleright$  all other node pairs within the cluster
15:            if  $D_{i,k} + D_{k,j} < D_{i,j}$  then       $\triangleright i \rightarrow k \rightarrow j$  shorter than  $i \rightarrow j$ 
16:               $D_{i,j} \leftarrow D_{i,k} + D_{k,j}$            $\triangleright$  switch to the shorter distance
17:               $P_{i,j} \leftarrow P_{k,j}$                  $\triangleright$  take the predecessor from  $k \rightarrow j$ 
18:    return  $D, P$ 

```

For cluster a , we note that the calculation of shortest paths in Algorithm 3.1 is $\mathcal{O}(s_a^3)$, where $s_a = |\{i \mid m_i = a\}|$ is the size of cluster a . In the following section, we establish linear complexity in the number of nodes, N_{node} , with assumptions on the maximum cluster size.

The introduction of energy as a target (see (3.1)) provides an opportunity to re-balance the clustering to account for small or large clusters. For this, we introduce a

rebalancing algorithm that calculates the energy increase in splitting clusters and the energy decrease in eliminating clusters. The overall process relies on the distances from Floyd-Warshall. In subsection 3.1, a balanced version of Bellman-Ford is introduced, leading to a balanced form of Lloyd clustering. The rebalancing algorithm is constructed in subsection 3.2 followed by theoretical observations. The rebalanced Lloyd algorithm requires several components and we summarize the dependence in Figure 3.

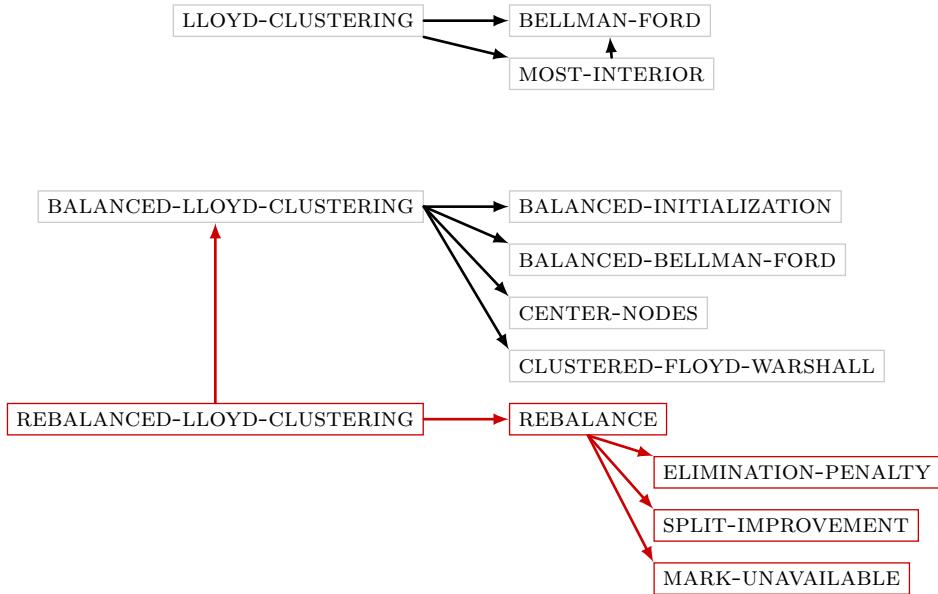


FIG. 3. Algorithm dependence. Rebalancing components are highlighted in red.

3.1. Balanced algorithms. One disadvantage of Lloyd clustering is that the clusters are not guaranteed (nor expected) to be uniformly sized. In many practical settings, a node is likely to have nearly the same distance to multiple centers. In this case, Lloyd clustering randomly assigns the node to a cluster; in contrast, balanced Lloyd clustering targets uniformly sized clusters, as described in Algorithm 3.2. In the balanced approach, if a node has the same distance to different centers, the node is assigned to a smaller cluster, leading to increased uniformity across clusters. Specifically, Floyd-Warshall (Algorithm 3.1) replaces balanced Bellman-Ford (see Algorithm 3.4) to compute the centroid of each cluster. A node of a cluster having the minimum sum of squared distance to other cluster nodes is taken as centroid of that region (Algorithm 3.5). Consequently, long, narrow clusters as in Figure 2 will expose centers near the true center, whereas the boundary-distances used in standard Lloyd clustering leave any boundary centers unchanged.

Note that on Line 11 of Algorithm 3.4, the condition $d_i + W_{i,j} = d_j$ should be implemented using an approximate comparison if floating point arithmetic is being used.

3.2. Rebalancing clustering. Balanced Lloyd clustering improves the uniformity and roundness of the clusters in the Lloyd clustering. Yet there is no guarantee that the energy is minimized *across* clusters, due to the initial seeding. In this section, we develop a rebalancing algorithm that reduces the overall energy

Algorithm 3.2 Balanced version of Lloyd clustering. See [Table 2](#) for variable definitions.

```

1: function BALANCED-LLOYD-CLUSTERING( $W, c, T_{\max}, T_{BF\max}$ )
2:    $m, d, p, n, s \leftarrow$  BALANCED-INITIALIZATION( $c, N_{\text{node}}$ )
3:    $t = 0$ 
4:   repeat
5:      $m, d, p, n, s \leftarrow$  BALANCED-BELLMAN-FORD( $W, m, c, d, p, n, s, T_{BF\max}$ )
6:      $D, P \leftarrow$  CLUSTERED-FLOYD-WARSHALL( $W, m$ )
7:      $c, d, p, n \leftarrow$  CENTER-NODES( $W, m, c, d, p, n, D, P$ )
8:      $t = t + 1$ 
9:   until  $t = T_{\max}$  or no change in any of  $m, c, d, p, n, s$ 
10:  return  $m, c, d, p, n, s, D, P$ 
```

Algorithm 3.3 Initialization for balanced algorithms. See [Table 2](#) for variable definitions.

```

1: function BALANCED-INITIALIZATION( $c, N_{\text{node}}$ )
2:    $m_i \leftarrow 0$  for all  $i = 1, \dots, N_{\text{node}}$             $\triangleright$  cluster membership for node  $i$ 
3:    $d_i \leftarrow \infty$  for all  $i = 1, \dots, N_{\text{node}}$         $\triangleright$  distance to node  $i$  from its cluster center
4:    $p_i \leftarrow 0$  for all  $i = 1, \dots, N_{\text{node}}$             $\triangleright$  predecessor node for node  $i$ 
5:    $n_i \leftarrow 0$  for all  $i = 1, \dots, N_{\text{node}}$             $\triangleright$  number of predecessor nodes for node  $i$ 
6:    $s_a \leftarrow 1$  for all  $a = 1, \dots, N_{\text{cluster}}$            $\triangleright$  size of cluster  $a$ 
7:   for  $a \leftarrow 1, \dots, N_{\text{cluster}}$  do
8:      $i \leftarrow c_a$                                  $\triangleright$   $i$  is the center node index for cluster  $a$ 
9:      $d_i \leftarrow 0$                                  $\triangleright$  distance of center to node  $i$  from itself is zero
10:     $m_i \leftarrow a$                                 $\triangleright$  center node  $i$  belongs to its own cluster
11:     $p_i \leftarrow i$                                  $\triangleright$  centers are their own predecessors
12:     $n_i \leftarrow 1$                                  $\triangleright$  centers have one predecessor
13:  return  $m, d, p, n, s$ 
```

in the clustering by *splitting* clusters into two and reducing energy, and by eliminating clusters leading to an increase in energy. The trade-off maintains a constant number of clusters, but reduces the total energy in the clustering. The rebalanced Lloyd algorithm is given in [Algorithm 3.6](#) and the rebalancing algorithm itself is given in [Algorithm 3.7](#).

The algorithm relies on two calculations, the first being in [Algorithm 3.8](#), which iterates through each cluster and calculates the energy penalty (increase) resulting from eliminating a cluster and merging each node with its nearest cluster. The nearest cluster of a node is defined based on the distance of the centre of the cluster from the node. Similarly, [Algorithm 3.9](#) computes the energy improvement (decrease) from optimally splitting each cluster into two clusters. Here, we determine the splitting (of each cluster) that results in the lowest energy by considering all possible pairs of new centers within the cluster.

With measures on the penalties and improvements in energy, the rebalancing algorithm proceeds by eliminating and splitting clusters in pairs, thereby reducing the total energy while keeping the number of clusters constant. At first, it eliminates the cluster with the smallest elimination penalty and splits the cluster with the largest split improvement, if these are distinct clusters. It then proceeds to eliminate the cluster with the second-smallest penalty and split the one with the second-largest

Algorithm 3.4 Balanced version of Bellman-Ford. See [Table 2](#) for variable definitions.

