LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-JRNL-777449

Multilevel Graph Embedding

B. G. Quiring, P. S. Vassilevski

June 10, 2019

Numerical Linear Algebra with Applications

Disclaimer

MULTILEVEL GRAPH EMBEDDING

BENJAMIN QUIRING† AND PANAYOT S. VASSILEVSKI

Abstract. The goal of the present paper is the design of embeddings of a general sparse graph into a set of points in $\mathbb{R}^d$ for appropriate $d \geq 2$. The embeddings that we are looking at aim to keep vertices that are grouped in communities together and keep the rest apart. To achieve this property, we utilize coarsening that respects possible community structures of the given graph. We employ a hierarchical multilevel coarsening approach that identifies communities (strongly connected groups of vertices) at every level. The multilevel strategy allows any given (presumably expensive) graph embedding algorithm to be made into a more scalable (and faster) algorithm. We demonstrate the presented approach on a number of given embedding algorithms and large-scale graphs and achieve speed-up over the methods in a recent paper.

1. Introduction

Data represented by graphs, possibly with edge weights representing relations between objects, generally have no geometric positions and hence no immediate numerical coordinates can be associated with them. However, for many graph analysis tools it is useful to place the vertices in a Euclidean space $\mathbb{R}^d$ for appropriate d so that one is able to identify neighbors and measure the distance between nearby objects. In particular, with coordinates (for $d \leq 3$) one is able to visualize the graph. This paper studies existing and proposes new algorithms for graph *embedding* with the ultimate goal to make them applicable to large-scale graphs. The approach we explore is not tied to a specific existing graph embedding algorithm, but aims to handle any existing one that is potentially expensive or infeasible at large scale, by applying it on smaller graphs and subgraphs. The smaller graphs are generated recursively in a multilevel fashion utilizing an appropriate graph coarsening algorithm. The proposed multilevel approach provides a general framework which is very powerful in the sense that it can be feasible at large scale for many existing graph embedding algorithms.

The multilevel concept is standard tool for achieving scalability of algorithms on sparse data. For example, the graph partitioning algorithm, METIS [1] exhibits this nature. The coarsening, especially aggregation coarsening, is a common tool for many algebraic multigrid (AMG) algorithms applied to sparse matrices (e.g., [4], [5], and more recently, [6], [7] and many other references therein). The AMG algorithms exploit not only the sparsity graph of the matrices but also the matrix entries.

Date: April 30, 2018–beginning; Today is May 1, 2020.

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

Key words and phrases. graphs, multilevel algorithms, embeddings.

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

In our setting of graphs with possible edge weights, we chose to use multilevel coarsening algorithms that respect communities, such as a scalable variant of modularity based pairwise matching coarsening [13].

We note though, that the multilevel approach we study does not depend on the actual coarsening method used, so the details of the coarsening are only briefly mentioned. More details on coarsening subject to certain objectives can be found in [1] (minimizing *surface-to-volume* ratio or *edge cut*) as well as [9] and [15], see also [13] (maximizing the modularity functional).

We also note that a multilevel approach with the same goal (to be agnostic of the coarsening and the given embedding algorithm) was recently utilized in [12]. The difference between their method and ours is in the refinement of the embedding: when an embedding for a coarse graph has been computed, to use it as a starting point for embedding the next finer level graph the authors in [12] utilize a convolutional neural network. To achieve scalability, they use its meta parameters trained at the coarsest level.

The method we propose uses the clusters obtained at every level of our multilevel coarsening to apply a given embedding algorithm locally to the clusters at every level, maintaining the relative positioning of the clusters (which are vertices of the graph one level coarser) and ensuring that the locally embedded clusters do not geometrically overlap. This allows us to also achieve scalability while still obtaining high-quality embeddings that approximate more expensive methods, with a speed-up over the algorithms presented in [12]. In summary, using the clusters at each coarsening level is a key ingredient of our multilevel approach.

The remainder of the present paper is structured as follows. In Section 2, we introduce some basic definitions and provide a brief outline of our approach. In the following Section 3, we describe the refinement procedure for using the coarser level embedding to generate the one level finer embedding. The main section, Section 4, describes how any given embedding algorithm can be employed in our multilevel setting to achieve scalability and lead to a faster, and generally more computationally feasible algorithm. At the end we provide some conclusions and an outlook in Section 5.

2. Preliminaries and outline

2.1. **Definitions.** A graph $G = (V, E)$ is a set of n vertices $V = V(G)$ and a set of m edges $E = E(G)$, where each edge $e \in E$ is an unordered pair of vertices (i, j), $i,\ j \in V$. A partition π of a graph G is a partition of its vertex set, V. A set $\mathcal{A} \in \pi$ of this partition is called an aggregate (syn. community, cluster, coarse vertex). Script capital letters $\mathcal{A}$, $\mathcal{B}$, *etc.* denote aggregates while lowercase i, j, *etc.* denote vertices in G, also referred to as fine level vertices.

Associated with G are values $\{a_{ij}\}_{i,j \in V}$, where a_{ij} is the weight of the edge between vertices i and j. If an edge $e = (i, j) \notin E$ does not exist in the graph its weight is taken to be 0. The $n \times n$ matrix $A = (a_{ij})$ is referred to as the *adjacency* matrix of G. We assume G is an undirected graph, making A a symmetric matrix.

Given a partition π one can form the coarse graph $G_c = \pi(G)$: the coarse vertices are the aggregates of π, and the coarse edges are the pairs $(\mathcal{A}, \mathcal{B})$ for which there

exists $i \in \mathcal{A}$ and $j \in \mathcal{B}$ such that $e = (i, j) \in E$. We can define coarse edge weights

$$a_{\mathcal{AB}} = \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{B}} a_{ij}.$$

Assuming the edge weights a_{ij} represent connectivity strength, the strength of connectivity between two aggregates is similarly measured by $a_{\mathcal{AB}}$.

An embedding of a graph $G = (V, E)$ into $\mathbb{R}^d$ is a function $f : V \to \mathbb{R}^d$. If G is finite, such embeddings are bounded and so can be scaled to be within the unit sphere or the unit cube in $\mathbb{R}^d$. Given a finite graph G, we would like to find an embedding of G in $\mathbb{R}^d$ for given d. This embedding should respect community structure; vertices that are not in the same communities should be far apart and those in the same community should be close. There are known algorithms which attempt to do this, such as the physics-inspired ForceAtlas2 algorithm [8]. These methods can become unwieldy for large graphs, however.

2.2. **Outline of the multilevel strategy.** To embed a large (sparse) graph G, we propose the following process: first partition G in a multilevel fashion via some method giving graphs $G = G_1, ..., G_c$ related via partitions $\{\pi_\ell\}_{\ell=1}^{c-1}$ such that $\pi_\ell(G_\ell) = G_{\ell+1}$. This hierarchy has coarsening factor α if for each i, $\alpha \approx \frac{|V(G_{i+1})|}{|V(G_i)|}$. Instead of directly embedding G we seek first to embed the coarsest level G_c using a possibly expensive, state-of-the-art, method. This initial embedding should place aggregates that are not connected to each other far apart for best results. Once an embedding for level ℓ has been obtained it is refined, creating an embedding for level $\ell - 1$. Applying such a procedure iteratively allows each of $G_{c-1}, ..., G_2, G_1 = G$ to be embedded via the original embedding for G_c and the refinement method. As already mentioned, multilevel approaches have been employed in [12] which use neural nets to move from layer ℓ to layer $\ell - 1$. We take a different approach (not involving neural nets) which attempts to use the clusters at every coarsening level.

