Machine Learning (2023) 112:1663–1692
https://doi.org/10.1007/s10994-022-06285-7

Local2Global: a distributed approach for scaling representation learning on graphs

Lucas G. S. Jeub[1] · **Giovanni Colavizza**[1,2] · **Xiaowen Dong**[3] · **Marya Bazzi**[1,4] · **Mihai Cucuringu**[1,5]

Received: 30 January 2022 / Revised: 19 September 2022 / Accepted: 17 November 2022 /
Published online: 24 February 2023

Abstract
We propose a decentralised "*local2global*" approach to graph representation learning, that one can a-priori use to scale any embedding technique. Our *local2global* approach proceeds by first dividing the input graph into overlapping subgraphs (or "*patches*") and training local representations for each patch independently. In a second step, we combine the local representations into a globally consistent representation by estimating the set of rigid motions that best align the local representations using information from the patch overlaps, via group synchronization. A key distinguishing feature of *local2global* relative to existing work is that patches are trained independently without the need for the often costly parameter synchronization during distributed training. This allows *local2global* to scale to large-scale industrial applications, where the input graph may not even fit into memory and may be stored in a distributed manner. We apply *local2global* on data sets of different sizes and show that our approach achieves a good trade-off between scale and accuracy on edge reconstruction and semi-supervised classification. We also consider the downstream task of anomaly detection and show how one can use *local2global* to highlight anomalies in cybersecurity networks.

Keywords Scalable graph embedding · Distributed training · Group synchronization

1 Introduction

The application of deep learning on graphs, or Graph Neural Networks (GNNs), has recently become popular due to their ability to perform well on a variety of tasks on non-Euclidean graph data. Common tasks for graph-based learning include link prediction (i.e., predicting missing edges based on observed edges and node features), semi-supervised node classification (i.e., classifying nodes based on a limited set of labeled nodes, node features, and edges), and unsupervised representation learning (i.e., embedding nodes in

Editor: Andrea Passerini.

✉ Lucas G. S. Jeub
lucasjeub@gmail.com

Extended author information available on the last page of the article

a low-dimensional space $\mathbb{R}^d$, with d much smaller than the number of nodes in the graph). Among the significant open challenges in this area of research is the question of scalability both during training and inference. Cornerstone techniques such as Graph Convolutional Networks (GCNs) (Kipf & Welling, 2017) make the training dependent on the neighborhood of any given node. Since in many real-world graphs the number of neighbors grows exponentially with the number of hops taken, the scalability of such methods is a significant challenge. In recent years, several techniques have been proposed to render GNNs more scalable, including layer-wise sampling (Hamilton et al., 2018) and subgraph sampling (Chiang et al., 2019) approaches (see Sect. 2).

We contribute to this line of work by proposing a decentralized divide-and-conquer approach to improve the scalability of network embedding techniques. While our work focuses on node embeddings, our proposed methodology can be naturally extended to edge embeddings. Our "*local2global*" approach proceeds by first dividing the network into overlapping subgraphs (or "patches") and training separate local node embeddings for each patch (local in the sense that each patch is embedded into its own local coordinate system). The resulting local patch node embeddings are then transformed into a global node embedding (i.e. all nodes embedded into a single global coordinate system) by estimating a rigid motion applied to each patch using the As-Synchronized-As-Possible (ASAP) algorithm (Cucuringu et al., 2012a, b). A key distinguishing feature of this "decentralised" approach is that we can train the different patch embeddings separately and independently, without the need to keep parameters synchronised. The benefit of *local2global* is fourfold: • (1) it is highly parallelisable as each patch is trained independently; • (2) it can be used in privacy-preserving applications and federated learning setups, where frequent communication between devices is often a limiting factor (Kairouz & McMahan, 2021), or "decentralized" organizations, where one needs to simultaneously consider data sets from different departments; • (3) it can reflect varying structure across a graph through asynchronous parameter learning; and • (4) it is a generic approach that, in contrast to most existing approaches, can be applied to a large variety of embedding techniques.

We assess the performance of the resulting embeddings on three tasks: one edge-level task, i.e., edge reconstruction, and two node-level tasks, i.e., semi-supervised classification and anomaly detection. In the edge reconstruction task, we use the learned node embeddings to reconstruct known edges in the network, verifying that the embedding captures the network information. In semi-supervised node classification, one has a few labeled examples and a large amount of unlabeled data, and the goal is to classify the unlabeled data based on the few labeled examples and any structure of the unlabeled data. Here we propose to first train an embedding for the network in an unsupervised manner and then train a classifier on the embedding to predict the labels. Finally, as an application we have applied our method to the problem of anomaly detection where a low-dimensional representation that denoises and preserves the intrinsic structural properties of the data is essential.

The remainder of this paper is structured as follows. We give a detailed overview of related work in Sect. 2 and describe our methodology in Sect. 3. We show and interpret the results of numerical experiments on a variety of tasks and data sets in Sects. 4 and 5. We conclude and offer directions for future work in Sect. 6. Our reference implementations of the *local2global* patch alignment method[1] and the distributed training of graph embeddings[2] are available from GitHub. This paper extends preliminary proof-of-concept results

[1] https://github.com/LJeub/Local2Global.
[2] https://github.com/LJeub/Local2Global_embedding.

that were published as an extended abstract at the KDD 2021 "Deep Learning on Graphs" workshop (Jeub et al., 2021).

2 Related work

This section briefly surveys scalable techniques for GNNs, and describes challenges arising when considering large scale graphs. Consider a graph $G(V, E)$ with $n = |V|$ nodes and $m = |E|$ edges. Let $\boldsymbol{A}$ be the adjacency matrix of G, where

$$A_{ij} = \begin{cases} 1, & (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$$

then a general GNN layer is a differentiable function $\boldsymbol{H} = \mathrm{GNN}(\boldsymbol{X};\boldsymbol{A})$, where $\boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{in}}}$ are the input node features and $\boldsymbol{H} \in \mathbb{R}^{n \times d_{\text{out}}}$ are the output node features. Let $\boldsymbol{x}_i$ be the ith row of $\boldsymbol{X}$ and similarly let $\boldsymbol{h}_i$ be the ith row of $\boldsymbol{H}$. Most GNN layers can be written as

$$\boldsymbol{h}_i = \phi\left(\boldsymbol{x}_i, \bigoplus_{j=1}^{n} C(\boldsymbol{A})_{ij} \psi(\boldsymbol{x}_i, \boldsymbol{x}_j)\right), \tag{1}$$

where ϕ and ψ are differentiable, vector-valued functions (typically with trainable parameters), $C(\boldsymbol{A})$ is the convolution matrix,[3] and $\bigoplus$ is a permutation-invariant aggregator function (such as sum, max, ...). The formulation in Eq. (1) is often referred to as message passing where $\psi(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the message from node j to node i. Many different architectures have been proposed following this general framework (see Wu et al. (2021) for a recent survey).

A particularly simple architecture which still has competitive performance (Shchur et al., 2019) is GCN (Kipf & Welling, 2017). Here, $C(\boldsymbol{A})$ is the normalised adjacency matrix with self-loops, i.e.,

$$C(\boldsymbol{A}) = \bar{\boldsymbol{A}} = \tilde{\boldsymbol{D}}^{-\frac{1}{2}} \tilde{\boldsymbol{A}} \tilde{\boldsymbol{D}}^{-\frac{1}{2}},$$

where $\tilde{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ and $\tilde{\boldsymbol{D}}$ is the diagonal matrix with entries $\tilde{\boldsymbol{D}}_{ii} = \sum_j \tilde{\boldsymbol{A}}_{ij}$. Further, ψ is a linear function, i.e., $\psi(x_j) = x_j^T \boldsymbol{W}$ where $\boldsymbol{W} \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ is a learnable parameter matrix. The GCN layer can be written in matrix form as

$$H = \mathrm{GCN}(\boldsymbol{X};\boldsymbol{A}) = \sigma\left(\bar{\boldsymbol{A}}\boldsymbol{X}\boldsymbol{W}\right), \tag{2}$$

where σ is an element-wise non-linearity (typically ReLU).