```

1: function BALANCED-BELLMAN-FORD( $W, m, c, d, p, n, s, T_{BFmax}$ )
2:    $t \leftarrow 0; z^{(t)} \leftarrow z$  for all variables  $z$                                  $\triangleright$  only for use in proofs
3:   repeat
4:     done  $\leftarrow$  true
5:     for  $i, j$  such that  $W_{i,j} > 0$  do                                      $\triangleright$  all pairs of adjacent nodes
6:        $s_i \leftarrow s_{m_i}$  if  $m_i > 0$ , else 0                                 $\triangleright$  size of cluster containing node  $i$ 
7:        $s_j \leftarrow s_{m_j}$  if  $m_j > 0$ , else 0                                 $\triangleright$  size of cluster containing node  $j$ 
8:       switch  $\leftarrow$  false
9:       if  $d_i + W_{i,j} < d_j$  then            $\triangleright j$  is closer to  $i$ 's center than its own
10:        switch  $\leftarrow$  true
11:       if  $d_i + W_{i,j} = d_j$  then            $\triangleright$  distance to  $j$  is similar from  $i$ 's center
12:         if  $s_i + 1 < s_j$  then            $\triangleright$  node  $i$ 's cluster is smaller (by 2 or more)
13:           if  $n_j = 0$  then            $\triangleright$  node  $j$  is free to switch (not a predecessor)
14:             switch  $\leftarrow$  true
15:       if switch then
16:          $s_{m_i} \leftarrow s_i + 1, s_{m_j} \leftarrow s_j - 1$            $\triangleright$  update cluster sizes
17:          $m_j \leftarrow m_i$                                  $\triangleright$  switch node  $j$  to the same cluster as  $i$ 
18:          $d_j \leftarrow d_i + W_{i,j}$                            $\triangleright$  use the distance via node  $i$ 
19:          $n_i \leftarrow n_i + 1, n_{p_j} \leftarrow n_{p_j} - 1$        $\triangleright$  update predecessor counts
20:          $p_j \leftarrow i$                                  $\triangleright$  predecessor of node  $j$  is now  $i$ 
21:         done  $\leftarrow$  false                             $\triangleright$  change was made; do not terminate
22:        $t \leftarrow t + 1; z^{(t)} \leftarrow z$  for all variables  $z$            $\triangleright$  only for use in proofs
23:   until  $t = T_{BFmax}$  or done
24:    $T \leftarrow t$                                                $\triangleright$  only for use in proofs
25:   return  $m, d, p, n, s$ 

```

improvement, again assuming they are distinct. This process continues until the energy will no longer be decreased (i.e., the next elimination penalty would be greater than or equal to the next split improvement), at which point rebalancing terminates. To access the clusters in sorted order we use an `ARGSORT(L)` function that returns the array of indexes $[i_1, i_2, \dots]$ so that L_{i_1}, L_{i_2}, \dots will be in sorted order.

During rebalancing, we assume that the elimination penalties and split improvements of the clusters do not change as we are actually eliminating and splitting other clusters. However, the penalty and improvement of cluster a depend on its neighboring clusters. For this reason, when we eliminate or split a cluster, we mark all of its neighbors as unavailable for being eliminated or split themselves. This ensures that the penalties and improvement values remain correct for all clusters that are under consideration for elimination or splitting at each step.

3.3. Theoretical observations. We have formulated the balanced Bellman-Ford algorithm with a cap on the maximum number of iterations, which will be necessary for proving linear complexity in [Theorem 3.6](#). In practice T_{BFmax} can be chosen to be the maximum expected cluster radius and implementations should warn if [Algorithm 3.4](#) reaches this limit, as this may indicate that the clusters are not connected. This observation relies on the following result.

THEOREM 3.1. *The clusters returned by [Algorithm 3.4](#) are connected if it termi-*

Algorithm 3.5 Update center nodes to be the cluster centroids. See Table 2 for variable definitions.

```

1: function CENTER-NODES( $W, m, c, d, p, n, D, P$ )
2:    $V_a \leftarrow \{i \mid m_i = a\}$  for all  $a = 1, \dots, N_{\text{cluster}}$             $\triangleright$  nodes in cluster  $a$ 
3:   for  $a = 1, \dots, N_{\text{cluster}}$  do                                 $\triangleright$  treat each cluster  $a$  separately
4:     for  $i \in V_a$  do
5:        $q_i \leftarrow \sum_{j \in V_a} (D_{i,j})^2$   $\triangleright$  sum of squared distances to other cluster nodes
6:        $i \leftarrow c_a$                                  $\triangleright$  current cluster center
7:       for  $j \in V_a$  do                       $\triangleright$  test node  $j$  as a new cluster center
8:         if  $q_j < q_i$  then           $\triangleright$  does node  $j$  have a strictly better metric?
9:            $i \leftarrow j$                    $\triangleright j$  will be the new cluster center
10:      if  $i \neq c_a$  then           $\triangleright$  have we found a new center?
11:         $c_a \leftarrow i$ 
12:         $n_j \leftarrow 0$  for all  $j \in V_a$            $\triangleright$  reset predecessor counts
13:        for  $j \in V_a$  do           $\triangleright$  update data for all nodes in the cluster
14:           $d_j \leftarrow D_{i,j}$ 
15:           $p_j \leftarrow P_{i,j}$ 
16:           $n_{p_j} \leftarrow n_{p_j} + 1$ 
17:    return  $c, d, p, n$ 
```

Algorithm 3.6 Rebalanced version of Lloyd clustering. See Table 2 for variable definitions.

```

1: function REBALANCED-LLOYD-CLUSTERING( $W, c, T_{\max}, T_{\text{BFmax}}$ )
2:    $t = 0$ 
3:   repeat
4:      $m, c, d, p, n, s, D, P \leftarrow \text{BALANCED-LLOYD-CLUSTERING}(W, c, T_{\max}, T_{\text{BFmax}})$ 
5:      $c \leftarrow \text{REBALANCE}(W, m, c, d, p, D)$ 
6:      $t = t + 1$ 
7:   until  $t = T_{\max}$  or no change in  $c$ 
8:   return  $m, c, d, p, n, s, D, P$ 
```

ates before the maximum number of iterations.

Proof. The proof relies on understanding the state of the variables within Algorithm 3.4 as the algorithm iterates. We denote this by using a superscript, $z^{(t)}$ to indicate the state of variable z at a given “time”, t , in the algorithm, with time T denoting completion.

We wish to show that, for each j , $m_j^{(T)} = m_i^{(T)}$ where $i = p_j^{(T)}$, which ensures that the predecessor-paths to cluster centers are contained within each cluster. Suppose not and let t be the last iteration when m_j and p_j were updated by Lines 17 and 20. Taking $i = p_j^{(t)}$ we have $m_i^{(t)} = m_j^{(t)} = m_j^{(T)} \neq m_i^{(T)}$ so there must be a later $t' > t$ at which m_i was updated for the last time. At this later time, we must have at least one of the following cases.

Case 1: condition on Line 9 is true. Then $d_i^{(t')} < d_i^{(t)}$, and so $d_i^{(t')} + W_{i,j} < d_i^{(t)} + W_{i,j} = d_j^{(t)} = d_j^{(T)}$. This is a contradiction because we cannot have $d_j > d_i + W_{i,j}$ when the algorithm terminates.

Case 2: conditions on Lines 11 and 13 are true. Then $n_i^{(t')} = 0$, which is impos-

Algorithm 3.7 Rebalance clusters by eliminating low-energy clusters and splitting the same number of high-energy clusters in two. See Table 2 for variable definitions.

```

1: function REBALANCE( $W, m, c, d, p, D$ )
2:    $V_a \leftarrow \{i \mid m_i = a\}$  for all  $a = 1, \dots, N_{\text{cluster}}$             $\triangleright$  nodes in cluster  $a$ 
3:    $L \leftarrow \text{ELIMINATION-PENALTY}(W, m, d, D)$ 
4:    $(S, c^1, c^2) \leftarrow \text{SPLIT-IMPROVEMENT}(m, d, D)$ 
5:    $M_a \leftarrow \text{TRUE}$  for all  $a \leftarrow 1, \dots, N_{\text{cluster}}$             $\triangleright$  all clusters are modifiable
6:    $L^{\text{sort}} \leftarrow \text{ARGSORT}(L)$ 
7:    $S^{\text{sort}} \leftarrow \text{ARGSORT}(S)$ 
8:    $i_L \leftarrow 1$             $\triangleright$  sorted index of cluster to eliminate
9:    $i_S \leftarrow N_{\text{cluster}}$             $\triangleright$  sorted index of cluster to split
10:  while  $i_L \leq N_{\text{cluster}}$  and  $i_S \geq 1$  do
11:     $a_L \leftarrow L_{i_L}^{\text{sort}}$             $\triangleright$  cluster to eliminate
12:     $a_S \leftarrow S_{i_S}^{\text{sort}}$             $\triangleright$  cluster to split
13:    if not  $M_{a_L}$  or  $a_L = a_S$  then            $\triangleright$  is cluster  $a_L$  modifiable and distinct?
14:       $i_L \leftarrow i_L + 1$ 
15:      continue
16:    if not  $M_{a_S}$  then            $\triangleright$  is cluster  $a_S$  modifiable?
17:       $i_S \leftarrow i_S - 1$ 
18:      continue
19:    if  $L_{a_L} \geq S_{a_S}$  then            $\triangleright$  will the energy not decrease?
20:      break
21:    MARK-UNAVAILABLE( $a_L, M, W, V_{a_L}$ )
22:    MARK-UNAVAILABLE( $a_S, M, W, V_{a_S}$ )
23:     $c_{a_L} \leftarrow c_{a_S}^1$             $\triangleright$  eliminate cluster  $a_L$ 
24:     $c_{a_S} \leftarrow c_{a_S}^2$             $\triangleright$  split cluster  $a_S$ 
25:  return  $c$ 

```

sible since $p_j^{(t')} = p_j^{(t)} = i$. \square

To understand the behavior of the balanced algorithms, we consider a modified energy that includes a second term for the cluster sizes. Define

$$(3.2) \quad H_\delta = \sum_{i=1}^{N_{\text{node}}} (d_i)^2 + \delta \sum_{a=1}^{N_{\text{cluster}}} (s_a)^2,$$

where d_i is the distance from node i to its cluster center, $s_a = |\{i \mid m_i = a\}|$ is the size (number of nodes) of cluster a , and

$$(3.3) \quad \delta = \left(\frac{\Delta_{\min}}{N_{\text{node}}} \right)^2$$

is chosen based on the minimum difference, Δ_{\min} , between distinct values of $W_{i,j}$ (or an arbitrarily small positive number if there are no distinct values of $W_{i,j}$). The first term in (3.2) is the sum of squared distances from nodes to their cluster centers, while the second term is the sum of squared cluster sizes. Note that δ is chosen so that the second term in (3.2) is always less than the minimum possible increment in the first term.

LEMMA 3.2. *Algorithm 3.4 results in a decrease of the energy (3.2), or preserves the energy if no change is made to the clustering.*

Algorithm 3.8 Calculate the energy increase that would result from eliminating each cluster. See Table 2 for variable definitions.