Next, we provide a brief outline of the coarsening scheme that we have implemented; more details are found in [13]. We first introduce the *modularity* functional [2] for a given weighted graph. Let $A = (a_{ij})$ be the weighted adjacency matrix associated with the graph, i.e., a_{ij} is nonzero if (i, j) is an edge of G. We allow for negative entries, but require that the rowsums (weighted degree) $wdeg(i) = \sum_j a_{ij}$ be positive. Also, let $T = \sum_i wdeg(i)$ be the total rowsum. The popular modularity functional associated with the partitioning $\{\mathcal{A}\}$ reads [2],

$$\mathcal{Q} = \frac{1}{T} \sum_{\mathcal{A}} \sum_{i,j \in \mathcal{A}} \left(a_{ij} - \frac{1}{T} wdeg(i) wdeg(j) \right).$$

which, at an intuitive level, has the effect of measuring the degree to which the clustering reflects actual community structure by balancing aggregate size while implicitly comparing the edges internal to communities with the edges external to communities. Specifically, $\frac{1}{T} wdeg(i) wdeg(j)$ can be interpreted as the probability that an edge between i and j may exist and the sum over the $a_{ij} - \frac{1}{T} wdeg(i) wdeg(j)$ is inspecting whether the distribution of edges internal to a community is more or less than

expected, given the probabilities. Also, $\mathcal{Q}$ lies in the interval $[-1, 1]$, and higher $\mathcal{Q}$ values are associated with a better community structure (as the distribution of edges internal to the community is higher than expected by just looking at the weighted degree of vertices), so the ultimate goal is to find aggregates which maximize $\mathcal{Q}$.

At the original fine level each $\mathcal{A}$ corresponds to a single vertex $\{i\}$. If we merge (match) two vertices i and j and form a new aggregate, the modularity will change, and the corresponding change we denote by $\Delta\mathcal{Q}_{ij}$. The modularity-based coarsening that we employ attempts to merge two neighboring vertices with large positive $\Delta\mathcal{Q}_{ij}$.

Our coarsening scheme utilizes $\Delta\mathcal{Q}_{ij}$ as edge weights. It is straightforward to see that $\frac{T}{2}\Delta\mathcal{Q}_{ij}$ are actually the entries of the *modularity* matrix $B = (b_{ij})$, $b_{ij} = a_{ij} - \frac{1}{T}wdeg(i)wdeg(j) = \frac{T}{2}\Delta\mathcal{Q}_{ij}$. Since B has zero rowsums, some of the entries $\Delta\mathcal{Q}_{ij}$ are negative and some are positive. We select an independent set of edges, which is a set of edges none of which are adjacent in the graph i.e. no two distinct edges share a vertex, in fashion similar to Jones and Plassmann's parallel algorithm [3]. Namely, we select an edge $e = (i, j)$ for matching if it has locally maximal weight $\Delta\mathcal{Q}_{ij}$, that is, if $\Delta\mathcal{Q}_{ij}$ is larger that all other $\Delta\mathcal{Q}$'s corresponding to its neighboring edges (two edges are neighbors if they share a common vertex). Such a choice does not interfere with selection of other edges for matching which allows for easy parallelization. We merge aggregates essentially until we cannot maximize $\mathcal{Q}$ further.

We apply this algorithm recursively to generate a hierarchy of coarse graphs. We note though that our main contribution is not in the coarsening, but in refining a given coarse level embedding to lead to fine level graph embedding, which we describe now.

3. The two-level coarse-to-fine procedure

We now discuss the refinement step to embed G_ℓ given an embedding of $G_{\ell+1}$. For simplicity (but without loss of generality) we restrict the discussion to a two-level method, with a fine graph G and a coarse graph G_c related by π. We represent the embedding of G_c by an array of coordinates $\{\mathbf{x}^{\mathcal{A}}\}_{\mathcal{A}\in V(G_c)}$ indexed by the coarse vertices. The embedding of G is represented by an array of coordinates $\{\mathbf{x}_i\}_{i\in V(G)}$ indexed by the fine vertices.

Suppose we have coordinates $\mathbf{x}^{\mathcal{A}} = (x_k^{\mathcal{A}})_{k=1}^d \in \mathbb{R}^d$ for each coarse vertex $\mathcal{A}$. Then for every $\mathcal{A}$ we want to find coordinates $\mathbf{x}_i = (x_{i,k})_{k=1}^d$ for each $i \in \mathcal{A}$. To do this we compute a ball (d-ball in $\mathbb{R}^d$) of radius $r^{\mathcal{A}}$ around each $\mathbf{x}^{\mathcal{A}}$ and spread the $\mathbf{x}_i$ within. Ideally, these balls should not intersect, as well as be quickly computable so as to make the multilevel method feasible. A visual example of this approach is shown in Figure 1.

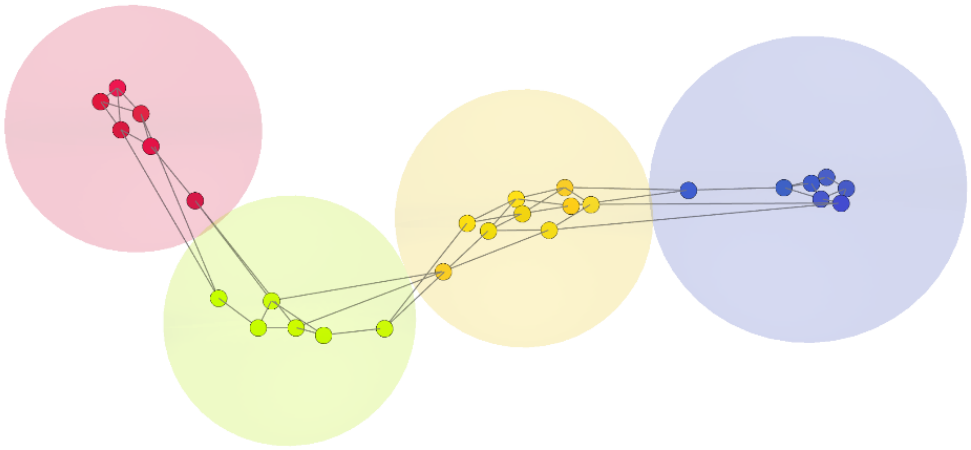


FIGURE 1. An example of embedding the fine layer via the embedded coarse layer. The coarse vertices are located at the center of the semi-transparent balls.

3.1. **Computing radii.** To construct balls around each coarse level vertex $\mathcal{A}$, we compute a simulation that continuously expands a ball around each aggregate's coordinates until it collides with another coarse vertex's ball. The balls stop expanding once they have collided with any other (expanding or non-expanding) ball. Each aggregate $\mathcal{A}$ may have its own rate of expansion $v_{\mathcal{A}}$ (for example, dependent on its size) though we use a constant rate of expansion across all aggregates for the implementation results discussed in this paper.

To compute this simulation we first we consider a new (dense) coarse-level graph whose vertex set is the same as G_c and whose edge weights $d_{\mathcal{AB}}$ are the distance between connected aggregates:

$$d_{\mathcal{AB}} = \begin{cases} \left\|\mathbf{x}^{\mathcal{A}} - \mathbf{x}^{\mathcal{B}}\right\|, & e_c = (\mathcal{A}, \mathcal{B}) \in E(G_c), \\ \infty, & \text{otherwise.} \end{cases}$$

Here $\|.\|$ is the standard Euclidean norm. Note that we consider the graph as dense, but the edges that are not originally present are heavily weighted which makes them non-essential in the following procedure. If desired, $\left\|\mathbf{x}^{\mathcal{A}} - \mathbf{x}^{\mathcal{B}}\right\|$ may be taken as the $d_{\mathcal{A},\mathcal{B}}$ for every $\mathcal{A}, \mathcal{B}$.

Taking this graph with positive weighted edges representing distances, we compute radii of the balls around each coarse vertex via the following described algorithm. The algorithm tracks the set of still-expanding balls, and uses a priority queue, which contains objects with associated priorities which allows inspection of the highest priority element and insertion and removal of elements ([14], p. 162), to determine the next collision between balls. The elements of the priority queue are edges each associated with the time of the potential collision between the balls of the vertices of the edge; in this case higher priorities correspond to sooner collision times.