The key scalability problems for GNNs only concern deeper architectures where we have l nested GNN layers (Chen et al., 2022), i.e.,

$$H = \underbrace{\mathrm{GNN}(\mathrm{GNN}(\ldots;\boldsymbol{A});\boldsymbol{A})}_{l}. \tag{3}$$

In particular, a single-layer GNN is easy to train in a scalable manner using mini-batch stochastic gradient descent (SGD). For simplicity, assume that we have a fixed feature dimension d, i.e., $d_{\text{in}} = d_{\text{out}} = d$ for all layers. The original GCN paper of Kipf and Welling (2017) uses full-batch gradient descent to train the model which entails the computation of

[3] Note that $C(\boldsymbol{A})$ should satisfy $C(\boldsymbol{P}\boldsymbol{A}\boldsymbol{P}^T) = \boldsymbol{P}C(\boldsymbol{A})\boldsymbol{P}^T$ for any $n \times n$ permutation matrix $\boldsymbol{P}$.

the gradient for all nodes before updating the model parameters. This is efficient in terms of time complexity per epoch $O(lmd + lnd^2)$, where n is the number of nodes and m is the number of edges. However, it requires storing all the intermediate embeddings and thus has memory complexity $O(lnd + ld^2)$. Further, as there is only a single parameter update per epoch, convergence tends to be slow.

The problem with applying vanilla mini-batch SGD (where we only compute the gradient for a sample of nodes, i.e., the batch) to a deep GNN model is that the embedding of the nodes in the final layer depends on the embedding of all the neighbours of the nodes in the previous layer and so on iteratively. Therefore, the time complexity for a single mini-batch update approaches that for a full-batch update as the number of layers increases, unless the network has disconnected components. There are mainly three families of methods (Chen et al., 2020; Chiang et al., 2019) that have been proposed to make mini-batch SGD training more efficient for GNNs.

Layer-wise sampling. The idea behind layer-wise sampling is to sample a set of nodes for each layer of the nested GNN model and compute the embedding for sampled nodes in a given layer only based on embeddings of sampled nodes in the previous layer, rather than considering all the neighbours as would be the case for vanilla SGD. This seems to have first been used by GraphSAGE (Hamilton et al., 2018), where a fixed number of neighbours is sampled for each node at each layer. However, this results in a computational complexity that is exponential in the number of layers and also redundant computations as the same intermediate nodes may be sampled starting from different nodes in the batch. Later methods avoid the exponential complexity by first sampling a fixed number of nodes for each layer either independently [FastGCN, Chen et al. (2018a)] or conditional on being connected to sampled nodes in the previous layer (LADIES, Zou et al. (2019)) and reusing embeddings. Both methods use importance sampling to correct for bias introduced by non-uniform node-sampling distributions. Also notable is Chen et al. (2018b), which uses variance reduction techniques to effectively train a GNN model using neighbourhood sampling as in GraphSAGE with only 2 neighbours per node. However, this is achieved by storing hidden embeddings for all nodes in all layers and thus has the same memory complexity as full-batch training.

Linear model. Linear models remove the non-linearities between the different GNN layers which means that the model can be expressed as a single-layer GNN with a more complicated convolution operator and hence trained efficiently using mini-batch SGD. Common choices for the convolution operator are powers of the normalised adjacency matrix (Wu et al., 2019) and variants of personalised Page-Rank (PPR) matrices (Busch et al., 2020; Chen et al., 2020; Bojchevski et al., 2020; Klicpera et al., 2019). Another variant of this approach is Frasca et al. (2020), which proposes combining different convolution operators in a wide rather than deep architecture. There are different variants of the linear model architecture, depending on whether the non-linear feature transformation is applied before or after the propagation (see Busch et al. (2020) for a discussion), leading to predict-propagate and propagate-predict architectures respectively. The advantage of the propagate-predict architecture is that one can pre-compute the propagated node features (e.g., using an efficient push-based algorithm as in Chen et al. (2020)) which can make training highly scalable. The disadvantage is that this will densify sparse features which can make training harder (Bojchevski et al., 2020). However, the results from Busch et al. (2020) suggest that there is usually not much difference in prediction performance between these options (or the combined architecture where trainable transformations are applied before and after propagation).

Subgraph sampling. Subgraph sampling techniques (Zeng et al., 2019; Chiang et al., 2019; Zeng et al., 2020) construct batches by sampling an induced subgraph of the full graph. In particular, for subgraph sampling methods, the sampled nodes in each layer of the model in a batch are the same. In practice, subgraph sampling seems to outperform layer-wise sampling (Chen et al., 2020). GraphSAINT (Zeng et al., 2020), which uses a random-walk sampler with an importance sampling correction similar to Chen et al. (2018a), Zou et al. (2019), seems to have the best performance so far. Our *local2global* approach shares similarities with subgraph sampling, most notably ClusterGCN (Chiang et al., 2019), which uses graph clustering techniques to sample the batches. The key distinguishing feature of our approach is that we train independent models for each patch (and then we stitch together the resulting embeddings), whereas for ClusterGCN, model parameters have to be kept synchronised for different batches, which hinders fully distributed training and its associated key benefits (see Sect. 1). Another recent method, [GNNAutoScale, Fey et al. (2021)], combines subgraph sampling with historical embeddings for the out-of-sample neighbors of nodes in the sampled subgraph. Historical embeddings avoid the loss of information inherent in subgraph sampling techniques, but require storing all intermediate embeddings for all nodes, resulting in a large memory footprint. While one can use slower memory for the historical embeddings without significantly impacting training speed, keeping historical embeddings synchronised will still result in significant communications overhead in a distributed setting.

3 The *local2global* algorithm

The key idea behind the *local2global* approach to graph embedding is to embed different parts of a graph independently by splitting the graph into overlapping "patches" and then stitching the patch node embeddings together to obtain a single global node embedding for each node. The stitching of the patch node embeddings proceeds by estimating the orthogonal transformations (i.e., rotations and reflections) and translations for the embedding patches that best aligns them based on the overlapping nodes.

Consider a graph $G(V, E)$ with node set V and edge set E. The input for the *local2global* algorithm is a patch graph $G_p(\mathcal{P}, E_p)$ together with a set of patch node embeddings $\mathcal{X} = \{\boldsymbol{X}^{(k)}\}_{k=1}^{p}$. The node set of the patch graph is the set of patches $\mathcal{P}$, where each node $P_k \in \mathcal{P}$ of the patch graph (i.e., a "patch") is a subset of V and has an associated node embedding $\boldsymbol{X}^{(k)} \in \mathbb{R}^{|P_k| \times d}$. We require that the set of patches $\mathcal{P} = \{P_k\}_{k=1}^{p}$ is a cover of the node set V (i.e., $\bigcup_{k=1}^{p} P_k = V$), and that the patch embeddings all have the same dimension d.

A pair of patches $\{P_i, P_j\}$ is connected, i.e., $\{P_i, P_j\} \in E_p$ if we have an estimate of the relative transformation between P_i and P_j. In particular, we require that a connected pair of patches satisfies the minimum overlap condition $\{P_i, P_j\} \in E_p \implies \left|P_i \cap P_j\right| \geq d + 1$ which ensures that we can estimate the relative orthogonal transformation.[4] We further assume that the patch graph is connected such that a global alignment is possible.[5] The best

[4] Note that a pair of patches P_i, P_j that satisfies the minimum overlap condition $\left|P_i \cap P_j\right| \geq d + 1$ is not necessarily connected in the patch graph, i.e., we do not necessarily estimate the relative transformations for all pairs of patches that satisfy the minimum overlap criterion.

[5] If the patch graph has multiple connected components, the aligned embedding for one component is independent of the aligned embeddings in other components.

way to construct the patch graph will depend on the application at hand. We describe two possible constructions in Sects. 4.2 and 5.2.

The *local2global* algorithm for aligning the patch embeddings proceeds in three stages and is an evolution of the approach in Cucuringu et al. (2012a, 2012b). We assume that each patch embedding $\boldsymbol{X}^{(k)}$ is a perturbed version of an underlying global node embedding $\boldsymbol{X}$, where the perturbation is composed of scaling ($\mathbb{R}_+$), reflection ($\mathbb{Z}_2$), rotation ($\mathrm{SO}(d)$), translation ($\mathbb{R}^d$), and noise. The goal is to estimate the transformation applied to each patch using only pairwise noisy measurements of the relative transformation for pairs of connected patches. In the first two stages, we estimate the scale and the orthogonal transformation to apply to each patch embedding, using a variant of the eigenvector synchronization method (Singer, 2011; Cucuringu et al., 2012a, b). Aligning the patches from the perspective of reflections amounts to associating, to each patch, a value of -1 (apply a reflection) or $+1$ (do not apply a reflection). Similarly, aligning the patches from the perspective of their rotations amounts to associating, to each patch, a rotation matrix (element of $\mathrm{O}(d)$). In the third stage, we estimate the patch translations by solving a least-squares problem. Aligning patches with respect to translations amounts to estimating, for each patch, the d-dimensional translation vector in $\mathbb{R}^d$. Note that unlike Cucuringu et al. (2012a, 2012b), we solve for translations at the patch level rather than solving a least-squares problem for the node coordinates. This means that the computational cost for computing the patch alignment is independent of the size of the original network and depends only on the amount of patch overlap, the number of patches and the embedding dimension.