```

1: function ELIMINATION-PENALTY( $W, m, d, D$ )
2:    $V_a \leftarrow \{i \mid m_i = a\}$  for all  $a = 1, \dots, N_{\text{cluster}}$             $\triangleright$  nodes in cluster  $a$ 
3:   for  $a = 1, \dots, N_{\text{cluster}}$  do
4:      $L_a \leftarrow 0$                                  $\triangleright$  energy penalty for eliminating cluster  $a$ 
5:     for  $i \in V_a$  do
6:        $d_{\min} \leftarrow \infty$            $\triangleright$  minimum distance to a different cluster center
7:       for  $j \in V_a$  do                 $\triangleright$  look for connectivity via  $j$ 
8:         for  $k$  such that  $W_{k,j} > 0$  do           $\triangleright$  all neighbors of  $j$ 
9:           if  $m_k \neq m_j$  then             $\triangleright$  is  $k$  in a different cluster to  $j$ ?
10:             if  $d_k + W_{k,j} + D_{j,i} < d_{\min}$  then       $\triangleright$  is  $k$ 's center closer?
11:                $d_{\min} \leftarrow d_k + W_{k,j} + D_{j,i}$ 
12:              $L_a \leftarrow L_a + (d_{\min})^2$            $\triangleright$  add the new energy for  $i$ 
13:              $L_a \leftarrow L_a - \sum_{i \in V_a} (d_i)^2$          $\triangleright$  subtract the current energy metric
14:   return  $L$ 
```

Algorithm 3.9 Calculate the energy decrease that would result from optimally splitting each cluster in two. See Table 2 for variable definitions.

```

1: function SPLIT-IMPROVEMENT( $m, d, D$ )
2:    $V_a \leftarrow \{i \mid m_i = a\}$  for all  $a = 1, \dots, N_{\text{cluster}}$             $\triangleright$  nodes in cluster  $a$ 
3:   for  $a = 1, \dots, N_{\text{cluster}}$  do
4:      $S_a \leftarrow \infty$            $\triangleright$  energy improvement for splitting cluster  $a$ 
5:     for  $i \in V_a$  do           $\triangleright$  first possible new center
6:       for  $j \in V_a$  do           $\triangleright$  second possible new center
7:          $S_{\text{new}} \leftarrow 0$            $\triangleright$  energy with centers  $i$  and  $j$ 
8:         for  $k \in V_a$  do           $\triangleright$  compute cost for node  $k$ 
9:           if  $D_{i,k} < D_{j,k}$  then           $\triangleright$  is  $k$  closer to center  $i$  or  $j$ ?
10:              $S_{\text{new}} \leftarrow S_{\text{new}} + (D_{i,k})^2$ 
11:           else
12:              $S_{\text{new}} \leftarrow S_{\text{new}} + (D_{j,k})^2$ 
13:           if  $S_{\text{new}} < S_a$  then           $\triangleright$  is this a better split?
14:              $S_a \leftarrow S_{\text{new}}$            $\triangleright$  store the new energy
15:              $c_a^1 \leftarrow i$            $\triangleright$  store the new centers  $i$  and  $j$ 
16:              $c_a^2 \leftarrow j$ 
17:              $S_a \leftarrow \sum_{i \in V_a} (d_i)^2 - S_a$            $\triangleright$  improvement from current cluster energy
18:   return  $S, c^1, c^2$ 
```

Proof. We will show that all steps in the algorithm that change d_j or s_a result in a strict decrease of H_δ .

Case 1: updates by Lines 16 and 18 with d_j strictly decreasing. Then the reduction in the first term in H_δ is at least $(\Delta_{\min})^2$ and any increase in the second term in H_δ is less than $(N_{\text{node}})^2$, so the definition of δ means the decrease strictly dominates.

Case 2: updates by Lines 16 and 18 with d_j constant and $s_j > s_i + 1$. Then the first term in H_δ is constant and $s_{m_i} \leftarrow s_i + 1$ and $s_{m_j} \leftarrow s_j - 1$ results in a strict decrease of $(s_{m_i})^2 + (s_{m_j})^2$. \square

Algorithm 3.10 Mark a cluster and all of its neighbors as unavailable. See [Table 2](#) for variable definitions.