If a coarse vertex's ball is still expanding it is *live*, otherwise it is *dead*. As balls collide they transition from live to dead. Each edge $(\mathcal{A}, \mathcal{B})$ in the queue can have its time of potential collision $t_{\mathcal{AB}}$ be in one of two cases (up to symmetry):

- If both $\mathcal{A}$ and $\mathcal{B}$ are live then the time for the two live balls to expand and cover the distance $d_{\mathcal{AB}}$ is $t_{\mathcal{AB}} = \frac{d_{\mathcal{AB}}}{v_{\mathcal{A}}+v_{\mathcal{B}}}$.
- If $\mathcal{A}$ is live and $\mathcal{B}$ is not live then the time for the single live ball to travel the distance $d_{\mathcal{AB}} - r^{\mathcal{B}}$ is $t_{\mathcal{AB}} = \frac{d_{\mathcal{AB}}-r^{\mathcal{B}}}{v_{\mathcal{A}}}$.

If both $\mathcal{A}$ and $\mathcal{B}$ are dead they will never collide at a later time. Initially all pairs are in the first case. The high-level idea behind the algorithm is that it continually takes the next collision between live balls and updates the times to collision for balls adjacent to the collision, repeating until no more collisions can occur (all coarse vertices are dead). The full algorithm to compute the simulation appears in Algorithm 1.

This algorithm finds a ball around all coarse vertices that have neighbors, so in particular the algorithm will always obtain balls for connected graphs with more than one vertex. Additionally, if non-edge pairs from the original coarse-level graph are given infinite distance between them in the coarse-level distance graph they will never collide, so these pairs can be ignored and the algorithm can exploit the sparsity of the original coarse-level graph.

The fact that the non-existent edges are ignored is what makes the requirement for the initial embedding to spread out sparsely connected aggregates vital. Two disconnected aggregates that are embedded close to each other will often have their balls intersect, which propagates to the finer levels. These edges may be considered (by taking $d_{\mathcal{AB}}$ to always be the distance between aggregates) but doing so is computationally infeasible for large graphs. One potential solution to this problem would be to use efficient k-nearest neighbor methods (for example, [16]) to locate geometrically close unconnected aggregates and use such information to ensure that their balls will not overlap.

3.2. **Parallel extension of the radii algorithm.** The algorithm described above is sequential. It can be implemented in both a *distributed-memory* and *shared-memory* parallel setting by computing the radii locally using the previous, coarser level, requiring the newly computed balls to be contained within the balls from the previous level. This allows the method to perform the simulation locally within each *coarser* aggregate.

Specifically, if $\mathfrak{A}$ is a coarser aggregate of the coarse vertices of G_c, then we run Algorithm 1 on coarse graph G_c restricted to the coarse vertices in $\mathfrak{A}$. with either only the original edges or all pairs of vertices. This gives us initial radii $r^{\mathcal{A}}$ for each $\mathcal{A} \in \mathfrak{A}$. We then perform rescaling to ensure that every one of the new balls lives within the larger ball computed around the coarser aggregate $\mathfrak{A}$ at the previous layer.

Recall the aggregates $\mathcal{A}$ with coordinates $\mathbf{x}^{\mathcal{A}}$ are embedded in the ball around the coarser aggregate $\mathfrak{A}$ with coordinate $\mathbf{x}^{\mathfrak{A}}$ and radius $r^{\mathfrak{A}}$. Now let

$$\alpha = \max_{\mathcal{A} \in \mathfrak{A}} \left\| \mathbf{x}^{\mathcal{A}} - \mathbf{x}^{\mathfrak{A}} \right\| + r^{\mathcal{A}}$$

Data: A coarse-level graph G_c with associated edge weights $d_{\mathcal{AB}}$ based on distances from a coarse-level embedding. Additionally, velocities $v_{\mathcal{A}}$ associated with each of the coarse vertices.
Result: A radius $r^{\mathcal{A}}$ associated with each coarse vertex $\mathcal{A}$ corresponding with the solution of the described simulation.
let Q be an empty priority queue;
for *each edge* $e_c = (\mathcal{A}, \mathcal{B})$ **do**
  insert e_c into Q with priority $\frac{d_{\mathcal{AB}}}{v_{\mathcal{A}}+v_{\mathcal{B}}}$;
end
while Q *is non-empty* **do**
  let $e_c = (\mathcal{A}, \mathcal{B}) \in Q$ be the element with highest priority;
  case *only* $\mathcal{A}$ *is live* **do**
    set $r^{\mathcal{A}} = d_{\mathcal{AB}} - r^{\mathcal{B}}$;
    set $\mathcal{A}$ to be dead;
    remove any pairs $(\mathcal{A}, \mathcal{C})$ where $\mathcal{C}$ is dead from the queue;
    remove and reinsert all pairs $(\mathcal{A}, \mathcal{C}) \in Q$ where $\mathcal{C}$ is alive with updated priority $t_{\mathcal{A},\mathcal{C}} = \frac{d_{\mathcal{AC}} - r^{\mathcal{A}}}{v_{\mathcal{C}}}$;
  end
  case *both* $\mathcal{A}$ *and* $\mathcal{B}$ *are live* **do**
    set $r^{\mathcal{A}} = \frac{d_{\mathcal{AB}}\ v_{\mathcal{A}}}{v_{\mathcal{A}}+v_{\mathcal{B}}}$ and $r^{\mathcal{B}} = \frac{d_{\mathcal{AB}}\ v_{\mathcal{B}}}{v_{\mathcal{A}}+v_{\mathcal{B}}}$;
    set both $\mathcal{A}$ and $\mathcal{B}$ to be dead;
    remove any pairs $(\mathcal{A}, \mathcal{C})$ where $\mathcal{C}$ is dead from the queue;
    remove any pairs $(\mathcal{B}, \mathcal{C})$ where $\mathcal{C}$ is dead from the queue;
    remove and reinsert all pairs $(\mathcal{A}, \mathcal{C}) \in Q$ where $\mathcal{C}$ is alive with updated priority $t_{\mathcal{A},\mathcal{C}} = \frac{d_{\mathcal{AC}} - r^{\mathcal{A}}}{v_{\mathcal{C}}}$;
    remove and reinsert all pairs $(\mathcal{B}, \mathcal{C}) \in Q$ where $\mathcal{C}$ is alive with updated priority $t_{\mathcal{B},\mathcal{C}} = \frac{d_{\mathcal{BC}} - r^{\mathcal{B}}}{v_{\mathcal{C}}}$;
  end
end

Algorithm 1: Algorithm for computing the simulation of expanding balls around aggregates.

where α is the distance between $\mathbf{x}^{\mathfrak{A}}$ and the farthest point on the balls of the aggregates $\mathcal{A}$, and could be smaller or larger than 1. We then update each $\mathbf{x}^{\mathcal{A}}$ to be

$$\mathbf{x}^{\mathfrak{A}} + r^{\mathfrak{A}} \frac{\mathbf{x}^{\mathcal{A}} - \mathbf{x}^{\mathfrak{A}}}{\alpha}$$

and update each $r^{\mathcal{A}}$ to be

$$r^{\mathfrak{A}} \frac{r^{\mathcal{A}}}{\alpha}$$

which is a rescaling of $\mathbf{x}^{\mathcal{A}}$ and the ball for $\mathcal{A}$ around $\mathbf{x}^{\mathfrak{A}}$ so that each newly computed ball is contained within the previous level's ball. Since there are no intersections between balls around the coarser aggregates on the previous level, the balls on the

current coarse level do not intersect. This procedure can be done locally for each coarser aggregate $\mathfrak{A}$ (i.e. it does not rely on more than one coarser aggregate) and so can be done in both a shared-memory or distributed-memory parallel setting.