Once such scaling, reflection, rotation, translations transformations are applied to each patch, all the patches should be synchronized in a globally consistent framework. In the current setting, we are essentially performing synchronization over the Euclidean group $\mathrm{Euc}(d) \cong O(d) \times \mathbb{R}^d$. The estimation procedure for scaling and orthogonal transformations are independent from each other and invariant under translations. However, the optimal translations depend on the other transformations and it is thus essential that the translation synchronization step is performed at the end (ie, synchronization over $\mathbb{R}^d$), after having performed the rotation synchronization in the first instance (ie, synchronization over $O(d) \cong \mathbb{Z}_2 \times SO(d)$). We illustrate the different steps of the *local2global* algorithm in Fig. 1.

3.1 Eigenvector synchronisation over scales

As an optional first step of the *local2global* algorithm, we synchronise the scales of the patch embeddings. Whether or not synchronisation over scales is beneficial depends on the properties of the chosen embedding method. In particular, if the embedding method determines the overall scale based on global properties of the graph (e.g., through normalisation), scale synchronisation is likely beneficial as one expects patch embeddings to have different scales. However, if the embedding method determines the overall scale only based on local properties (e.g., by trying to fix edge lengths), scale synchronisation may not be necessary and may hurt the final embedding quality by fitting noise. We assume that to each patch P_i there corresponds an unknown scale factor $s_i \in \mathbb{R}_+$. For each pair of connected patches $(P_i, P_j) \in E_p$, we have a noisy estimate $r_{ij} \approx s_i s_j^{-1}$ for the relative scale of the two patches. We estimate r_{ij} using the method from Horn et al. (1988) as

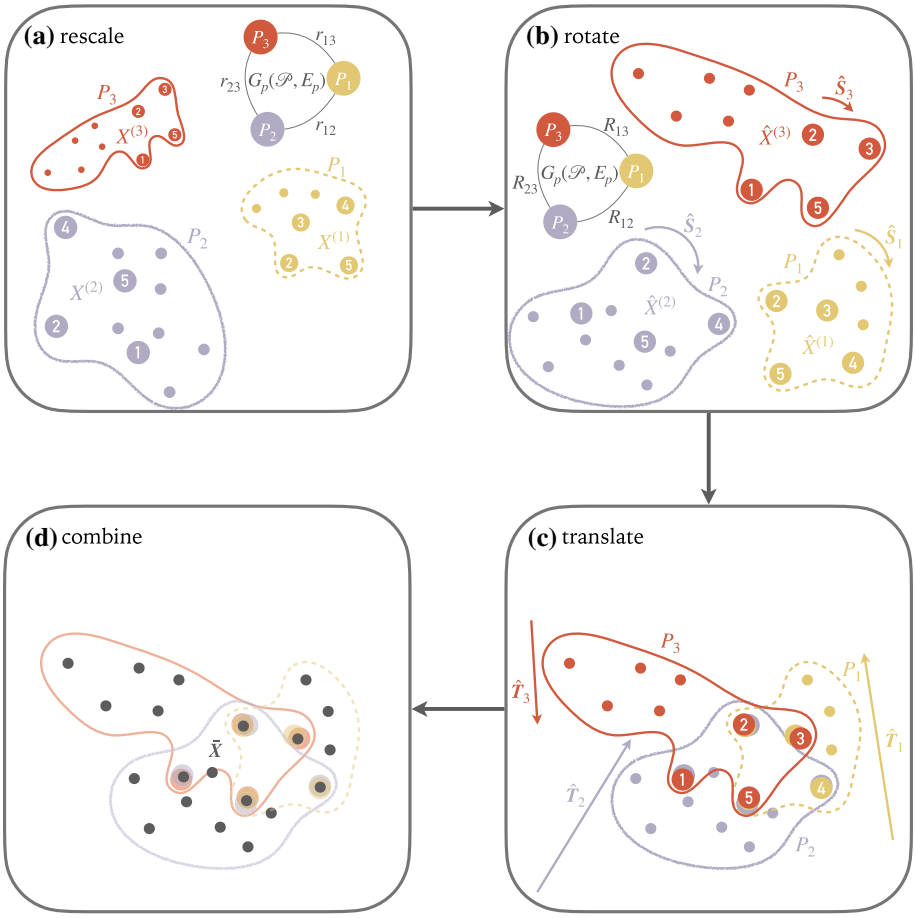

Fig. 1 Schematic overview of the different steps of the local2global algorithm. (**a**) Synchronisation over scales. (**b**) Synchronisation over orthogonal transformations. (**c**) Synchronisation over translations. (**d**) Global node embedding as centroid of aligned patch embeddings

$$r_{ij} = \frac{\sqrt{\sum_{u\in P_i\cap P_j} X_u^{(i)} - \frac{1}{|P_i\cap P_j|}\sum_{v\in P_i\cap P_j} X_v^{(i)}}}{\sqrt{\sum_{u\in P_i\cap P_j} X_u^{(j)} - \frac{1}{|P_i\cap P_j|}\sum_{v\in P_i\cap P_j} X_v^{(j)}}}. \tag{4}$$

For each pair of connected patches $(P_i, P_j) \in E_p$, we have a consistency equation $s_i \approx r_{ij}s_j$, as we can estimate the scale of a patch based on the scale of one of its neighbors in the patch graph and the relative scale between them. Using a weighted average to combine estimates based on different neighbors, we arrive at

$$s_i \approx \sum_j \frac{w_{ij}r_{ij}s_j}{\sum_j w_{ij}}, \tag{5}$$

where we set $w_{ij} = \left|P_i \cap P_j\right|$ if $\{P_i, P_j\} \in E_p$ and $w_{ij} = 0$ otherwise, as we expect pairs of patches with larger overlap to result in a more reliable estimate of the relative scale. Based on Eq. (5), we use the leading eigenvector $\hat{\boldsymbol{s}}$ of the matrix with entries

$$\frac{w_{ij} r_{ij}}{\sum_j w_{ij}},$$

(which has strictly positive entries by the Perron-Frobenius theorem, provided the patch graph is connected) to estimate the scale factors. We normalise $\hat{\boldsymbol{s}}$ such that $\frac{1}{p}\sum_j \hat{s}_j = 1$ to fix the global scale and compute the scale-synchronized patch embeddings as $\tilde{\boldsymbol{X}}^{(i)} = \boldsymbol{X}^{(i)}\hat{s}_i$.

3.2 Eigenvector synchronisation over orthogonal transformations

Synchronisation over orthogonal transformations proceeds in a similar way to synchronisation over scales. We assume that to each patch P_i, there corresponds an unknown group element $S_i \in O(d) \simeq Z_2 \times SO(d)$ (where $O(d)$ is the orthogonal group and $SO(d)$ is the group of rotations with elements represented by a $d \times d$ orthogonal matrices), and for each pair of connected patches $(P_i, P_j) \in E_p$ we have a noisy proxy for $S_i S_j^{-1}$, which is precisely the setup of the *group synchronization* problem.

For a pair of connected patches $P_i, P_j \in \mathcal{P}$ such that $\{P_i, P_j\} \in E_p$ we can estimate the relative rotation/reflection via a Procrustes alignment, by applying the method from Horn et al. (1988)[6] to their overlap as $\left|P_i \cap P_j\right| \geq d + 1$. Thus, we can construct a block matrix $\boldsymbol{R}$ where $\boldsymbol{R}_{ij}$ is the $d \times d$ orthogonal matrix representing the estimated relative transformation from patch P_j to patch P_i if $\{P_i, P_j\} \in E_p$ and $\boldsymbol{R}_{ij} = \boldsymbol{0}$ otherwise, such that $\boldsymbol{R}_{ij} \approx \boldsymbol{S}_i \boldsymbol{S}_j^T$ for connected patches.