```

1: function MARK-UNAVAILABLE( $a, M, W, V_a$ )
2:    $M_a \leftarrow \text{False}$                                  $\triangleright$  cluster  $a$  is unavailable
3:   for  $i \in V_a$  do
4:     for  $j$  such that  $W_{i,j} > 0$  do           $\triangleright$  all neighboring nodes of cluster  $a$ 
5:        $M_{m_j} \leftarrow \text{False}$                    $\triangleright$  cluster of node  $j$  is unavailable

```

LEMMA 3.3. [Algorithm 3.5](#) results in a decrease of the energy [\(3.2\)](#), or preserves the energy if no change is made to the clustering.

Proof. Only updates by [Line 14](#) will change d_j . Because the change is caused by the use of j as the new center, and $q_j < q_i$, the first term in H_δ strictly decreases and the second term is unchanged. \square

THEOREM 3.4. [Algorithm 3.2](#) terminates, even if $T_{\max} = \infty$.

Proof. From [Lemmas 3.2](#) and [3.3](#), all steps in the algorithms that change d_i or s_a result in a strict decrease of H_δ . Because H_δ is positive and can only take a finite number of values, and we terminate when no changes are made, this ensures termination. \square

THEOREM 3.5. [Algorithm 3.7](#) results in a decrease or preservation of the energy [\(3.2\)](#).

Proof. Because of [Line 19](#), each elimination/split pairing explicitly results in a decrease of the first term in [\(3.2\)](#) and thus also a decrease in the overall value of H_δ due to the choice of δ . \square

To give bounds on the computational complexity of the algorithms we require the following assumptions on the graph structure.

Assumption 3.1. Assume that the number of edges in G incident on each vertex in the graph is bounded independently of N_{node} , and that the initial centers, c , are such that clusters found by [Algorithm 3.4](#) have size bounded independently of N_{node} .

THEOREM 3.6. Under [Assumption 3.1](#), if T_{\max} and $T_{BF\max}$ are both bounded independently of N_{node} , then the total cost of [Algorithm 3.2](#) is $\mathcal{O}(N_{\text{node}})$.

Proof. This follows from cost estimates for each of the components of [Algorithm 3.2](#). The inner loop of [Algorithm 3.4](#) has complexity equal to the number of edges in G , which is $\mathcal{O}(N_{\text{node}})$ by [Assumption 3.1](#). If $T_{BF\max} = \mathcal{O}(1)$, then [Algorithm 3.4](#) has complexity $\mathcal{O}(N_{\text{node}})$. The cost of Floyd-Warshall on each cluster is cubic in the cluster size, which we assume (as a function of the initial centers and clusters) to be $\mathcal{O}(1)$, giving a total cost of $\mathcal{O}(N_{\text{cluster}}) = \mathcal{O}(N_{\text{node}})$. [Algorithm 3.5](#) also has linear complexity. Since T_{\max} in [Algorithm 3.2](#) is also $\mathcal{O}(1)$, the total complexity is, then, $\mathcal{O}(N_{\text{node}})$. \square

THEOREM 3.7. Under [Assumption 3.1](#), [Algorithm 3.7](#) terminates with cost at most $\mathcal{O}(N_{\text{node}} \log N_{\text{node}})$ and, if T_{\max} is bounded independently of N_{node} , then [Algorithm 3.6](#) also has $\mathcal{O}(N_{\text{node}} \log N_{\text{node}})$ total cost.

Proof. From [Theorem 3.6](#), the cost of [Algorithm 3.2](#) is $\mathcal{O}(N_{\text{node}})$. Both [Algorithm 3.8](#) and [Algorithm 3.9](#) iterate over all clusters and perform bounded work per cluster (using the bound on cluster size from [Assumption 3.1](#)), so they have cost $\mathcal{O}(N_{\text{cluster}}) = \mathcal{O}(N_{\text{node}})$. Similarly, [Algorithm 3.10](#) has cost independent of N_{node}

because it only iterates over nodes within a single cluster. The cost of [Algorithm 3.7](#) is thus $\mathcal{O}(N_{\text{node}} \log N_{\text{node}})$ because [Lines 6](#) and [7](#) are $\mathcal{O}(N_{\text{cluster}} \log N_{\text{cluster}})$ and all other loops and subroutines are $\mathcal{O}(N_{\text{node}})$. Finally, assuming $T_{\max} = \mathcal{O}(1)$, we have the same cost for [Algorithm 3.6](#). \square

Remark 3.8. The algorithmic complexity of [Algorithm 3.7](#) and, hence, that of [Algorithm 3.6](#), can be reduced to $\mathcal{O}(N_{\text{node}})$ by changing the algorithm to separately treat fixed-size sets of clusters. That is, rather than considering all clusters at once, partition the set of clusters into subsets and run [Algorithm 3.7](#) separately on each subset. This will avoid the $\mathcal{O}(N_{\text{node}} \log N_{\text{node}})$ sorts in [Lines 6](#) and [7](#), leaving the cost as linear in N_{node} . We expect this would lead to some slight reduction in the quality of the rebalance, because eliminate/split pairings will only be considered within a subset but, for large subsets, we would not expect this to make a significant difference. This subset approach is also the natural way to parallelize [Algorithm 3.7](#), with one subset per processor.

Remark 3.9. Parallelization of [Algorithms 3.2](#) and [3.6](#) relies on parallelization of the other underlying algorithms. Both [Algorithms 3.1](#) and [3.5](#) operate independently on each cluster and are, thus, naturally parallelizable. [Algorithm 3.4](#) could be naturally parallelized by applying it independently to the set of nodes owned by each processor in a parallel decomposition.

4. Numerical Results. In this section, we highlight the value of balanced Lloyd clustering with rebalancing for smoothed aggregation multigrid. All computations are performed with PyAMG [3]. Unless stated otherwise, all results below consider a standard Poisson problem of form

$$(4.1a) \quad -\nabla \cdot \nabla U = F \quad \text{in } \Omega, \\ (4.1b) \quad \vec{n} \cdot \nabla U = 0 \quad \text{on } \partial\Omega,$$

where Neumann boundary conditions are used to highlight clustering near the boundary.² [Equation \(4.1\)](#) is discretized using either standard P^1 linear finite elements on a triangulation of the domain, Ω , or Q^1 bilinear finite elements on a quadrilateral mesh of Ω , yielding a matrix problem of the form

$$(4.2) \quad Au = f.$$

In the following convergence tests, f is set to zero and a random approximation to u is used to initialize the AMG cycling.

We consider three main cases of clustering in the context of AMG: standard Lloyd clustering ([Algorithm 2.3](#)), balanced Lloyd clustering ([Algorithm 3.2](#)), and balanced Lloyd clustering with rebalancing ([Algorithm 3.7](#)). For each of these, we require a definition of the weight matrix, W , and the number of clusters, N_{cluster} . In each case, we bound the number of inner iterations of Lloyd clustering at five and the number of rebalance sweeps at four; in practice, this is a conservative bound and the iterations complete much earlier (due to no change in the clustering state).

To form the weight matrix, W , we consider the so-called *evolution* measure [18] which associates a value of strength for each edge in the graph of A in [\(4.2\)](#) based on smoothing properties. This leads to an initial non-negative weight matrix, \widehat{W} , where a large edge value $\widehat{W}_{i,j}$ indicates that nodes i and j should be clustered together.

²With Dirichlet conditions, we would observe “singleton” clusters for the isolated points. This does not impact the method, only visualization.