Alternatively, when locally computing radii within the coarser aggregate $\mathfrak{A}$ any edges of the form $e_c = (\mathcal{A}, \mathcal{B})$ with $\mathcal{A} \in \mathfrak{A}$ and $\mathcal{B} \notin \mathfrak{A}$ can use the priority $t_{\mathcal{AB}} = \frac{r^{\mathfrak{A}}}{v_{\mathcal{A}}}$, which approximates the collision of $\mathcal{A}$ with $\mathcal{B}$ by the collision of $\mathcal{A}$ with the ball around $\mathfrak{A}$.

We note that currently our implementation is not distributed-memory parallel but is shared-memory parallel and does use the first local method described above (not the alternative). The time to compute the radii with the local method appears to be negligible compared with the time needed for the rest of the embedding algorithm, described in the next section.

4. Extending embedding algorithms to multilevel embedding algorithms

In this section, we first present our general approach to multilevel embedding, and then apply it to two specific settings. We continue with describing the two-level procedure for clarity, as the multilevel method is the iterative application of the two-level method between consecutive levels of the multilevel partitioning.

4.1. **Extending any given algorithm to a multilevel method.**

For any given graph embedding algorithm, referred to as the *base* graph embedding algorithm, we first embed the coarsest level of the hierarchy using the base algorithm. From here (in a recursive fashion through the hierarchy), one can embed a finer level from the next coarser one by running the given base embedding algorithm for each aggregate $\mathcal{A}$ on the finer graph restricted to the members of $\mathcal{A}$, obtaining "local" coordinates for each finer vertex $i \in \mathcal{A}$. This is the *local* method which gives the *local* embedding of the fine-level graph restricted to a single aggregate; when applied to the original, fine-level graph (or to a whole level) the base embedding algorithm is referred to as the *global* method. For each aggregate $\mathcal{A}$, once the local coordinates have been computed for each fine-level vertex, we translate the coordinates to be centered about the origin, scale them to be within $r^{\mathcal{A}}$ away from the origin, and finally translate them to be centered about $\mathbf{x}^{\mathcal{A}}$. After handling every aggregate, we obtain coordinates for every fine vertex and have embedded the finer layer. The process is then recursively applied to the next finer layer. Since the size of aggregates on each level can be made small if the partitioning was done with an appropriate coarsening factor, even a computationally expensive base embedding algorithm can be used locally. This process extends the given base embedding algorithm to a multilevel one.

Additionally, one can consider when embedding an aggregate $\mathcal{A}$ in the local embedding method (in addition to the coarse graph restricted to $\mathcal{A}$) all fine vertices adjacent to $\mathcal{A}$ (that is, the set $\{j \mid e = (i, j) \in E(G), i \in \mathcal{A}, j \notin \mathcal{A}\}$) as well as their edges crossing into $\mathcal{A}$ so as to capture boundary conditions and global embedding structure. The coordinates of the *external* vertices j (i.e. $j \notin \mathcal{A}$) can be approximated by the coordinate of the aggregate they belong to, so if $j \in \mathcal{B}$ then $\mathbf{x}_j \approx \mathbf{x}^{\mathcal{B}}$ is used by the local embedding method for $\mathcal{A}$. The local embedding method does not update

the coordinates of the external vertices. As with the radii, the application of the local embedding method can be done in both a distributed-memory or shared-memory parallel setting by applying it per aggregate. We refer to base embedding algorithms which use the approximate external information as semi-local methods.

Finally, if the base embedding algorithm is iterative in nature, then our multilevel method can be used as a preconditioner, providing better initial coordinates (than random) for applying the base embedding algorithm to the whole of the fine level graph as a global method. In this case, we refer to the base embedding algorithm being applied globally for some iterations during the multilevel method as global iterations. We note that these can also be done on the coarser graphs at each level of the hierarchy, and we still refer to them as global methods and global iterations. In contrast, the iterations which are applied on a per-aggregate basis are referred to as local iterations. The word 'global' will essentially refer to inspecting a graph without looking at the computed multilevel community structure, while locally will generally refer to inspecting aggregates of the community structure.

4.2. **Embedding via functional minimization.** In this section, we present a semi-local embedding algorithm that exploits functional minimization. It can be used as a *possibly expensive* base algorithm, which from our general approach described above can be used as a component for embedding in a multilevel fashion. We do not propose this method as an excellent standalone method, but present how it fits with our multilevel method and how the multilevel method helps aleviate some of its flaws. We describe the two-level version of the method. All coordinates are initialized randomly within the unit cube in $\mathbb{R}^d$.

We are trying to update the coordinate $\mathbf{x}_i$ where $i \in \mathcal{A}$. We consider the functional

$$\mathbf{J}_{loc}(\mathbf{t}) = \sum_{j:e=(i,j)} w_j \left\|\mathbf{t} - \mathbf{x}_j\right\|^2 + \sum_{j:j\in\mathcal{A},j\neq i} \frac{m_j}{\left\|\mathbf{t} - \mathbf{x}_j\right\|}$$

where $\|.\|$ is the standard Euclidean norm and $w_j, m_j > 0$ are chosen weights, such as the weight assigned to the edge $e = (i, j)$ or a chosen constant. The coordinates of vertices $j \in \mathcal{B} \neq \mathcal{A}$ where $e = (i, j)$ exists are approximated using $\mathbf{x}^{\mathcal{B}}$, as previously mentioned. The first term of 4.2 pulls connected vertices together while the second pushes all pairs of vertices apart.

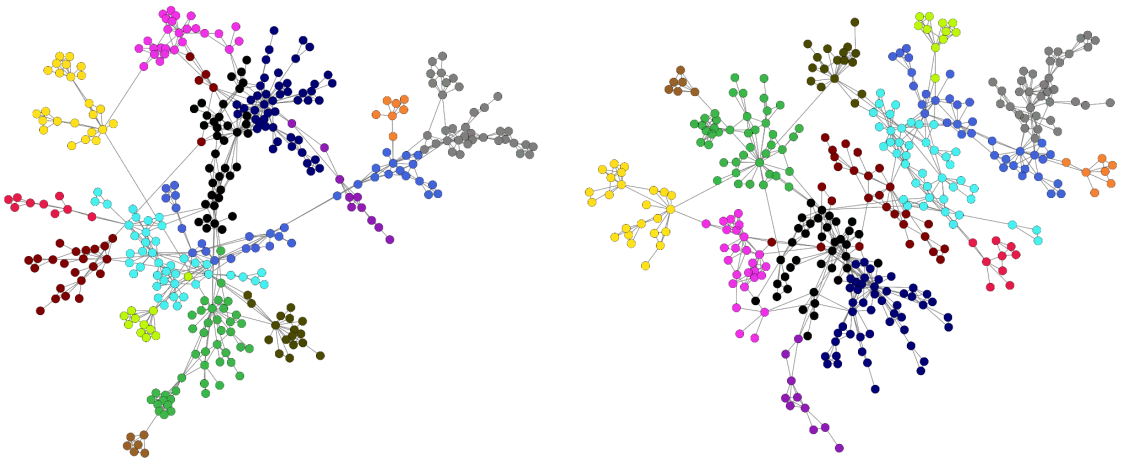
To find a better coordinate for vertex i we would like to take

$$\mathbf{x}_i = \min_{\mathbf{t}\in\mathbb{R}^d} \mathbf{J}_{loc}(\mathbf{t}).$$

However, a minimum over the whole space may be difficult to find. Therefore we employ an iterative process which only searches through one-dimensional subspaces.

We choose a finite set of points $\{\mathbf{p}\}$; these will be related to the directions in which $\mathbf{x}_i$ may update in. For example, in $\mathbb{R}^d$ these could be for example $\pm e_i$, where e_i is the i^{th} basis vector, the points of the regular d-simplex inscribed in the unit d-sphere, random unit vectors, or $\{\mathbf{x}_j \mid e = (i, j) \in E(G)\}$. Taking the previous coordinate of i, $\mathbf{x}_i$, for each $\mathbf{p}$ we examine the function

$$\mathbf{J}^{\mathbf{p}}_{loc}(t) = \sum_{j:e=(i,j)} w_j \left\|(1-t)\mathbf{x}_i + t\mathbf{p} - \mathbf{x}_j\right\|^2 + \sum_{j:j\in\mathcal{A},j\neq i} \frac{m_j}{\left\|(1-t)\mathbf{x}_i + t\mathbf{p} - \mathbf{x}_j\right\|}$$



(a) Minimization method applied globally to the fine graph (not using aggregates).