In the noise-free case, we have the consistency equations $\boldsymbol{S}_i = \boldsymbol{R}_{ij}\boldsymbol{S}_j$ for all i, j such that $\{P_i, P_j\} \in E_p$. One can combine the consistency equations for all neighbours of a patch to arrive at

$$\boldsymbol{S}_i = \sum_j \boldsymbol{M}_{ij}\boldsymbol{S}_j, \qquad \boldsymbol{M}_{ij} = \frac{w_{ij}\boldsymbol{R}_{ij}}{\sum_j w_{ij}}, \tag{6}$$

where we use $w_{ij} = \left|P_i \cap P_j\right|$ to weight the contributions, as we expect a larger overlap to give a more robust estimate of the relative transformation. We can write Eq. (6) as $\boldsymbol{S} = \boldsymbol{M}\boldsymbol{S}$, where $\boldsymbol{S} = (\boldsymbol{S}_1, \ldots, \boldsymbol{S}_p)^T$ is a $pd \times d$ block-matrix and $\boldsymbol{M}$ is a $pd \times pd$ block-matrix. Thus, in the noise-free case, the columns of $\boldsymbol{S}$ are eigenvectors of $\boldsymbol{M}$ with eigenvalue 1. Thus, following Cucuringu et al. (2012a, b), we can use the d leading eigenvectors[7] of $\boldsymbol{M}$ as the basis for estimating the transformations. Let $\boldsymbol{U} = (\boldsymbol{U}_1, \ldots, \boldsymbol{U}_p)^T$ be the $pd \times d$ matrix whose columns are the d leading eigenvectors of $\boldsymbol{M}$, where $\boldsymbol{U}_i$ is the $d \times d$ block of $\boldsymbol{U}$ corresponding to patch P_i. We obtain the estimate $\hat{\boldsymbol{S}}_i$ of $\boldsymbol{S}_i$ by finding the nearest orthogonal transformation to $\boldsymbol{U}_i$ using an SVD (Horn et al., 1988), and hence the estimated rotation-synchronised embedding of patch P_i is $\hat{\boldsymbol{X}}^{(i)} = \tilde{\boldsymbol{X}}^{(i)}\hat{\boldsymbol{S}}_i^T$.

[6] Note that the rotation/reflection can be estimated without knowing the relative translation.

[7] While $\boldsymbol{M}$ is not symmetric, it is similar to a symmetric matrix and thus has real eigenvectors.

3.3 Synchronisation over translations

After synchronising the rotation of the patches, we can estimate the translations by solving a least-squares problem. Let $\hat{\boldsymbol{X}}_i^{(k)} \in \mathbb{R}^d$ be the (rotation-synchronised) embedding of node i in patch P_k ($\hat{\boldsymbol{X}}_i^{(k)}$ is only defined if $i \in P_k$).

Let $\boldsymbol{T}_k \in \mathbb{R}^d$ be the translation of patch k; then, in the noise-free case, the following consistency equations hold true

$$\hat{\boldsymbol{X}}_i^{(k)} + \boldsymbol{T}_k = \hat{\boldsymbol{X}}_i^{(l)} + \boldsymbol{T}_l, \qquad i \in P_k \cap P_l. \tag{7}$$

We can combine the conditions in Eq. (7) for each edge in the patch graph to arrive at

$$\boldsymbol{B}\boldsymbol{T} = \boldsymbol{C}, \qquad \boldsymbol{C}_{(P_k,P_l)} = \frac{\sum_{i \in P_k \cap P_l} \hat{\boldsymbol{X}}_i^{(k)} - \hat{\boldsymbol{X}}_i^{(l)}}{\left|P_k \cap P_l\right|}, \tag{8}$$

where $\boldsymbol{T} \in \mathbb{R}^{|\mathcal{P}| \times d}$ is the matrix such that the kth row of $\boldsymbol{T}$ is the translation $\boldsymbol{T}_k$ of patch P_k and $\boldsymbol{B} \in \{-1, 1\}^{|E_p| \times |\mathcal{P}|}$ is the incidence matrix of the patch graph with entries $\boldsymbol{B}_{(P_k,P_l),j} = \delta_{lj} - \delta_{kj}$, where δ_{ij} denotes the Kronecker delta. Equation (8) defines an overdetermined linear system that has the true patch translations as a solution in the noise-free case. In the practical case of noisy patch embeddings, we can instead solve Eq. (8) in the least-squares sense

$$\hat{\boldsymbol{T}} = \underset{\boldsymbol{T} \in \mathbb{R}^{p \times d}}{\arg\min} \|\boldsymbol{B}\boldsymbol{T} - \boldsymbol{C}\|_2^2. \tag{9}$$

We estimate the aligned node embedding $\bar{\boldsymbol{X}}$ in a final step using the centroid of the aligned patch embeddings of a node, i.e.,

$$\bar{\boldsymbol{X}}_i = \frac{\sum_{\{P_k \in \mathcal{P} : i \in P_k\}} \hat{\boldsymbol{X}}_i^{(k)} + \hat{\boldsymbol{T}}_k}{\left|\{P_k \in \mathcal{P} : i \in P_k\}\right|}. \tag{10}$$

3.4 Scalability of the *local2global* algorithm

The patch alignment step of *local2global* is highly scalable and does not directly depend on the size of the input data. The cost for computing the matrix M is $O(\left|E_p\right| od^2)$ where o is the average overlap between connected patches (typically $o \sim d$) and the cost for computing the vector b is $O(\left|E_p\right| od)$. Both operations are trivially parallelisable over patch edges. The translation problem can be solved with an iterative least-squares solver with a per-iteration complexity of $O(\left|E_p\right| d)$. The limiting step for *local2global* is usually the synchronization over orthogonal transformations which requires finding d eigenvectors of a $d|\mathcal{P}| \times d|\mathcal{P}|$ sparse matrix with $\left|E_p\right| d^2$ non-zero entries for a per-iteration complexity of $O(\left|E_p\right| d^3)$. This means that in the typical scenario where we want to keep the patch size constant, the patch alignment scales almost linearly with the number of nodes in the dataset, as we can ensure that the patch graph remains sparse, such that $\left|E_p\right|$ scales almost linearly with the number of patches. The $O(\left|E_p\right| d^3)$ scaling puts some limitations on the embedding dimension attainable with the *local2global* approach, though, as we can see from the experiments in

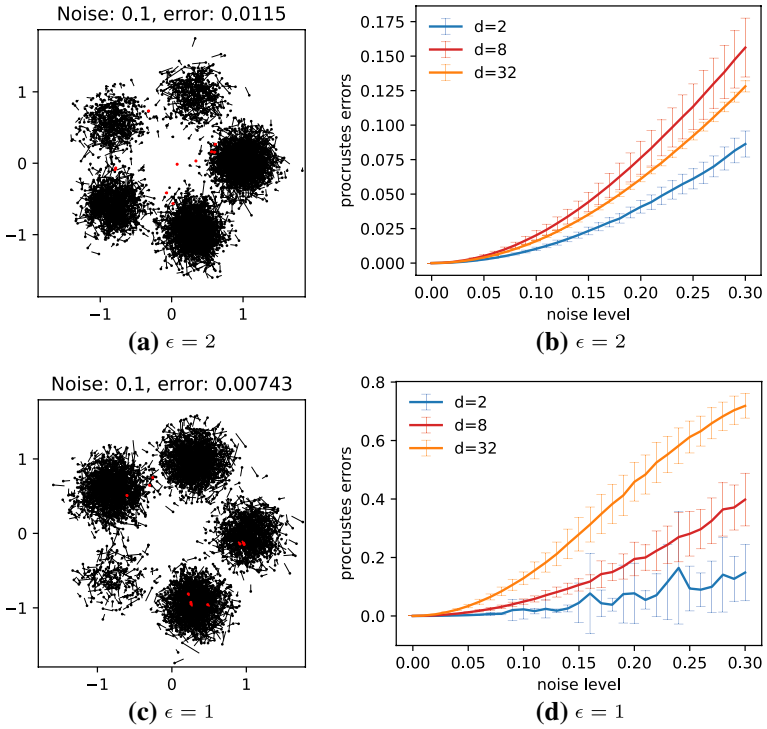

Fig. 2 Reconstruction error for *local2global* on synthetic test data with (**a**), (**b**) large patch overlaps ($\epsilon = 2$) and (**c**), (**d**) small patch overlaps ($\epsilon = 1$). (**a**) and (**c**) The tails indicate the difference between original and reconstructed points and patch centers are highlighted in red

Sect. 4.5, it remains feasible for reasonably high embedding dimension. We note that one can use a hierarchical version of *local2global* (see Sect. 6) to further increase the embedding dimension in a scalable fashion.