The algorithms above make use of an assumption that the graph associated with W is connected, but this is not guaranteed to be the case for that of $\widehat{W}_{i,j}$. Thus, we augment \widehat{W} with a small padding for each edge in A , defining \widetilde{W} as

$$(4.3) \quad \widetilde{W}_{i,j} \leftarrow \widehat{W}_{i,j} + 0.1 \quad \text{if } A_{i,j} \neq 0$$

The Lloyd-based clustering presented here is based on shortest distances in the graph of W . As a result, we consider the inverse strength as a proxy for distance. This results in defining W so that

$$(4.4) \quad W_{i,j} = \frac{1}{\widetilde{W}_{i,j}} \quad \text{if } \widetilde{W}_{i,j} \neq 0,$$

so that strong edges refer to shorter distances. With this inversion, the additional padding added above indicates a long distance in the weight matrix.

4.1. Varying cluster numbers. While Greedy and MIS-based clustering have been used successfully in many settings, they do not provide a mechanism to control the number of resulting clusters. Here, we explore the ability of Lloyd clustering to target specific numbers of clusters. As motivation, consider the model problem on an unstructured triangulation of the unit disk with 10 245 vertices and 20 158 elements. We construct multigrid hierarchies using rebalanced Lloyd clustering, setting the target number of points in each cluster at each level to a fixed value between 3 and 20. We estimate the asymptotic convergence factor by the geometric mean of the last five residual norms at convergence, say k iterations:

$$(4.5) \quad \rho = \left(\frac{\|r^{(k)}\|}{\|r^{(k-5)}\|} \right)^{\frac{1}{4}},$$

where $r^{(k)} = f - Au^{(k)}$ is the residual vector after k iterations. Combined with a model for the *cost* of each multigrid cycle, given by the total number of non-zeros in the sparse-matrix operations in the cycle (i.e., the *cycle complexity*, χ), this leads to a measure of the work per digit of accuracy (WPD) for the method:

$$(4.6) \quad \text{WPD} = \frac{\chi}{-\log_{10}(\rho)}.$$

Figure 4 shows that the efficiency (and effectiveness) of an AMG method can vary depending on the (average) number of points per cluster; in this case, we observe that very small clusters lead to rapid convergence (small ρ), yet due to the slower coarsening, the total complexity of the multigrid cycle is higher.

In the end, balanced Lloyd clustering with rebalancing leads to well-formed clusters and the ability to use a vast range of cluster sizes. Figure 5 illustrates a range of cases for a smaller mesh of the same domain, from five (large) clusters at one extreme to 250 small clusters including singleton and many pairwise clusters. True pairwise clustering [16, 5] is not represented; however, it remains an open question whether a Lloyd-type algorithm could render nearly pairwise clustering using modified criteria for tiebreaking and rebalancing.

4.2. Tiebreaking. Algorithm 3.4 introduces “tiebreaking” on Line 12. If a node in the graph is equidistant from multiple centers, then the node becomes a member of the neighboring cluster if the neighboring cluster is smaller by two in size than

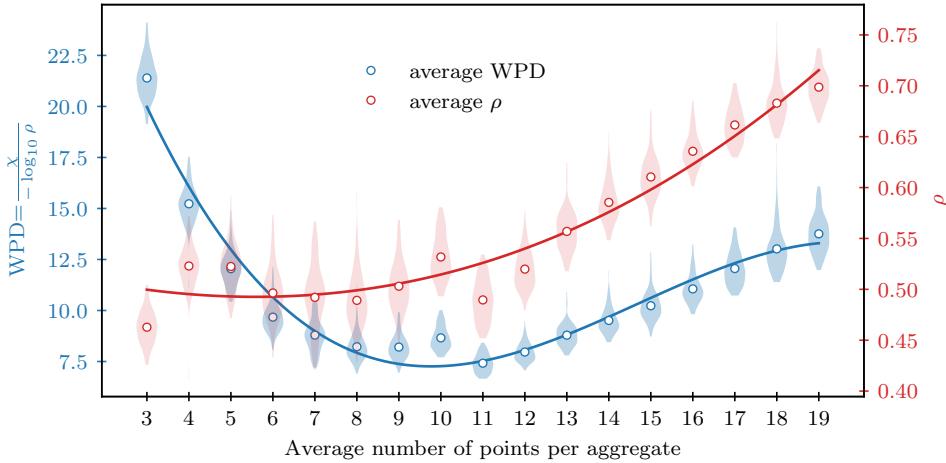


FIG. 4. Work per digit (WPD) of accuracy and convergence ρ for clustering sizes ranging from 3–19 points per cluster (on average) using rebalanced Lloyd clustering. The average over 100 runs is marked \circ and a trendline from a smoothed cubic spline is given for the mean (solid).

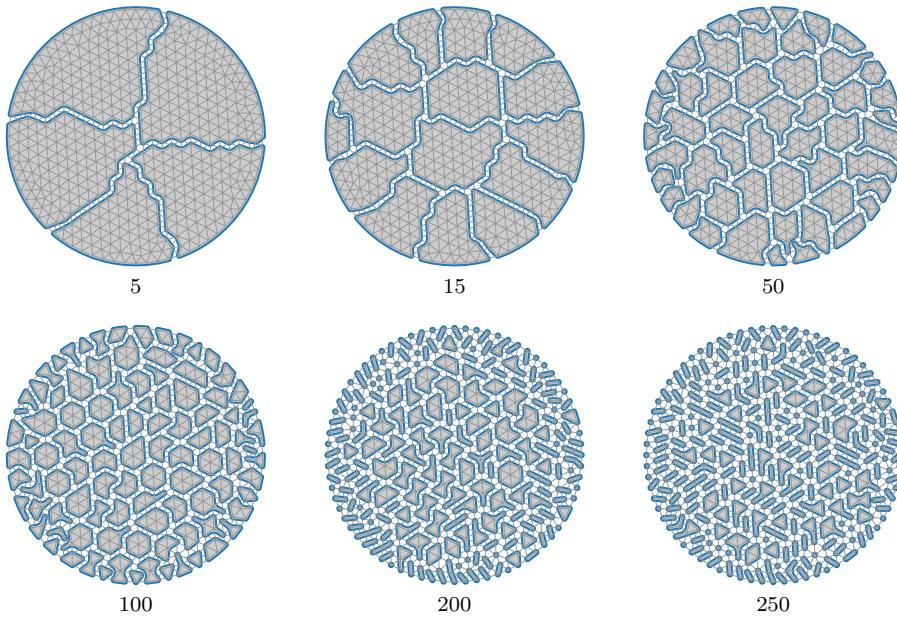


FIG. 5. Example clustering patterns with the number of clusters ranging from 5 to 250 using rebalanced Lloyd clustering.

the current cluster of the node. Tiebreaking in the balanced Bellman-Ford algorithm impacts the uniformity of the sizes of the clusters. To quantify uniformity, we consider discretizing (4.1) on a uniform 64×64 quadrilateral mesh. We cluster the nodes using the balanced Bellman-Ford algorithm *with* and *without* tiebreaking, requesting the number of clusters be equal to 10% of the fine-grid number of nodes (rounded down when this is not an integer). We randomly distribute the initial seeding 1000 times and, in each case, compute the following metrics: the number of zero diameter clusters

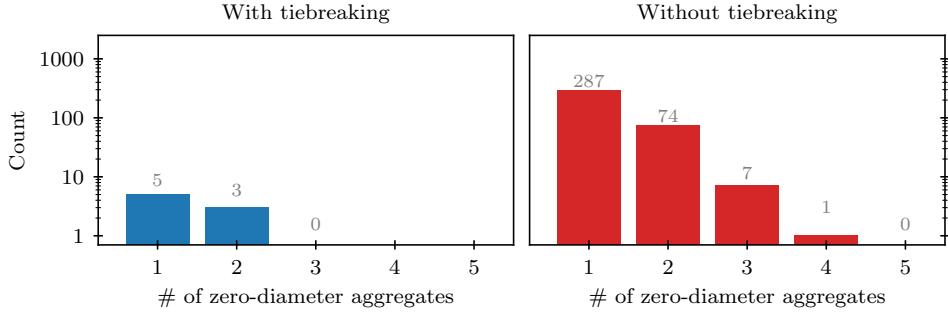


FIG. 6. Distribution of the number of clusters having zero diameter for balanced Lloyd clustering with or without tiebreaking for a 64×64 mesh.

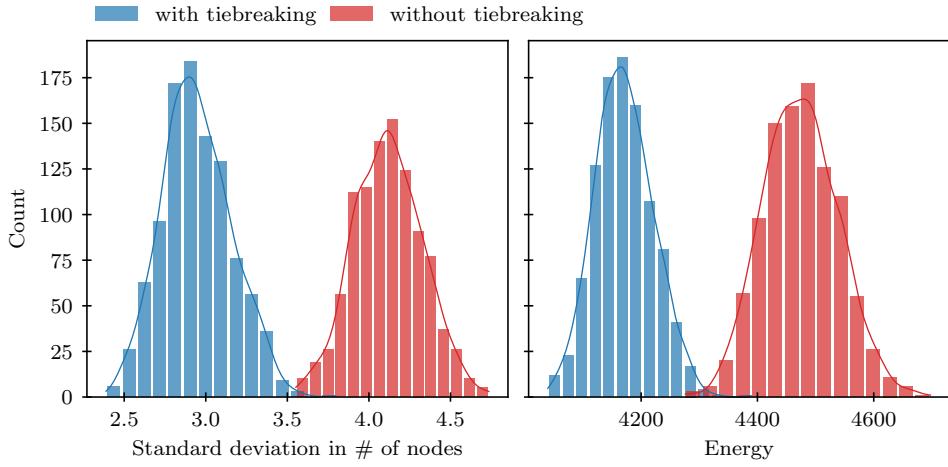


FIG. 7. Distribution of the standard deviation in the number of nodes and distribution of energy for balanced Lloyd clustering with or without tiebreaking on a 64×64 mesh.

(i.e., singleton clusters), the standard deviation in the number of nodes per cluster, and the energy for each clustering (defined by (3.1)).

Figure 6 shows the number of clusters having zero diameter with and without tiebreaking, highlighting that tiebreaking substantially decreases the number of clusterings with zero diameter clusters, from over one-third of clusterings to about one percent. Likewise, Figure 7 (left) shows the effect of tiebreaking on the distribution of the standard deviation in the number of nodes. Here, tiebreaking leads to a decrease yielding clusters more uniform in size. Tiebreaking also contributes to clusters that are more round — this is supported by Figure 7 (right), where we see that tiebreaking decreases the energy of the system. We emphasize that tiebreaking is an inexpensive strategy that clearly improves performance of the clustering.

4.3. Rebalancing. To quantify the improvements in cluster quality as we move from standard to balanced, and then to rebalanced Lloyd clustering, we again consider a 64×64 quadrilateral mesh. Nodes in this mesh are clustered using the three methods, again using 10% of N_{node} to determine N_{cluster} . In each case, the clustering is repeated 1000 times, yielding a standard deviation of cluster diameter, standard deviation of number of nodes in clusters, and energy for each test. The results are averaged and

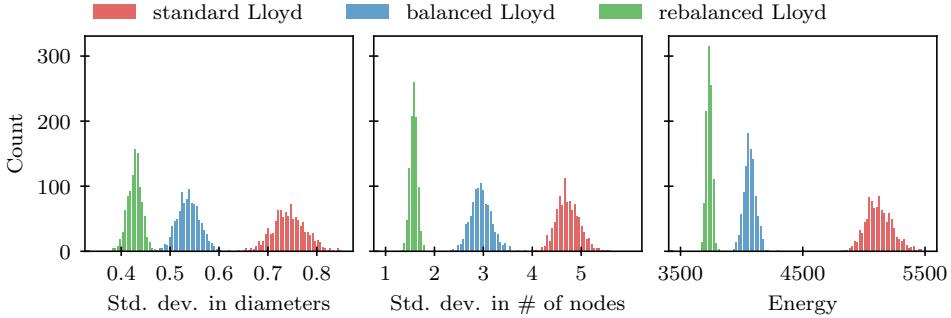


FIG. 8. Distribution of standard deviation in diameters, distribution of standard deviation in number of nodes, and distribution in energy for different clustering methods for a 64×64 mesh.

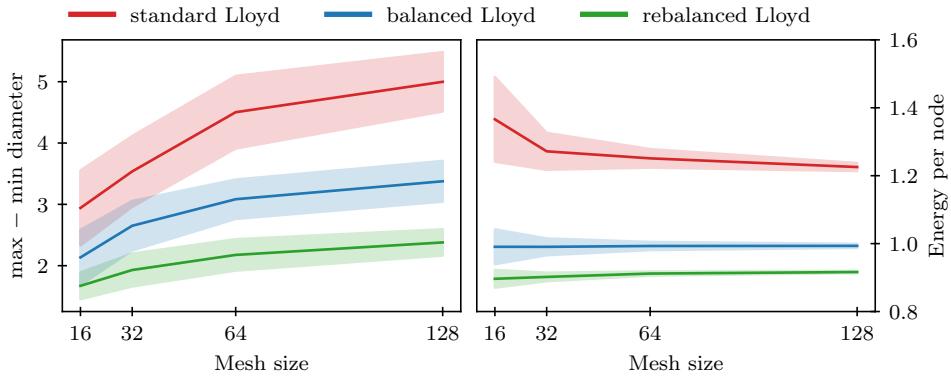


FIG. 9. (Left) Difference between maximum and minimum diameters of clusters averaged over 1000 samples; (Right) Energy per node averaged over 1000 samples. The shaded regions mark one standard deviation from the mean.

the same experiment is performed for 16×16 , 32×32 , and 128×128 meshes.

Figure 8 shows the distributions for each method in the case of a 64×64 mesh. Lower standard deviation of diameter and standard deviation of number of nodes suggest that the clusters that result from rebalanced Lloyd are more uniform in shape and size compared to the other methods. This is also reflected by the lower energy for rebalanced Lloyd clustering. The figure also highlights that variation in the metrics is lower for rebalanced Lloyd, pointing to the consistency in the method over multiple runs.

Figure 9 shows the difference between the maximum and minimum diameters and the energy, averaged over 1000 samples, for each of the clustering methods as we vary problem size. The figures underscore that rebalanced Lloyd yields more uniform, rounded clusters having less energy than the other two clustering methods as the mesh size grows.