(b) Minimization method applied in a multilevel fashion.

FIGURE 2. Embeddings obtained via the minimization method. The same graph (`ca-netscience` [10]) is embedded for both images, and the colors represent the (coarsest) aggregates used and are the same between the two images. We let w_j be a *large* constant (≈ 10000) and $m_j = 1$, and performed 10 global iterations of the minimization method on each level, including the fine level. The points $\mathbf{p}$ were $\pm e_1, \pm e_2$. We ran the global method until convergence. Even on this small graph embedding, the unparallelized multilevel method is more than 10x faster at embedding. The multilevel method also removes the close geometrical association of non-connected communities which occurs mainly with the blue and light green aggregates, and appears to give a better use of the embedding space and simply provides a better structured embedding, thus correcting some deficiencies of the base embedding algorithm. Overall, we see that the less expensive multilevel method 2(b) with only a few global iterations achieves a better quality embedding than the expensive global method 2(a).

for $0 < t < 1$. This is Equation 4.2 such that $\mathbf{t}$ is in the open line segment $\{(1 - t)\mathbf{x}_i + t\mathbf{p} \mid 0 < t < 1\}$.

We find a local minimum $\mathbf{t}_\mathbf{p}$ for $\mathbf{J}_{loc}$ on each of these intervals and take $\mathbf{x}_i$ to be the minimum on $\mathbf{J}_{loc}$ from those. To embed the current level in the multilevel hierarchy, we iterate over each $\mathcal{A}$ and $i \in \mathcal{A}$, updating $\mathbf{x}_i$, until the change in $\mathbf{J}_{loc}(\mathbf{x}_i)$ is small.

We can either use this method as a global one and apply it to the whole graph (taking a two-level hierarchy with the coarse level containing only a single aggregate), or as described above it can be applied in a multilevel fashion, with just a few global iterations applied to each level. The results of the global method (i.e. no aggregates used) and the multilevel version with a small number of global iterations are in Figure 2(a) and Figure 2(b), respectively, one may see that the multilevel method corrects some problems that the global method has such as geometrically overlapping non-adjacent aggregates.

We now discuss the extension of an existing algorithm to a semi-local multilevel one by applying the proposed multilevel scheme described in Section 4.1 to the ForceAtlas2 algorithm.

4.3. **Extending the ForceAtlas2 Algorithm.** ForceAtlas2 is a physics-inspired force-directed algorithm for graph embedding in $\mathbb{R}^d$ designed for visualization purposes and incorporated into the Gephi network visualization software [8]. ForceAtlas2 relies on three forces: an attraction force pulling together connected vertices, a repelling force pushing apart all pairs of vertices, and a gravitational force pulling all vertices towards the origin of the embedding space. The forces are used to compute velocities for vertices and the vertices are moved iteratively until only a small change is observed. We present a semi-local extension of the algorithm below which adds a fourth force that factors in external information.

We present the method in a two-level fashion as in the previous section. All coordinates are initialized randomly, and we assume throughout the iterative process that all coordinates $\mathbf{x}_i$ are distinct and non-zero. Let $\mathbf{dir}(i,j) = \frac{\mathbf{x}_j - \mathbf{x}_i}{\|\mathbf{x}_j - \mathbf{x}_i\|}$ be the unit vector from $\mathbf{x}_i$ pointing towards $\mathbf{x}_j$ (the direction from $\mathbf{x}_i$ towards $\mathbf{x}_j$).

The attraction force pulling i towards j is given by

$$\mathbf{F}_{attract}(i,j) = k_{attract}\, a_{ij}\, \|\mathbf{x}_j - \mathbf{x}_i\|\, \mathbf{dir}(i,j) \tag{1}$$

where $k_{attract} > 0$ is a constant. Note that if (i,j) is not an edge then $a_{ij} = 0$ and so this force is similarly 0.

The repulsion force pushing i away from j is given by

$$\mathbf{F}_{repel}(i,j) = -k_{repel}\, \frac{(wdeg(i)+1)(wdeg(j)+1)}{\|\mathbf{x}_j - \mathbf{x}_i\|^2}\, \mathbf{dir}(i,j) \tag{2}$$

where $k_{repel} > 0$ is a constant.

The force of gravity for vertex i is

$$\mathbf{F}_{gravity}(i) = -k_{gravity}\, (wdeg(i)+1)\, \frac{\mathbf{x}_i}{\|\mathbf{x}_i\|} \tag{3}$$

where $k_{gravity} > 0$ is a constant.

The semi-local component, which we call the orientation force, for vertex $i \in \mathcal{A}$ and external vertex $j \in \mathcal{B}$ is

$$\mathbf{F}_{orient}(i,j) = k_{orient}\, \frac{1}{\|\mathbf{x}_i\|}\, \mathbf{dir}(\mathcal{A},\mathcal{B}) \tag{4}$$

where k_{orient} is a constant. This force orients the local embedding of the aggregate to match the global geometric structure of the embedding. We scale by $\frac{1}{\|\mathbf{x}_i\|}$ so that vertices are not pulled too far away from the origin in the case that $\mathbf{F}_{orient}(i,j)$ is a dominant force on i; the further $\mathbf{x}_i$ is from the origin the less dominant this force becomes. We note that when we apply ForceAtlas2 as a global method then this force does not exist (not used), but when it is applied as a local method this force is used.

The total force acting on vertex i, $\mathbf{F}_i$, is the sum of the forces in equations (1), (2), (3), and (4):

$$\mathbf{F}_i = \sum_{\substack{e=(i,j)\in E(G)\\ j\neq i\\ j\in\mathcal{A}}} \big[\mathbf{F}_{attract}(i,j) + \mathbf{F}_{repel}(i,j)\big] + \sum_{\substack{e=(i,j)\in E(G)\\ j\notin\mathcal{A}}} \big[\mathbf{F}_{orient}(i,j)\big] + \mathbf{F}_{gravity}(i)$$

We let $\mathbf{x} = (\mathbf{x}_i)$ and $\mathbf{F} = \mathbf{F}(\mathbf{x}) = (\mathbf{F}_i)$. In [8], the authors have proposed a method that effectively implements an explicit time-stepping scheme for driving the dynamical system $\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x})$ starting from some (random) initial position $\mathbf{x} = \mathbf{x}_{(0)}$, to a stationary point. As a result, once the forces $\mathbf{F}(\mathbf{x})$ for a given configuration $\mathbf{x} = (\mathbf{x}_i)$ are computed, the vertices are moved according to how the sum of the forces dictate. We note that a measure of convergence of ForceAtlas2 can be based on the sum of the norms of the forces, since when reaching stationary point they will be zero. That is, we use

$$\sqrt{\sum_i |\mathbf{F}_i|^2} \tag{5}$$

as smaller forces indicate the vertices are moving less. This measure will not necessarily monotonically decrease, however, as the ForceAtlas2 does not directly minimize this measurement, as opposed to the functional minimization procedure described previously. The above modified procedure combined with our multilevel framework is the multilevel ForceAtlas2 algorithm.

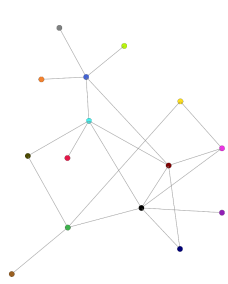
The ForceAtlas2 procedure appears to obtain a high-quality result for with relatively few iterations for small graphs. Since the size of aggregates are small (and in fact, coarsening methods often allow for choosing the approximate size of aggregates via choosing the coarsening factor) they require only a few iterations of the semi-local ForceAtlas2 algorithm to obtain quality results for the local embeddings, whereas applying ForceAtlas2 in a global manner takes many more.