The preprocessing to divide the network into patches scales as $O(m)$. The speed-up attainable due to training patches in parallel depends on the oversampling ratio (i.e., the total number of edges in all patches divided by the number of edges in the original graph). As seen in Sect. 4.5, we achieve good results with moderate oversampling ratios.

3.5 Global reconstruction for synthetic data

The core goal of the *local2global* algorithm is to reconstruct a global embedding from a set of noisy local embeddings. Testing this reconstruction directly on real-world graph datasets is difficult due to the lack of ground-truth embeddings. Instead, we consider synthetic test data where we generate the ground-truth embedding directly in this section and use the performance on down-stream tasks as a proxy for embedding quality when evaluating *local2global* on real-world data in Sect. 4.

We generate the synthetic data by first selecting 5 cluster centers c_i, $i \in \{1, \dots, 5\}$. For $d = 2$, we use 5 equally-spaced points on the unit circle as the cluster centers and for $d > 2$ we sample the cluster centers uniformly at random on the d-dimensional unit

sphere. For each cluster, we sample the number of points n_i uniformly at random from [128, 2000] and sample n_i points from an uncorrelated d-dimensional normal distribution with mean c_i and standard deviation 0.2. The resulting ground-truth embedding has non-trivial variations in point density and obvious clusters (see Fig. 2). From the ground-truth embedding we construct a set of overlapping patches as follows:

1. Select 10 data points uniformly at random as the patch centers p_j, $j \in \{1, \dots, 10\}$ and initialise the patches $P_j = \{p_j\}$.
2. For each data point, find the distances d_j to all patch centers and add it to patch P_j if $d_j \leq \epsilon \min_j d_j$ (i.e., a point is added to the patch with the nearest center and any other patch with a center that is at most a factor ϵ further away than the nearest one).
3. Consider each patch P_i in turn and if $\left|P_i\right| < 128$ or $\left|\{j : \left|P_i \cap P_j\right| \geq 64\}\right| < 4$, expand P_i by adding the next-nearest data point that is not already included in the patch. Repeat until all patches satisfy the constraints.
4. Construct an initial patch graph $G_p(\mathcal{P}, E_p)$ by connecting a pair of patches P_i, P_j if $\left|P_i \cap P_j\right| \geq 64$.
5. If $G_p(\mathcal{P}, E_p)$ is not connected, consider all pairs of patches P_i, P_j such that P_i and P_j are in different components and order them by the distance between patch centers. Find the 64 data points with the smallest combined distance to p_i and p_j for the first pair and add them to both patches if they are not already included and add $\{P_i, P_j\}$ to E_p. Apply the same procedure to the next-closest pair of patches until the patch graph becomes connected. (Note that multiple edges may be added between the same pair of original components before the patch graph becomes connected.)

For the results in Fig. 2, we generate 10 independent ground-truth embeddings for each choice of d using the procedure above and report the mean and standard deviation of the results. We construct the patch embeddings by perturbing the ground truth embedding for data points in the patch by first adding normally distributed noise with mean 0 and standard deviation given by the noise level in Fig. 2. Next we apply an orthogonal transformation sampled uniformly at random, a normally-distributed shift with standard deviation 100 and a scaling sampled log-uniformly from [0.01, 100]. Finally, we apply the *local2global* algorithm to reconstruct the global embedding and use standard Procrustes analysis to compute the alignment error between the ground-truth and reconstructed embedding where the Procrustes error is the sum of squared differences between the standardised and aligned ground-truth and reconstructed embeddings. We consider two scenarios, $\epsilon = 2$, which results in large patch overlaps and a dense patch graph and $\epsilon = 1$, where the patch overlaps are only driven by the connectivity and degree constraints, resulting in small patch overlaps and a sparse patch graph. Comparing the results between Fig. 2b and d, we see that we can closely reconstruct the ground-truth for small levels of noise in both scenarios. However, increasing the amount of overlap between patches clearly improves the robustness of the reconstruction for high levels of noise especially for higher-dimensional embeddings.

Table 1 Data sets

	Nodes	Edges	Features	Classes
Cora (Yang et al., 2016)	2708	10556	1433	7
Amazon photo (Shchur et al., 2019)	7650	238162	745	8
Pubmed (Yang et al., 2016)	19717	88648	500	3
MAG240m (Hu et al., 2021)	121751666	2593241212	768	153

4 Evaluation on edge reconstruction and semi-supervised classification

4.1 Data sets

To test the viability of the *local2global* approach to graph embeddings, we consider data sets of increasing node set size. For the small-scale tests, we consider the Cora and Pubmed citation data sets from Yang et al. (2016) and the Amazon photo co-purchasing data set from Shchur et al. (2019).

For the large-scale test we use the MAG240m data from Hu et al. (2021). For MAG240m, we consider only the citation edges and preprocess the data set to make the edges undirected. We use the features as is without further processing. We show some statistics of the data sets in Table 1.

4.2 Patch graph construction

The first step in the *local2global* embedding pipeline is to divide the network $G(V, E)$ into overlapping patches. In some federated-learning applications, the network may already be partitioned and some or all of the following steps may be skipped provided the resulting patch graph is connected and satisfies the minimum overlap condition for the desired embedding dimension. Otherwise, we proceed by first partitioning the network into non-overlapping clusters and then enlarging clusters to create overlapping patches. This two-step process makes it easier to ensure that patch overlaps satisfy the conditions for the *local2global* algorithm without introducing excessive overlaps, than if we were to use a clustering algorithm that produces overlapping clusters directly. We use the following pipeline to create the patches:

- *Partition the network* into p non-overlapping clusters $\mathcal{C} = \{C_k\}_{k=1}^{p}$ such that $|C_k| \geq \frac{d+1}{2}$ for all k. We use METIS (Karypis & Kumar, 1998) to cluster the networks for the small-scale experiments in Sect. 4.5. For the large-scale experiments using MAG240m, we use FENNEL (Tsourakakis et al., 2014) to cluster the network. FENNEL obtains high-quality clusters in a single pass over the nodes and is thus scalable to very large networks. We choose these two techniques for the purpose of this paper as they are relatively standard and they are scalable as needed. One could a priori use any other clustering technique or assign nodes to clusters based on known metadata.
- *Initialize the patches* to $\mathcal{P} = \mathcal{C}$ and define the patch graph $G_p(\mathcal{P}, E_p)$, where $\{P_i, P_j\} \in E_p$ iff there exist nodes $i \in P_i$ and $j \in P_j$ such that $\{i, j\} \in E$. (Note that if G is connected,

G_p is also connected.) If the patch graph defined in this way is not connected (which can happen if the underlying graph G has multiple connected components), we proceed by finding the connected components of the patch graph. For each pair of connected components of the patch graph we sample a pair of patches and add the corresponding edge to the patch graph, ensuring that the patch graph is connected.

- *Sparsify the patch graph* G_p to have mean degree k using Algorithm 1 adapted from the effective-resistance sampling algorithm of Spielman and Srivastava (2011). The patch graph constructed in the previous step is often dense which would make the alignment step of the *local2global* algorithm non-scalable. The goal for the sparsification step is to minimise the influence of edges in G that fall between patches and that are not represented by a corresponding edge in G_p. Such edges are likely excluded from the training process which may bias the results. We use the conductance

$$c_{ij} = \frac{\left|\{(u,v)\in E : u\in P_i, v\in P_j\}\right|}{\min\left(\left|\{(u,v)\in E : u\in P_i\}\right|, \left|\{(u,v)\in E : u\in P_j\}\right|\right)} \tag{11}$$

as a proxy for the importance of a patch edge $\{P_i, P_j\}$ in representing the underlying structure of G. The assumption here is that an edge between patches with many internal edges has a much smaller influence on the resulting embedding than one between patches where at least one of the patches has a small number of internal edges. Based on the conductance weights, we first compute the effective resistance of each patch edge using the algorithm of Spielman and Srivastava (2011). We then construct the sparsified patch graph by first including all edges of a maximum spanning tree (this ensures that the patch graph remains connected) and then sampling the remaining edges without replacement with probability proportional to the effective resistance. For MAG240m, we construct the sparsified patch graph based on the raw conductance values directly rather than the effective resistances as computing effective resistances for all patch edges is no longer computationally feasible at this scale.