4.4. Algebraic multigrid convergence. In the application of clustering to algebraic multigrid, cluster quality plays an important role in overall convergence of the method, but one that is not yet quantified by existing sharp measures. While we can easily confirm improvement (or degradation) in the measured convergence factor after making a change to a clustering, it is difficult to directly assess if an individual

cluster is the cause of poor convergence.

One way to *localize* a bound on AMG convergence is to consider the classical bound based on smoothing and approximation properties [20, 13]. This theory considers the convergence of a two-grid cycle with post-relaxation given by $u \leftarrow u + M(f - Au)$ and coarse-grid correction given by $u \leftarrow u + P(P^T AP)^{-1}P^T(f - Au)$. We write $G = I - MA$ and $T = I - P(P^T AP)^{-1}P^T A$ as the error-propagation operators of relaxation and coarse-grid correction, respectively, with the error-propagation operator of the two-grid scheme given by GT . The diagonal of SPD matrix A is denoted by D . In what follows, we assume that A is SPD, P is of full rank, and $\|G\|_A < 1$. Theorem 4 of [13] shows that if there exist constants $\alpha, \beta > 0$ such that

$$\begin{aligned}\|Ge\|_A^2 &\leq \|e\|_A^2 - \alpha\|e\|_{AD^{-1}A}^2 \text{ for all } e, \\ \text{and } \|Te\|_A^2 &\leq \beta\|Te\|_{AD^{-1}A}^2 \text{ for all } e,\end{aligned}$$

then $\|GT\|_A \leq (1 - \alpha/\beta)^{1/2}$. The first of these is known as the *smoothing property*, since it concerns the action of relaxation, G , on errors, e . The second is referred to as the *approximation property*, since it quantifies the action of the coarse-grid correction process. Equations (19) and (20) of [13] show that this approximation property is guaranteed by the existence of a constant $\beta > 0$ such that $\inf_{e_c} \|e - Pe_c\|_D^2 \leq \beta\|e\|_A^2$ for all e . Choosing $e_c = (P^T DP)^{-1}P^T De$ and defining $T_D = I - P(P^T DP)^{-1}P^T D$ then allows us to quantify such a β as

$$(4.7) \quad \beta = \sup_{e \neq 0} \frac{e^T T_D^T D T_D e}{e^T A e}.$$

We find this β by solving for the largest eigenvalue of the generalized eigenvalue problem $T_D^T D T_D e = \lambda A e$, and let e be the associated eigenvector. To localize the measure over a single cluster, we decompose the inner product in the numerator into a sum over clusters, writing $\beta = \sum_{i=1}^{N_{\text{cluster}}} \beta_i$, where

$$(4.8) \quad \beta_i = \frac{\left(\sum_{j \in \Omega_i} (DT_D e)_j (T_D e)_j \right)}{e^T A e}$$

This comes from writing the numerator of (4.7) as the inner product of $DT_D e$ with $T_D e$, and then localizing the summation in that inner product over each cluster.

We again consider the Poisson problem on a triangulation of the unit disk with 528 unknowns, and compute (4.8) for each cluster generated by each method. Figure 10 shows that the extreme values of β_i are reduced through rebalancing. Indeed, this is reflected in the convergence shown in Figure 11, where we observe a dramatic reduction in the number of iterations for solvers with these clusters. It is, of course, important to note that not *every* clustering generated by standard Lloyd exhibits similarly poor performance. The quality of the initial clustering used to seed the algorithm plays an important role in determining the multigrid performance. Results seen here are for a representative, randomly generated, initial seeding.

4.5. Additional problems in Algebraic Multigrid. As additional evidence of the effectiveness of rebalanced Lloyd clustering, we consider several examples in both 2D and 3D.

3D restricted channel: The 3D domain Ω is defined by a spline on the points

$$[(0, 4, -8), (0, 4, -6), (0, 1, 0), (0, 4, 6), (0, 4, 8)],$$

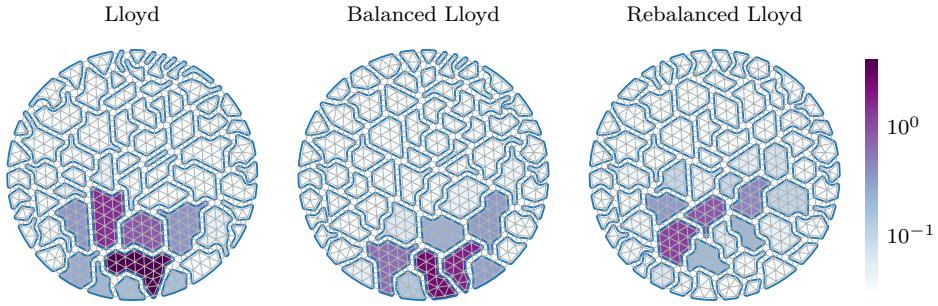
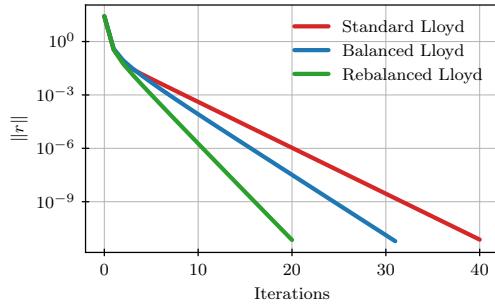
FIG. 10. Localizing β to each cluster via (4.8).

FIG. 11. Example convergence for two-level AMG with different clusterings.

rotated about the z -axis, (see Table 1). A 3D tetrahedral mesh with 16 921 elements is generated with Gmsh [9] through pygmsh [21]. We use Firedrake [19] to discretize (4.1) with linear finite elements on tetrahedra, and select an average of 25 points per cluster.

2D restricted channel: The 2D domain Ω is defined by $[-2, 2] \times [-1, 1] \setminus C$ with $C = C^+ \cup C^-$, for C^\pm representing discs of radius 0.8 at $(0, \pm 1)$ (see Table 1). As for the 3D restricted channel, we use Gmsh to generate a graded, triangular mesh with 5832 elements, with a characteristic length of 0.012 at the center and growing to 0.12 at the left/right edges. This forces tighter clustering toward the center, as shown in Table 1. The discretization matrix for (4.1) is constructed with linear finite elements, and we target clusters of size 8.

2D anisotropic diffusion: The 2D domain is defined by the unit square, and we consider the problem $-\nabla \cdot K \nabla u = f$ with pure Dirichlet conditions. We define the anisotropic diffusion tensor as $K = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} 1 & \\ & \varepsilon \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix}^T$, for $\varepsilon = 0.1$ and $\theta = \pi/3$. We discretize this on a 42×42 uniform mesh (with 1681 elements) and Q1 bilinear elements, and specify a target cluster size of 12.

P2 elements: The 2D domain is a unit disc, on which we consider (4.1). A triangular mesh is constructed with 982 elements and P2 quadratic finite elements are used to generate the discretization matrix. We specify 5 nodes per cluster.

In each of the examples of Table 1, a zero right-hand side is used to assess convergence of the smoothed aggregation multigrid solver. From the convergence histories, we see that rebalanced Lloyd clustering improves solver convergence, even for these relatively benign problems. For the restricted channel problems, the resulting clustering resembles the expected isotropic behavior with well rounded clusters. Likewise,

in the case of anisotropy, we see that the clustering mimics the diffusion direction, while maintaining balance across clusters. Finally, the P2 case reveals the benefit of specifying the coarsening ratio: in this case, the coarsening ratio of 1/5 outperforms greedy coarsening (which yields a ratio of around 1/10).

As a final example, [Table 1](#) highlights a parallel partitioning of an arc heated combustion channel at the University of Illinois Urbana-Champaign³. In this case, rebalanced Lloyd effectively partitions the $\sim 100k$ mesh elements, keeping refined features such as the injector local to a cluster.

5. Conclusions and extensions. In this paper, we study and extend the use of Lloyd’s algorithm for determining clusters in graphs. Our proposed *balanced* and *rebalanced* Lloyd clustering algorithms are linear in time, guarantee connected clusters, and are consistent with minimizing a quadratic energy functional. In addition, the algorithms are implemented in Python/C++ and are available through the open source project PyAMG [3]. One major topic for future work is the choice of that energy functional; while the steps in the algorithms above are consistent with an ℓ^2 -distance style energy, they can easily be extended to other energy functionals in a consistent way. Theoretical guidance is clearly needed to determine the proper choice of such a functional. We also note that we consider only serial algorithms in this paper; properly extending these approaches to their parallel counterparts is also an important subject for future research.