Sample embeddings of the fine-level graph from the global ForceAtlas2 algorithm as well as both the multilevel version with global iterations applied at each level and without any global iterations applied at any level are shown in Figure 3(c), 3(f), and 3(i) respectively.

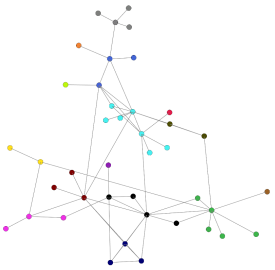
Figures 3(a), 3(b), and 3(c) show the results of the base ForceAtlas2 algorithm on the three different levels of the partitioning structure. Figures 3(d), 3(e), and 3(f) show the results of the multilevel ForceAtlas2 procedure with global iterations applied to each of the levels. Figures 3(g), 3(h), and 3(i) show the results of the multilevel ForceAtlas2 procedure without global iterations applied to each of the levels.

The visual quality of the multilevel embedding with global iterations is approximately the same as the original algorithm, though the global algorithm took many more iterations to create an embedding of the same visual quality. The high variance of the geometric densities of vertices in Figure 3(i) indicates why having different speeds for the radii computations may be useful: a faster rate of increase for larger aggregates would provide a larger ball around these aggregates, giving more room to spread their vertices around in. For visualization purposes this could give a higher quality embedding since the density of vertices in the space will have less variance.

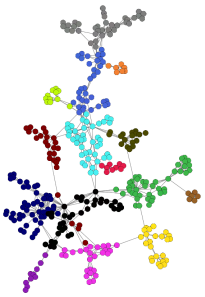
In Figure 4 we show the performance of the multilevel approach with no global iterations applied to a much larger graph on which the global ForceAtlas2 method becomes infeasible to run. We zoom in on the embedding structure to show that at each level, the communities are in fact embedded in a way that respects the structure of the graph and the global structure of the embedding. We note that the graph in Figure 4 is planar, and that a non-planar example is used in Figure 3. The global



(a) Global embedding of the coarsest level.

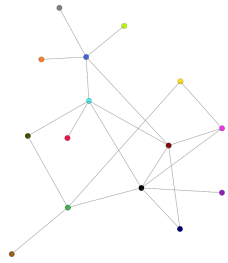


(b) Global embedding of the middle level.



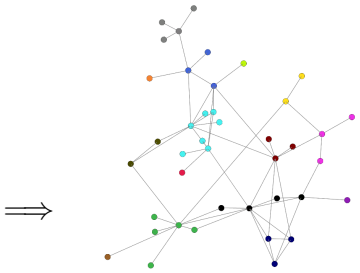
(c) Global embedding of the finest level.

||

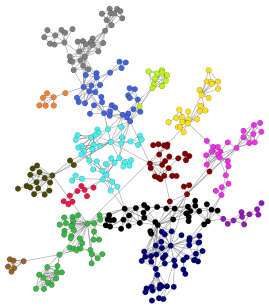


(d)

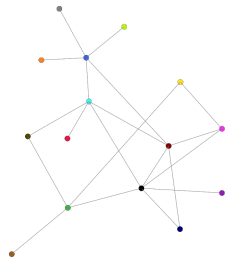
||



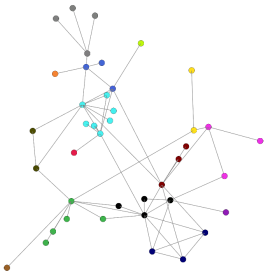
(e) Multilevel embedding of the middle level with global iterations applied on all levels.



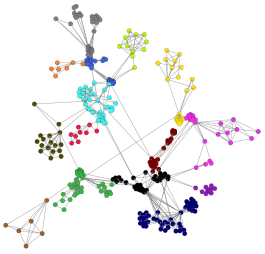
(f) Multilevel embedding of the finest level with global iterations applied on all level.



(g)



(h) Multilevel embedding of the middle level without global iterations applied on any level.



(i) Multilevel embedding of the finest level without global iterations applied on any level.

FIGURE 3. A comparison of the global ForceAtlas2 embedding with 1000 iterations, the multilevel procedure with 10 global iterations at each level and 10 local iterations for embedding each aggregate on each level, and the multilevel ForceAtlas2 embedding with 10 local iterations for embedding each aggregate on each level. The constants (k_{repel}, etc) of the ForceAtlas2 algorithm were set to 1. The same graph (`ca-netscience` [10]) is embedded as in Figures 2(a) and 2(b) with the color of the aggregates still matching. Each level of embedding is also shown.

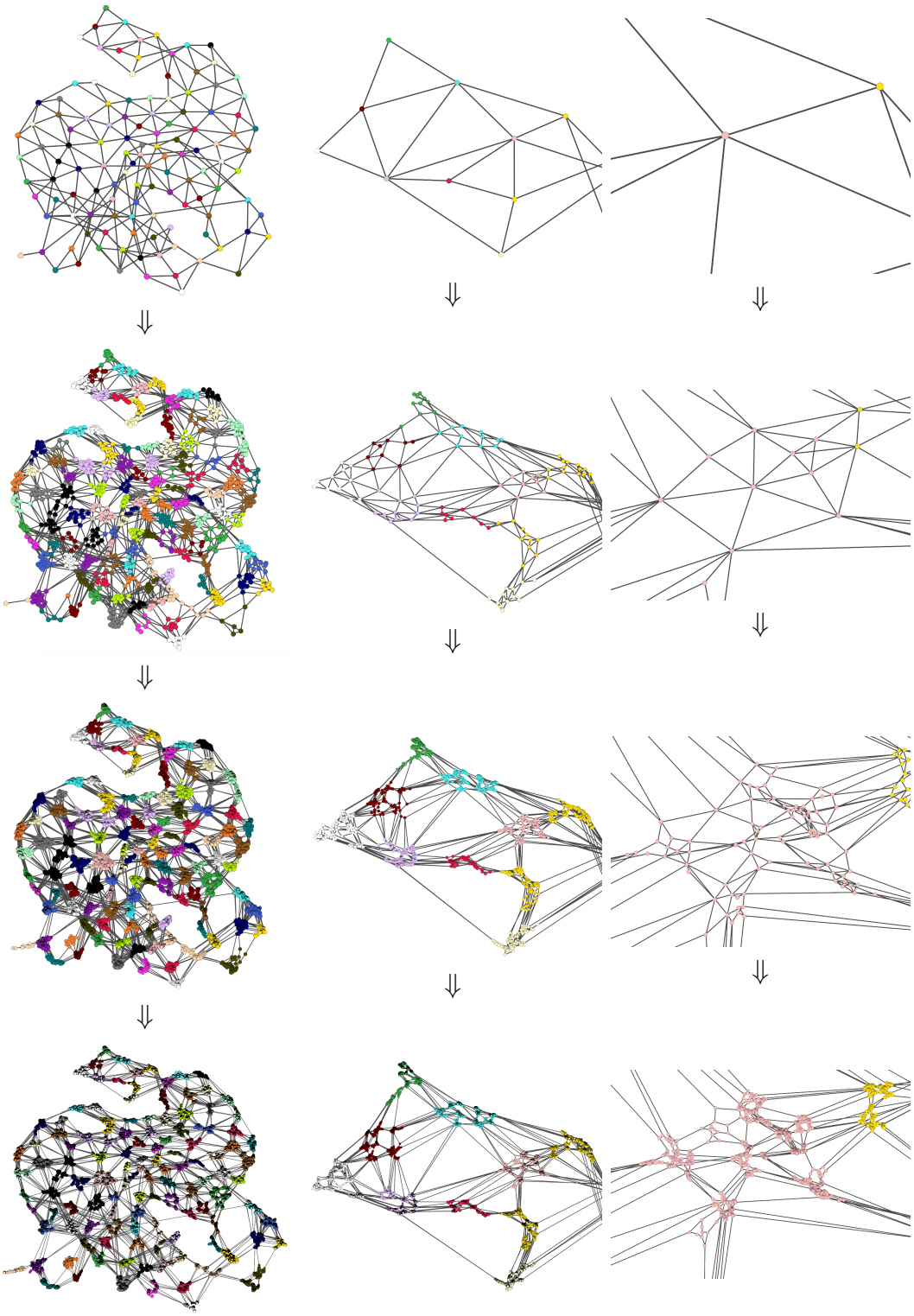


FIGURE 4. Zooming in on an embedding of `usroads` (126k vertices, 324k edges) [10] from the multilevel ForceAtlas2 with no global iterations. Each row corresponds to a different level in the hierarchy, and moving from left to right zooms in towards the left corner. Colors represent the coarsest aggregates, and due to the large number of aggregates some colors are duplicated.