- *Expand the patches* to create the desired patch overlaps. We define a lower bound $l \geq d+1$ and upper target u for the desired patch overlaps and use Algorithm 2 to expand the patches such that $\left|P_i \cap P_j\right| \geq l$ for all $\{P_i, P_j\} \in E_p$.

Algorithm 1: Sparsify patch graph

Input: $G_p(\mathcal{P}, E_p)$, $G(V, E)$, target patch degree k
Result: sparsified patch graph $G_p(\mathcal{P}, \tilde{E}_p)$
foreach $\{P_i, P_j\} \in E_p$ **do**
  Compute conductance weight c_{ij} using Eq. (11)
foreach $\{P_i, P_j\} \in E_p$ **do**
  Compute effective resistance r_{ij} between P_i and P_j in $G_p(\mathcal{P}, E_p, c)$ using the algorithm of Spielman and Srivastava (2011);
  Let $w_{ij} = r_{ij}c_{ij}$;
Initialize $\tilde{E}_p$ with a maximum spanning tree of $G_p(\mathcal{P}, E_p, w)$;
Sample the remaining $(k-1)p+1$ edges from $E_p \setminus \tilde{E}_p$ without replacement and add them to $\tilde{E}_p$, where edge $\{P_i, P_j\}$ is sampled with probability proportional to w_{ij};
return $G_p(\mathcal{P}, \tilde{E}_p)$

Algorithm 2: Create overlapping patches

Input: $\mathcal{C}$, E_p, $G(V, E)$, min overlap l, max overlap u
Result: Overlapping patches $\mathcal{P}$
Initialise $\mathcal{P} = \mathcal{C}$;
Define the neighbourhood of a set of nodes U as

$$N(U) = \{j : i \in U, (i, j) \in E, j \notin U\};$$

foreach $P_i \in \mathcal{P}$ **do**
  foreach P_j *s.t.* $\{P_i, P_j\} \in E_p$ **do**
    Let $F = N(C_i) \cap C_j$;
    while $|P_i \cap C_j| < l/2$ **do**
      if $|F| + |P_i \cap C_j| > u/2$ **then**
        reduce F by sampling uniformly at random such that $|F| = u/2 - |P_i \cap C_j|$;
      Let $P_i = P_i \cup F$;
      Let $F = (N(F) \cap C_j) \setminus P_i$;
return $\mathcal{P}$

4.3 Embedding models

We consider two embedding approaches with different objective functions, the variational graph auto-encoder (VGAE) architecture of Kipf and Welling (2016) and deep-graph infomax (DGI) architecture of Veličkovč et al. (2019). We reiterate however that one can a-priori incorporate any embedding method within our pipeline (see Sect. 3). From this perspective, the expressivity of the *local2global* algorithm largely depends on the chosen baseline embedding method, as the patch synchronisation process would promote similarity between local embeddings and the global embedding. For instance, a smaller number of patches would mean less distortion of the global embedding. Nevertheless, the present work mainly focuses on distributed representation learning and we leave a full investigation on expressivity as future work.

For the choices of the training hyperparameters we largely follow the choices of Kipf and Welling (2016), Veličkovč et al. (2019). We use the Adam optimizer (Kingma & Ba, 2015) for training with learning rate set to 0.001 to train all models. We use early-stopping on the training loss with a maximum training time of 10 000 epochs and a patience of 20 epochs as in Veličkovč et al. (2019).[8] For DGI, we normalize the bag-of-word features such that the features for each node sum to 1 as in Veličkovč et al. (2019). However, we do not apply the feature normalization when training VGAE models as it severely hurts performance. We set the hidden dimension of the VGAE models to $4 \times d$, where d is the embedding dimension.[9]

4.4 Evaluation tasks

We consider two tasks for evaluating the embedding quality. The first task we consider is network reconstruction, where one tries to reconstruct the edges based on the embedding.

[8] We use the same training strategy also for the VGAE models as we find that it gives a small improvement over the training strategy of Kipf and Welling (2016) in preliminary experiments.

[9] We explored setting the hidden dimension to $2 \times d$ in preliminary experiments. However this led to training instability for small embedding dimension.

We use the inner product between embedding vectors as the scoring function and use the area under the ROC curve (AUC) as the quality measure.

The second task we consider is semi-supervised classification. We follow the procedure of Veličkovč et al. (2019) and train a classification model to predict the class labels based on the embedding. For the small-scale tests in Sect. 4.5.1 we use a simple logistic classifier whereas for MAG240m we use a deeper MLP as detailed in Sect. 4.5.2. In Sect. 4.5.1, we use 20 random labelled examples per class as the training set, 500 random examples as the validation set and the remaining examples as the test set. We sample a new random train-test split for each run of the classifier. We train the classifier for a maximum of 10 000 epochs with early stopping based on the validation loss with a patience of 20 epochs using the ADAM optimiser (Kingma & Ba, 2015) with a learning rate of 0.01.[10] For the large-scale test on MAG240m, we use the full training data to train the classifier and evaluate the performance on the validation set,[11] except for 10 000 randomly sampled papers which are used as the validation set.

4.5 Results

To get some intuition about the embeddings we obtain, we first visualise the results using UMAP (McInnes et al., 2020). We show the results for a $d = 128$ stitched *local2global* embeddings obtained with VGAE in Fig. 3. For the small-scale data sets, we observe that the different classes appear clustered in the embedding. While some clusters are visible for MAG240m, they appear less well-defined. This may be partly due to the much larger number of classes (153), and may also suggest that the quality of the embedding is not as good.

4.5.1 Small-scale tests

For the small-scale tests it is feasible to train the embedding models directly on the entire data set using full-batch gradient descent. This makes it possible to directly compare the quality of the embeddings obtained by stitching patch-embeddings using *local2global* with using the underlying embedding method directly. As a baseline, we also consider the case where we do not apply any alignment transformations and simply compute the node embeddings by taking the centroid over the independently trained patch embeddings (no-l2g below). Additionally, we also perform an ablation study where we only apply some of the patch-alignment steps during the stitching process. This results in the following five scenarios:

- *full:* Model trained on the full data.
- *l2g:* Separate models trained on the subgraph induced by each patch and stitched using the *local2global* algorithm.
- *rotate-only* Patches are aligned by only synchronising over orthogonal transformations without applying rescaling or translation.
- *translate-only* Patches are aligned by finding the optimal translations without synchronisation over scales or orthogonal transformations.

[10] We found this choice of learning rate to work well in preliminary experiments whereas much larger learning rates can result in unstable training and much smaller learning rates result in very slow training.

[11] The labels for the test set are not currently included in the public data.