REFERENCES

- [1] D. ARTHUR AND S. VASSILVITSKII, *K-Means++: The advantages of careful seeding*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’07, USA, 2007, Society for Industrial and Applied Mathematics, p. 1027–1035.
- [2] N. BELL, S. DALTON, AND L. N. OLSON, *Exposing fine-grained parallelism in algebraic multigrid methods*, SIAM Journal on Scientific Computing, 34 (2012), pp. C123–C152.
- [3] N. BELL, L. N. OLSON, AND J. SCHRODER, *PyAMG: Algebraic multigrid solvers in Python*, Journal of Open Source Software, 7 (2022), p. 4142, <https://doi.org/10.21105/joss.04142>.
- [4] W. N. BELL, *Algebraic multigrid for discrete differential forms*, PhD thesis, University of Illinois at Urbana-Champaign, 2008, <http://hdl.handle.net/2142/81820>.
- [5] J. BRANNICK, Y. CHEN, J. KRAUS, AND L. ZIKATANOV, *Algebraic multilevel preconditioners for the graph laplacian based on matching in graphs*, SIAM Journal on Numerical Analysis, 51 (2013), pp. 1805–1827, <https://doi.org/10.1137/120876083>.
- [6] M. BREZINA, J. MANDEL, ET AL., *Convergence of algebraic multigrid based on smoothed aggregation*, Numerische Mathematik, 88 (2001), pp. 559–579.
- [7] W. L. BRIGGS, V. E. HENSON, AND S. F. MCCORMICK, *A Multigrid Tutorial*, SIAM Books, Philadelphia, 2000. Second edition.
- [8] J. ERICKSON, *Algorithms*, 2019, <http://jeffe.cs.illinois.edu/teaching/algorithms/>.
- [9] C. GEUZAINE AND J.-F. REMACLE, *Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities*, International Journal for Numerical Methods in Engineering, 79 (2009), pp. 1309–1331, <https://doi.org/10.1002/nme.2579>.
- [10] G. KARYPIS AND V. KUMAR, *METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*, 2022, <https://github.com/KarypisLab/METIS>. v5.1.1.
- [11] B. KELLEY AND S. RAJAMANICKAM, *Parallel, portable algorithms for distance-2 maximal independent set and graph coarsening*, in 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), Los Alamitos, CA, USA, June 2022, IEEE Computer Society, pp. 280–290, <https://doi.org/10.1109/IPDPS53621.2022.00035>.
- [12] S. LLOYD, *Least squares quantization in PCM*, IEEE Transactions on Information Theory, 28 (1982), pp. 129–137.
- [13] S. MACLACHLAN AND L. OLSON, *Theoretical bounds for algebraic multigrid performance: review and analysis*, Numer. Linear Alg. Appl., 21 (2014), pp. 194–220.

³<https://tonghun.mechse.illinois.edu/research/hypersonics-act-ii/>, <https://ceesd.illinois.edu/>

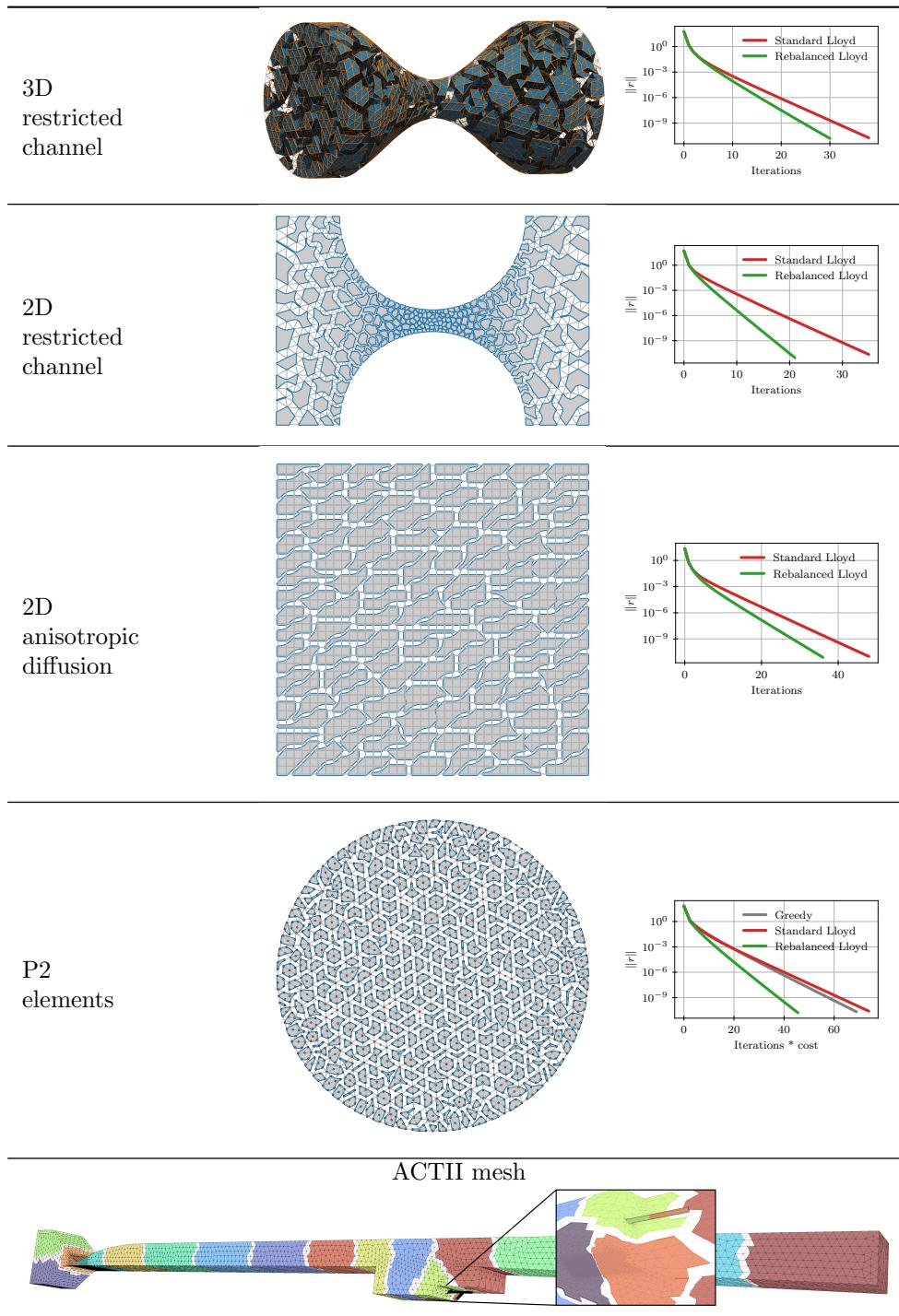


TABLE 1
Additional examples. ACTII mesh credit: Mike Anderson at UIUC.

- [14] T. A. MANTEUFFEL, L. N. OLSON, J. B. SCHRODER, AND B. S. SOUTHWORTH, *A root-node-based algebraic multigrid method*, SIAM J. Sci. Comput., 39 (2017), pp. S723–S756, <https://doi.org/10.1137/16M1082706>.
- [15] S. MÍKA AND P. VANĚK, *Acceleration of convergence of a two-level algebraic algorithm by aggregation in smoothing process*, Applications of Mathematics, 37 (1992), pp. 343–356.
- [16] A. NAPOV AND Y. NOTAY, *An algebraic multigrid method with guaranteed convergence rate*, SIAM Journal on Scientific Computing, 34 (2012), pp. A1079–A1109, <https://doi.org/10.1137/100818509>.
- [17] Y. NOTAY, *An aggregation-based algebraic multigrid method*, Electronic transactions on numerical analysis, 37 (2010), pp. 123–146.
- [18] L. N. OLSON, J. SCHRODER, AND R. S. TUMINARO, *A new perspective on strength measures in algebraic multigrid*, Numerical Linear Algebra with Applications, 17 (2010), pp. 713–733.
- [19] F. RATHGEBER, D. A. HAM, L. MITCHELL, M. LANGE, F. LUPORINI, A. T. T. MCRAE, G.-T. BERCEA, G. R. MARKALL, AND P. H. J. KELLY, *Firedrake: automating the finite element method by composing abstractions*, ACM Trans. Math. Softw., 43 (2016), pp. 24:1–24:27, <https://doi.org/10.1145/2998441>.
- [20] J. W. RUGE AND K. STÜBEN, *Algebraic multigrid (AMG)*, in Multigrid Methods, S. F. McCormick, ed., vol. 3 of Frontiers in Applied Mathematics, SIAM, Philadelphia, PA, 1987, pp. 73–130.
- [21] N. SCHLÖMER, *pygmsh: A Python frontend for Gmsh*, <https://doi.org/10.5281/zenodo.1173105>, <https://github.com/nschloe/pygmsh>.
- [22] K. STÜBEN, *An introduction to algebraic multigrid*, in Multigrid, U. Trottenberg, C. Oosterlee, and A. Schüller, eds., Academic Press, London, 2001, pp. 413–528.
- [23] U. TROTTERBERG, C. W. OOSTERLEE, AND A. SCHÜLLER, *Multigrid*, Academic Press, London, 2001.
- [24] R. S. TUMINARO AND C. TONG, *Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines*, in SuperComputing 2000 Proceedings, J. Donnelley, ed., 2000.
- [25] P. VANĚK, *Acceleration of convergence of a two-level algorithm by smoothing transfer operators*, Applications of Mathematics, 37 (1992), pp. 265–274.
- [26] P. VANĚK, J. MANDEL, AND M. BREZINA, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.

Appendix A. Review of algebraic multigrid methods. Algebraic multigrid methods seek to approximate solutions to sparse linear systems of the form

$$(A.1) \quad Au = f$$

for $A \in \mathbb{R}^{N_{\text{node}} \times N_{\text{node}}}$, and $u, f \in \mathbb{R}^{N_{\text{node}}}$. Here, we outline *aggregation-based AMG* methods for use as an application in the development of Lloyd-style clustering. The set of indices, $\{1, \dots, N_{\text{node}}\}$, enumerate the degrees of freedom (DoFs) and represent the fine level in the multilevel grid hierarchy. This set is partitioned and grouped into disjoint clusters, see [Definition 2.1](#).

Each cluster represents a node in the coarse grid and, collectively, the cluster mapping defines a tentative restriction operator, \hat{R} , as

$$(A.2) \quad \hat{R}_{a,i} = \begin{cases} 1 & \text{if vertex } i \text{ is in cluster } a, \\ 0 & \text{otherwise.} \end{cases}$$

An example with 12 fine nodes and 3 coarse nodes (clusters) is given in [Figure 12](#); the pattern for (the transpose of) \hat{R} is also illustrated.

The restriction pattern defines the tentative interpolation pattern through $\hat{Z} = \hat{R}^T$. Smoothed aggregation (SA) AMG proceeds by using the nonzero pattern of \hat{Z} as a partition of unity to localize a given global set of vectors, C , defining the near-null space of matrix A and, then, smoothing each column of the resulting matrix, Z , with (for example) weighted Jacobi. This defines the smoothed interpolation operator, Z , from which a coarse-level operator is defined over cluster DoFs as $A_c = Z^T AZ$.

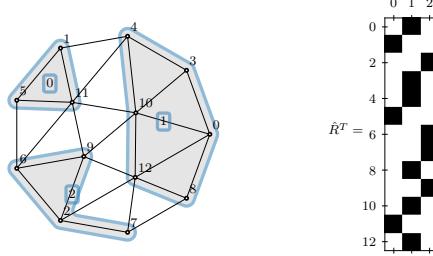


FIG. 12. Example clustering and restriction matrix.

The complete algorithm for constructing SA AMG is given in [Algorithm A.1](#), where we note the omission of several details (and optional parameters, denoted by $[opt]$) since the focus of this work is primarily on [Line 4](#). We refer the reader to [\[26, 6, 22\]](#) for a more complete description and analysis of aggregation-based AMG methods. Here, we note that [Line 3](#) is critically important to the convergence of the method; in practice, unit weights or algebraic distances *can* be used, yet generalized measures such as the *evolution* measure [\[18\]](#) (used in [section 4](#)) have proven robust in practice.

Algorithm A.1 Smoothed aggregation — setup