ForceAtlas2 or multilevel ForceAtlas2 with a few global iterations, even just at coarser levels, would likely alleviate the clumping of vertices that the multilevel method alone gives. However, running the global ForceAtlas2 method to completion is computationally infeasible, and even just a few global iterations on the finest level proves very expensive, though applying the global iterations only to the coarser levels is feasible.

4.4. **Results.** To evaluate our embedding method, we inspect the speed-up from using our shared-memory parallel implementation over a sequential version and also provide some baseline partitioning times. We then evaluate the multilevel ForceAtlas2 method as a preconditioner for the global ForceAtlas2 method by inspecting the convergence of the ForceAtlas2 method on randomly initialized coordinates and coordinates initialized via our multilevel method. Table 1 provides information about the graphs tested.

We note that METIS does not reflect the state-of-the-art in fast partitioning; for more discussion see [15].

Dataset	$\|V\|$, $\|E\|$	number of levels	sequential partition time (seconds)
`ENZYMES-g479` [10]	28, 98	2	0
`mesh` [1]	100, 360	2	0
`ca-netscience` [10]	379, 1828	3	0
`Blog` [11]	10k, 330k	4	2.3
`usroads` [10]	126k, 324k	6	4.3
`Youtube` [11]	1.1M, 3M	6	145
`roadNet-CA` [10]	2M, 2.8M	7	106
`socfb-B-anon` [10]	3M, 21M	7	1364
`soc-livejournal` [10]	4M, 28M	7	1358
`soc-orkut` [10]	3M, 106M	7	3783

TABLE 1. Data for the graphs used, including the number and size of levels along with the partitioning time. All graphs are partitioned with a coarsening factor of 0.1 using METIS (sequential) as a baseline for partitioning timing. METIS allows the construction of a multilevel hierarchy with a consistent coarsening factor between the levels of the hierarchy.

[1]The vertices and edges of a 9×9 grid of squares.

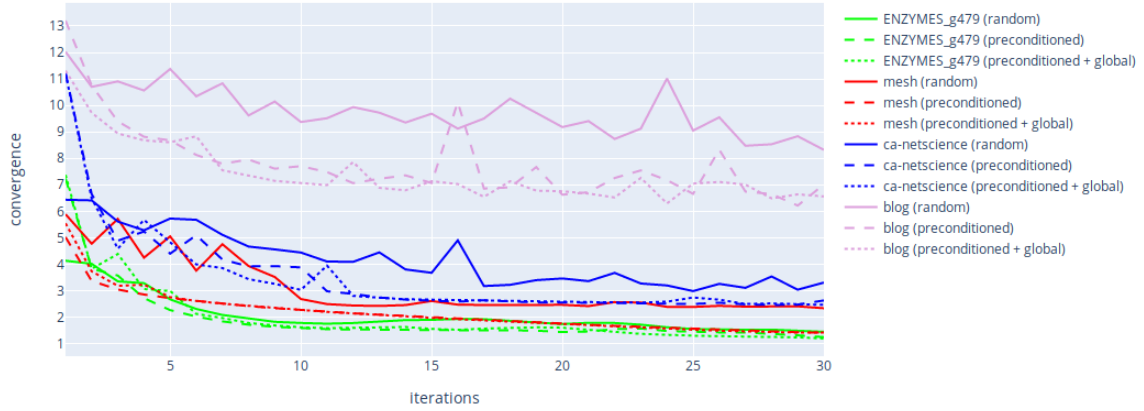
Dataset	sequential embedding time (seconds)	shared-memory parallel (36 threads) embedding time (seconds)	Speed-up from parallelization	MILE embedding time [12] (seconds)	Speed-up over MILE
`Blog`	2	1	(2x)	10.2	(10x)
`usroads`	4	2	(2x)	-	-
`Youtube`	212	54	(4x)	1153.2	(21.35x)
`roadNet-CA`	157	6	(26x)	-	-
`socfb-B-anon`	446	26	(17x)	-	-
`soc-livejournal`	452	25	(18x)	-	-
`soc-orkut`	841	48	(18x)	-	-

TABLE 2. Sequential and shared-memory parallel running times for the multilevel ForceAtlas2 embedding method. All constants (k_{repel}, etc.) were set to 1, and we used $d = 2$. The multilevel ForceAtlas2 method was performed with 1000 iterations on the coarsest level, 10 local iterations were used to embed each aggregate on each level and no global iterations on any level (other than the coarsest) were performed.

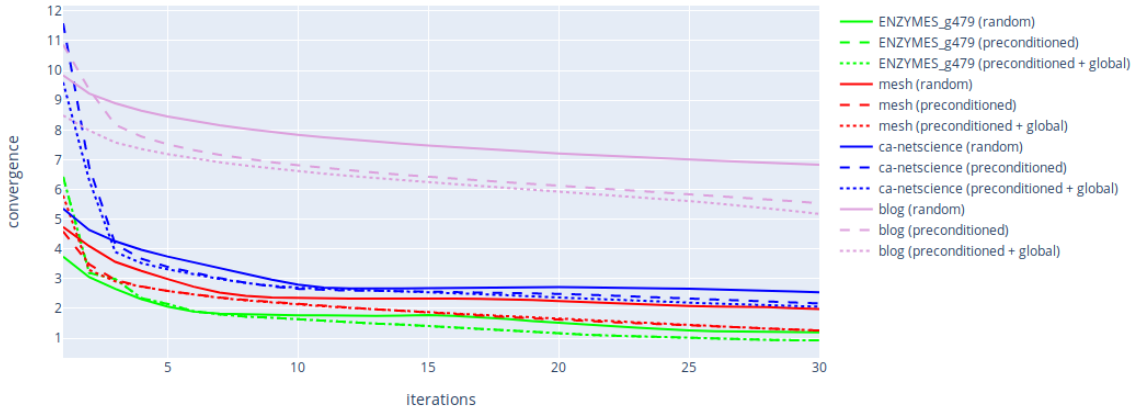
Table 2 provides timing information associated with the multilevel ForceAtlas2 method. The time it takes for the algorithm to complete, even sequentially, is very promising for the general method we propose. Compared to [12] (parallelized, 32 threads), which used the `Blog` (PPI in [12]) and `Youtube` graphs, our parallelized method achieves a 10-20x speed-up. The authors of [12] also tested on a graph with 8M vertices and 40M edges, which they embedded in "less than 3 hrs". The `Orkut` graph has a much larger edge set than this graph, which we were able to embed in less than a minute with the parallel implementation. With a state-of-the-art partitioner the total time to embed should remain less than 5 minutes. We note that the user also has a great deal of control over the running time (which is roughly correlated with embedding quality) since the user can specify exactly how many ForceAtlas2 iterations (both local and global) are used at every level. We also note that increasing the dimension should scale the running time by only a constant factor, and that the global ForceAtlas2 method is not feasible to run for the larger graphs.