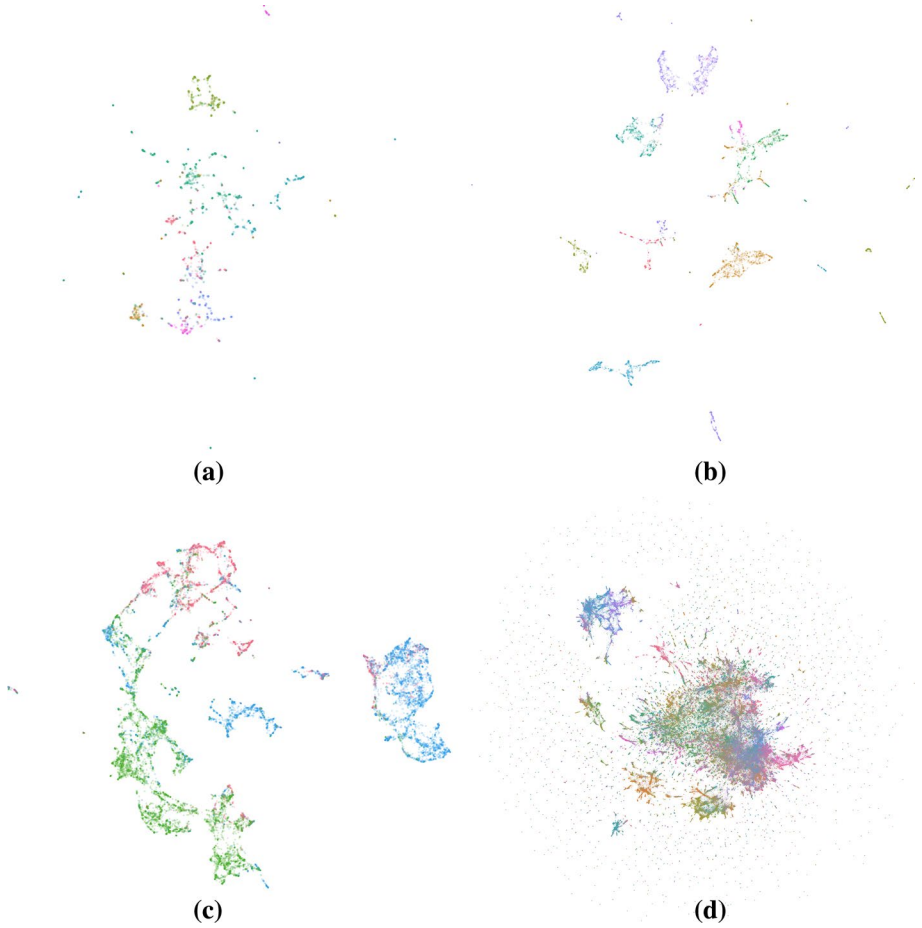

Fig. 3 UMAP projection of 128-dimensional VGAE-l2g embeddings for (**a**) Cora, (**b**) Amazon photo, (**c**) Pubmed, and (**d**) MAG240m. Nodes are coloured by class label. For MAG240m, the visualisation is based on a sample of 500 000 labeled papers

- *no-l2g:* Centroid over patch embeddings that contain the node without applying any alignment transformations.

For Cora and Amazon photo, we split the network into 10 patches and sparsify the patch graph to a target mean degree $k = 4$. For Pubmed, we split the network into 20 patches and sparsify the patch graph to a target mean degree of $k = 5$. We set the lower bound for the overlap to $l = 256$ and upper bound to $u = 1024$.[12]

[12] These parameter choices result in sufficient overlap between patches to obtain stable alignment results without excessive oversampling of edges. Using standard model-selection procedures such as cross-validation to select the hyper-parameters for the patch graph construction is difficult as the best performance can typically be achieved by including each node in every patch which results in an ensemble embedding over full models and looses all scalability and parallelisation benefits.

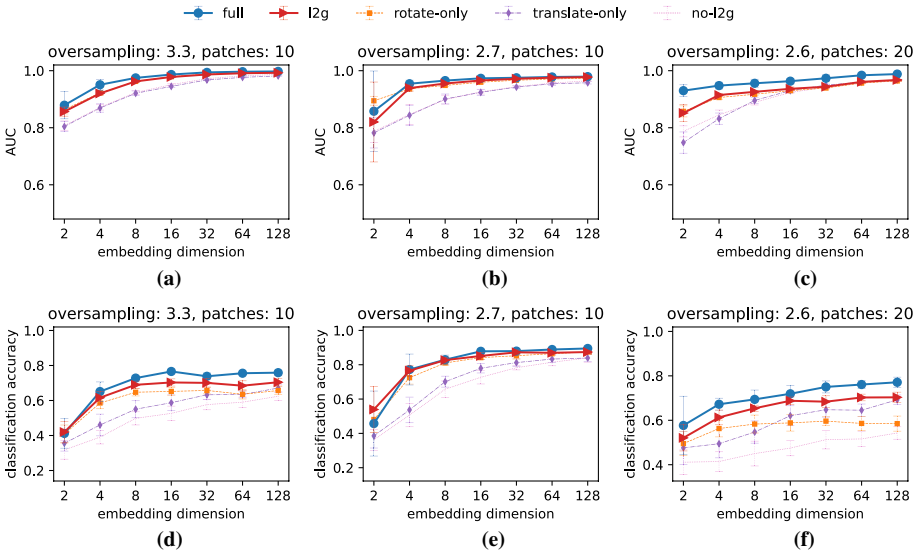

Fig. 4 VGAE embeddings: AUC network reconstruction score (top) and classification accuracy (bottom) as a function of embedding dimension using full data or stitched patch embeddings for (**a**), (**d**) Cora, (**b**), (**e**) Amazon photo, and (**c**), (**f**) Pubmed

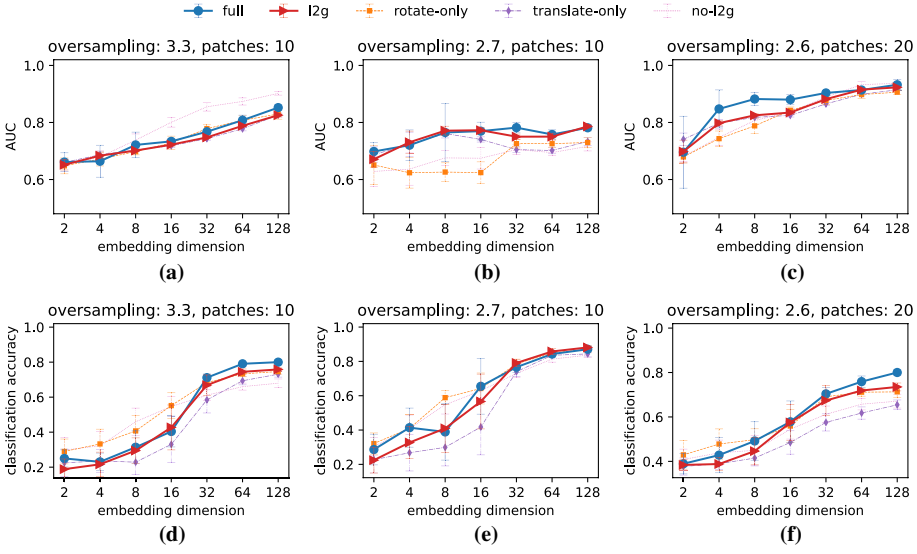

Fig. 5 DGI embeddings: AUC network reconstruction score (top) and classification accuracy (bottom) as a function of embedding dimension using full data or stitched patch embeddings for (**a**), (**d**) Cora, (**b**), (**e**) Amazon photo, and (**c**), (**f**) Pubmed

We train each model 10 times starting with different random initialisations and additionally consider 5 independent random train-test splits for each embedding (for a total of 50 random train-test splits) when evaluating the semi-supervised classification performance.

We show the resulting means and standard deviations for VGAE embeddings in Fig. 4 and for DGI embeddings in Fig. 5. Overall, the gap between the results for 'l2g' and 'full' is generally small. The aligned 'l2g' embeddings typically outperform the unaligned 'no-l2g' baseline by a significant margin highlighting the benefit of the alignment steps. Notable exceptions are the DGI embeddings for low embedding dimension (see Fig. 5) where the unaligned 'no-l2g' embeddings outperform both the 'full' and 'l2g' embeddings. However, the overall accuracy is low and the behaviour of 'l2g' closely tracks that of the 'full' embedding, suggesting that this is a limitation of the underlying embedding approach, which requires relatively high embedding dimension for good performance, rather than of the *local2global* alignment. Comparing the partial alignment scenarios we see that 'l2g' typically performs at least as well as 'rotate-only' and 'translate-only' with the notable exception of low-dimensional DGI embeddings. While the performance of 'rotate-only' is often very close to that of 'l2g', the synchronisation over orthogonal transformations is by far the most computationally intensive alignment operation, such that there is little reason to skip the other transformations as they do not hurt and sometimes can be a significant improvement (see especially the VGAE embeddings for Pubmed in Fig. 4f).

Comparing the two embedding methods, we see that VGAE has better network reconstruction performance than DGI. This is not surprising, as network reconstruction corresponds to the training objective of VGAE. In terms of semi-supervised classification performance, we see that VGAE achieves good results already for low embedding dimension with $d = 16$ often being sufficient whereas DGI requires higher embedding dimensions for good results. This makes VGAE a better fit for use with *local2global* in practice due to the scalability characteristics of the *local2global* algorithm (see Sect. 3.4). For networks of this size, full-batch training is still very fast and memory requirements are not an issue. As such, we do not expect to see significant benefits from using *local2global* in terms of execution times. However, for completeness, we include a comparison of execution times in "Appendix".