```

1: function SA-SETUP( $A_0$ ,  $N_{\text{level}}$ ,  $C$ )
2:   for  $\ell \leftarrow 0, \dots, N_{\text{level}} - 1$  do
3:      $W \leftarrow \text{EDGE-WEIGHTS}(A_\ell, [opt])$      $\triangleright$  determine strong edges in graph of  $A$ 
4:      $m, c \leftarrow \text{CLUSTER}(W, [opt])$            $\triangleright$  cluster membership and centers
5:      $Z_\ell \leftarrow \text{INTERPOLATION}(m, C, [opt])$      $\triangleright$  form interpolation
6:      $A_{\ell+1} = Z_\ell^T A_\ell Z_\ell$                    $\triangleright$  construct coarse-level operator
7:   return  $\{A_\ell\}_0^{N_{\text{level}}}, \{Z_\ell\}_0^{N_{\text{level}}-1}$ 
```

With a multigrid hierarchy of coarse operators and interpolation, multigrid (MG) iterates via the familiar V-cycle as in [Algorithm A.2](#). In [section 4](#), we have considered both two-level ($N_{\text{level}} = 2$) and multilevel results, underscoring improved convergence by improving the clustering, while leaving the other multigrid parameters untouched. As we use a subscript within these algorithms to denote the level within the multigrid hierarchy, we use a superscript to indicate the multigrid iteration number, with $u^{(k+1)} = \text{MG-V-CYCLE}(A_0, \dots, A_{N_{\text{level}}}, Z_0, \dots, Z_{N_{\text{level}}-1}, u^{(k)}, f)$.

Algorithm A.2 MG cycle

```

1: function MG-V-CYCLE( $A_0, \dots, A_{N_{\text{level}}}, Z_0, \dots, Z_{N_{\text{level}}-1}, u_0, f_0$ )
2:   for  $\ell = 0, \dots, N_{\text{level}} - 1$  do
3:      $u_\ell \leftarrow \text{RELAX}(A_\ell, u_\ell, f_\ell)$            $\triangleright$  fixed number of relaxation sweeps
4:      $f_{\ell+1} \leftarrow Z_\ell^T (f_\ell - A_\ell u_\ell)$        $\triangleright$  compute restricted residual
5:      $u_{N_{\text{level}}} \leftarrow A_{N_{\text{level}}}^{-1} f_{N_{\text{level}}}$      $\triangleright$  solve coarsest level problem
6:   for  $\ell = N_{\text{level}} - 1, \dots, 0$  do
7:      $u_\ell \leftarrow u_\ell + Z_\ell u_{\ell+1}$              $\triangleright$  interpolate and correct
8:      $u_\ell \leftarrow \text{RELAX}(A_\ell, u_\ell, f_\ell)$        $\triangleright$  fixed number of relaxation sweeps
9:   return  $u_0$ 
```

Appendix B. Notation. [Table 2](#) summarizes the notation.

Symbol	Definition	Domain
A	left hand side operator in the linear system $Au = f$	$\mathbb{R}^{N_{\text{node}} \times N_{\text{node}}}$
a	cluster index	$\{1, \dots, N_{\text{cluster}}\}$
B	set of border nodes between clusters; $B \subseteq V$	$\mathcal{P}(V)$
c_a	center node index for cluster a	V
c_a^1, c_a^2	new cluster centers if cluster a is split; see Algorithm 3.9	V
δ	second-term energy scaling coefficient, see (3.3)	\mathbb{R}
d_i	shortest-path distance to node i from nearest center; $d_i = \infty$ if node i is not in a cluster	$\bar{\mathbb{R}}_{\geq 0}$
$D_{i,j}$	shortest-path distance from node i to j within a single cluster; $D_{i,j} = \infty$ if there is no such path $i \rightarrow j$	$\bar{\mathbb{R}}_{\geq 0}$
Δ_{\min}	minimum difference between distinct values of $W_{i,j}$	\mathbb{R}
E	set of edges in the graph G	$\mathcal{P}(V \times V)$
f	right hand side of the linear system $Au = f$	$\mathbb{R}^{N_{\text{node}}}$
G	graph with nodes V , edges E , and weights W	
H	shortest-path energy function, see (3.1)	\mathbb{R}
H_δ	energy function minimized by clustering, see (3.2)	\mathbb{R}
i, j, k	node indices	V
L_a	energy increase if cluster a is eliminated; see Algorithm 3.8	\mathbb{R}
M_a	whether cluster a is modifiable during rebalancing	{True, False}
m_i	cluster index (membership) containing node i	$\{1, \dots, N_{\text{cluster}}\}$
N_{node}	number of nodes	\mathbb{N}_1
N_{cluster}	number of clusters	\mathbb{N}_1
n_i	number of nodes with i as predecessor	\mathbb{N}_0
$P_{i,j}$	predecessor index for node j on the shortest path $i \rightarrow j$ within a cluster; $P_{i,j} = 0$ if there is no path $i \rightarrow j$	$V \cup \{0\}$
p_i	predecessor index for node i on the shortest path from its cluster center; $p_i = 0$ if node i is not in a cluster	$V \cup \{0\}$
q_i	sum of squared distances from node i to all other nodes in the same cluster	$\bar{\mathbb{R}}_{\geq 0}$
S_a	energy decrease if cluster a is split; see Algorithm 3.9	\mathbb{R}
s_a	size (number of nodes) of cluster a	\mathbb{N}_1
\mathfrak{s}_i	size (number of nodes) of the cluster containing node i ; $\mathfrak{s}_i = 0$ if node i is not in a cluster	\mathbb{N}_0
T	total number of time/iteration steps taken by an algorithm (T_{\max} and T_{BFmax} denote the maximum)	\mathbb{N}_0
t	time/iteration index	\mathbb{N}_0
u	solution vector in the linear system $Au = f$	$\mathbb{R}^{N_{\text{node}}}$
V	set of nodes in the graph; $V = \{1, \dots, N_{\text{node}}\}$	$\mathcal{P}(\mathbb{N}_1)$
V_a	set of nodes in cluster a ; $V_a \subseteq V$	$\mathcal{P}(V)$
$W_{i,j}$	weighted adjacency matrix of the graph where $W_{i,j}$ is the edge weight $i \rightarrow j$	\mathbb{R}
z	generic variable placeholder	—

TABLE 2

List of symbols. Here $\mathcal{P}()$ denotes the power set and $\bar{\mathbb{R}}$ is the extended reals.