For most graphs almost all of the time spent in the embedding procedure was used in the embedding of the final level. This means that to improve the quality of the embedding with experiencing significant slowdown one may do many more local iterations on the coarser levels, and perhaps also global iterations on the most coarse levels. Parallelization achieves decent speed up for the number of threads it uses as both ForceAtlas2 and the radii computations exploit shared-memeory parallelization well, although some components of these methods cannot be parallelized. Another attractive feature of our approach is that the benefits of parallelization appear to grow with the size of the input graph.

As stated previously, the time to compute the radii is very fast. For example, even with an asymptotically suboptimal implementation, on the finest level of the `orkut` graph the time to compute the radii was 1.25 seconds while the time to embed the finest level was 44.4 seconds (total embedding time was 48 seconds). This further gives evidence that our proposed multilevel method may work well for any given embedding algorithm; even extremely cheap methods will likely not have their embedding times dominated by the radii computations.



(a) Convergence results for $d = 2$.



(b) Convergence results for $d = 3$.

FIGURE 5. We plot the $\log_{10}$ of convergence value ($\log_{10}(\sqrt{\sum_i |\mathbf{F}_i|^2})$), for the global ForceAtlas2 method using random and multilevel ForceAtlas2 initialized coordinates on various graphs. The randomly initialized coordinates were scaled to be within the ball of radius $|V|$ around the origin (as opposed to the unit ball), and similarly for the coordinates initialized via the multilevel ForceAtlas methods. The Figure 5(a) corresponds to the methods run with dimension $d = 2$, and Figure 5(b) corresponds to the methods run with dimension $d = 3$. We tested randomly initialized coordinates ("random"), the multilevel ForceAtlas2 method with no global iterations ("preconditioned"), and the multilevel ForceAtlas2 method with 10 global iterations applied to every level except for the original fine graph ("preconditioned + global").

We also inspect the degree to which are method serves as a preconditioner for ForceAtlas2. Such a metric provides evidence that our embedding method approximates the globally high-quality embedding ForceAtlas2 produces (even though it does not directly attempt to mimick ForceAtlas2 globally), and in general using a multilevel method as a preconditioner serves as a method to evaluate a community-based embedding: since a community-based embedding should not only try to embed communities well locally but also have some global coherence, comparison using a method which obtains high-quality results and does both of these (though in an expensive manner) proves beneficial. For applications other than visualization, which

is the main target of the ForceAtlas2 procedure, a similar method could be used, but with a global method designed for the respective applications. For example, a global procedure which creates high-quality coordinates for classification tasks could be used.

Figure 5 provides a plot of the convergence of the global ForceAtlas2 method applied to a random initialization of coordinates, initialization of coordinates with our multilevel ForceAtlas2 method, and initialization of coordinates with our multilevel ForceAtlas2 method with global iterations, for embedding in both 2 and 3 dimensions. We measure the degree to which the global ForceAtlas2 method has converged by the measure in Equation (5). In both cases, our multilevel method served as a good preconditioner, improving the convergence by a about an order of magnitude for $d = 2$ and slightly less than an order of magnitude for $d = 3$. Additionally, the improvement appears to grow as the size of the graph does, which is expected as our method provides a good approximation to the global structure and as the input graph grows large, it becomes increasingly difficult for the global ForceAtlas2 method to 'sort out' the locations of the vertices relative to each other. The global iterations provided almost no benefit for small graphs, though did some benefit for the larger graphs, decreasing the variance in the convergence when $d = 2$ and slightly improving the convergence over the plain multilevel method for both $d = 2$ and $d = 3$. These results signal that our method approximates global structure well and provides a good estimate for visualization purposes.

5. Conclusions and outlook

In this paper, we demonstrated that by utilizing multilevel coarsening of sparse large-scale graphs that respects community structures at every coarsening level, we can make any available embedding algorithm more scalable and computationally feasible, by applying it locally at every level. We demonstrated the feasibility of this approach to the base algorithm of [8], as well as a simple functional minimization embedding procedure. We achieved a significant speed-up with our shared-memory parallel implementation over the recent paper [12], which could be extended to a distributed-memory parallel implementation. Additionally, our method provides many options, especially for iterative base algorithms, which can be fine-tuned for visualization or other purposes in a future update to our implementation. Developing more techniques to help improve the quality of embeddings, such as eliminating the "clumping" as seen in Figures 3(i) and 4, is a clear next step. Two potential remedies include considering overlapping communities, such as in [17], in the coarsening so as to place each fine vertex in a location that approximates their location within each aggregate they belong to, and to examine each level of the multilevel hierarchy multiple times in the spirit of the "V-cycles" of the Algebraic Multigrid method [5] to adjust the location of vertices within the global embedding (as opposed to the local embedding around an aggregate). We stress agian, that the main goal of the current paper was to demonstrate the feasibility of the proposed multilevel approach which, as we have shown, is promising.

Our code is available under a GNU GPL 2.1 license at github.com/LLNL/graph-embed.

Acknowledgment

This work was part of a summer project performed by the first author as summer intern at LLNL in 2018 and 2019.

References

[1] G. Karypis, G. and V. Kumar, "*A fast and high quality multilevel scheme for partitioning irregular graphs,*" SIAM Journal on Scientific Computing **20**(1)(1999), pp. 359–392.

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

[3] M. T. Jones and P. E. Plassmann, "*A Parallel Graph Coloring Heuristic,*" SIAM Journal on Scientific Computing, **14**(3), 1993, pp. 654–669.

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

[5] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, J. Ruge, *Adaptive Smoothed Aggregation αSA Multigrid*, SIAM Review, Vol. 47, pp. 317–346 (2005).

[6] Y. Notay, *An Aggregation-Based Algebraic Multigrid Method*, Electronic Transactions on Numerical Analysis, Vol. 37, pp. 123–146 (2010).

[7] P. D'Ambra, S. Filippone, 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, `dl.acm.org/citation.cfm?doid=3233179.3190647`.

[8] M. Jacomy, T. Venturini, S. Heymann, M. Bastian, "*ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software.*" 2014. PLoS ONE 9(6): e98679. `https://doi.org/10.1371/journal.pone.0098679`

[9] V. D. Blondel, J. Guillaume, R. Lambiotte, E. Lefebvre, "*Fast unfolding of communities in large networks.*" 2008. Journal of Statistical Mechanics: Theory and Experiment. `arxiv.org/abs/0803.0476`

[10] R. A. Rossi and N. K. Ahmed, "*The Network Data Repository with Interactive Graph Analytics and Visualization.*" Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence. `networkrepository.com`

[11] R. Zafarani and H. Liu, Social Computing Data Repository at ASU (2009). Tempe, AZ: Arizona State University, School of Computing, Informatics and Decision Systems Engineering. `socialcomputing.asu.edu`.

[12] J. Liang, S. Gurukar, and S. Parthasarathy, "*A Generic Approach to Scale Graph Embedding Methods.*" `www.kdd.org/kdd2018/files/deep-learning-day/DLDay18_paper_25.pdf`

[13] B. Quiring and P. S. Vassilevski, "*Properties of the Graph Modularity Matrix and its Applications,*" Lawrence Livermore National Laboratory Technical Report LLNL-TR-779424, June 26, 2019.

[14] T. Cormen, et al, "*Introduction to Algorithms, Third Edition.*" 2009, MIT Press.

[15] S. Ghosh, et al. "*Distributed Louvain Algorithm for Graph Community Detection*". 2018. IEEE International Parallel and Distributed Processing Symposium.

[16] M. Parwary, et al. "*PANDA: Extreme Scale Parallel K-Nearest Neighbor on Distributed Architectures*". 2016. IEEE International Parallel and Distributed Processing Symposium.

[17] Sharon, E., Brandt, A. and Basri, R., 2000, June. Fast multiscale image segmentation. In Proceedings IEEE Conference on Computer Vision and Pattern Recognition. IEEE CVPR 2000 (Vol. 1, pp. 70-77).

† Computer Science Depatrtmemnt, Northastern University, Boston, MA 02115, U.S.A., Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551, U.S.A.

Email address: `quiring.b@northeastern.edu, vassilevski1@llnl.gov`