An important hyper-parameter for the *local2global* algorithm is the choice of the number of patches as it controls the amount of parallelisation possible. For small networks, a large number of patches will result in excessive patch overlaps. For large networks, (see, e.g., Sect. 4.5.2), the choice of the number of patches is essentially dictated by memory constraints.[13] In Fig. 6 we show the behaviour of the semi-supervised classification accuracy as we increase the number of patches for Pubmed. While we see a slight downward trend in accuracy as the number of patches increases, the effect is small and seems to decrease with embedding dimension. The oversampling ratio increases slowly with the number of patches from a ratio of 1.3 for 5 patches to 3.4 for 50 patches. We also observed similar trends on the smaller networks.

4.5.2 Large-scale tests

For the large-scale MAG240m network we use two iterations of FENNEL (Tsourakakis et al., 2014) to divide the network into 10 000 clusters.[14] We set the lower bound for the overlap to $l = 512$ and the upper bound to $u = 2048$ and sparsify the patch graph to have a mean degree $\langle k \rangle = 20$.

[13] One can in principle combine *local2global* with layer-wise sampling to overcome this constraint.

[14] The choice for the number of patches is largely driven by the requirement for the largest patch to be trainable within the 32GB memory of a single GPU on the JADE2 cluster used for training.

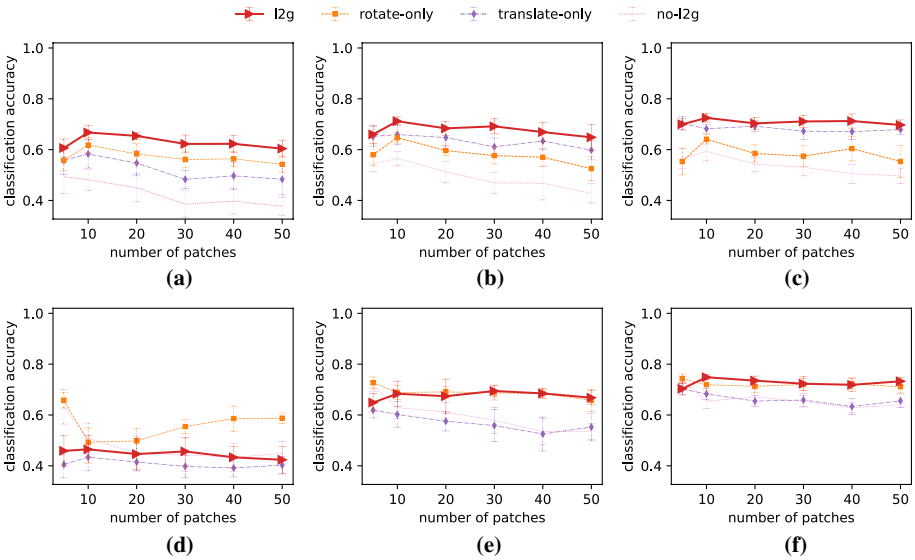

Fig. 6 Effect of the number of patches on the classification accuracy achieved by *local2global* embeddings for Pubmed for VGAE with (**a**) $d = 8$, (**b**) $d = 32$, and (**c**) $d = 128$ and DGI with (**d**) $d = 8$, (**e**) $d = 32$, and (**f**) $d = 128$

Table 2 Network reconstruction (AUC) and classification accuracy on MAG240m for $d = 128$ with VGAE embeddings (first two rows) and DGI embeddings (second two rows)

	l2g	no-l2g
VGAE reconstruction (AUC)	0.96	0.70
VGAE classification (Acc.)	0.54	0.42
DGI reconstruction (AUC)	0.91	0.69
DGI classification (Acc.)	0.53	0.45

We show the results for VGAE and DGI in Table 2. For the semi-surpervised classification results we use a grid search over the parameters of the MLP as in Hu et al. (2021), with hidden dimension in $\{128, 256, 512, 1024\}$, number of layers in $\{2, 3, 4\}$, and dropout probability in $\{0, 0.25, 0.5\}$. We fix the batch size to 100 000 and use the Adam optimizer (Kingma & Ba, 2015) with learning rates in $\{0.01, 0.001, 0.0001\}$. We use early stopping based on the validation loss with patience of 20 epochs and maximum number of epochs set to 1000. We use batch normalisation (Ioffe & Szegedy, 2015) after each layer together with a ReLU non-linearity. We search over the hyperparameters separately for each embedding and report results for the model with the highest validation accuracy. However, the best parameter setting is consistent across all embeddings reported in Table 2 with the highest validation accuracy achieved by a 4-layer MLP with hidden dimension 1024 trained with learning rate set to 0.0001. There is a clear improvement when comparing the results for *local2global* (l2g) with the no transformation baseline (no-l2g), and performance results for network reconstruction are high for both embedding methods. While the classification accuracies we obtain with *local2global* in Table 2 with VGAE and DGI are on par with that of MLP in Hu et al. (2021), we note that the classification accuracy results are not directly comparable. All approaches in Hu et al. (2021) differ from the *local2global*

pipeline in at least one of two fundamental ways: (1) the training step is supervised and (2) there is no dimensionality reduction on a feature level.

5 Application to anomaly detection

As an application of our *local2global* algorithm, we consider the task of anomaly detection in a temporal cyber-security data set. An anomaly is defined as an outlier data point, which does not follow the collective common pattern of the majority of the data at hand, and there are strong premises for highlighting and separating it from the rest of the data. One popular class of such methods falls under the umbrella of unsupervised learning algorithms, techniques aimed at uncovering patterns in data without any available labels. Existing methods usually look for areas of high density of points and assign them to clusters, while points in less dense regions are not included in the clusters and flagged as anomalies Ester et al. (1996). However, this approach hinges on the assumption that there exists a single global embedding of all the points in the study (where points that behave more similarly have a higher proximity in the embedding), and this embedding is then leveraged for the anomaly detection task via a preferred method of choice. One could use the *local2global* framework to significantly increase the scalability of such an approach. In what follows, we investigate whether the consistency of embeddings over time can be used to detect outlier behavior.

In particular, we consider the netflow part of the Los Alamos National Laboratory Unified Host and Network Data Set (LANL-netflow, Turcotte et al. (2018)). LANL-netflow consists of 89 days of processed network flow records (see Turcotte et al. (2018) for details). For our experiments we aggregate the network flows per day, restricted to TCP. We treat the network as directed and unweighted, only considering the presence or absence of a directed connection. For the embeddings, we consider each day as a patch and treat the network as bipartite, computing separate source and destination embeddings for each node using the SVD-based method of Dhillon (2001).

5.1 Patch embeddings

In particular, let $\boldsymbol{A}^{(t)}$ be the adjacency matrix for the network at day t, where $A^{(t)}_{ij} = 1$ if there was a directed connection between source node i and destination node j during day t and $A^{(t)}_{ij} = 0$ otherwise. We consider the normalized adjacency matrix $\boldsymbol{A}^{(t)}_n = \boldsymbol{D}^{(t)^{-\frac{1}{2}}}_s \boldsymbol{A}^{(t)} \boldsymbol{D}^{(t)^{-\frac{1}{2}}}_d$, where $\boldsymbol{D}^{(t)}_s$ and $\boldsymbol{D}^{(t)}_d$ are the diagonal source and destination degree matrices respectively, i.e., $\boldsymbol{D}^{(t)}_{s\ ii} = \sum_j \boldsymbol{A}^{(t)}_{ij}$ and $\boldsymbol{D}^{(t)}_{d\ jj} = \sum_i \boldsymbol{A}^{(t)}_{ij}$. We define the (unaligned) source embedding $\boldsymbol{X}^{(t)}$ and destination embedding $\boldsymbol{Y}^{(t)}$ based on the singular value decomposition of $\boldsymbol{A}^{(t)}_n$ such that $\boldsymbol{X}^{(t)} = \boldsymbol{D}^{(t)^{-\frac{1}{2}}}_s \boldsymbol{U}^{(t)}$ and $\boldsymbol{Y}^{(t)} = \boldsymbol{D}^{(t)^{-\frac{1}{2}}}_d \boldsymbol{V}^{(t)}$ where $\boldsymbol{U}^{(t)}$ is the matrix with columns given by the d leading left singular vectors and $\boldsymbol{V}^{(t)}$ the matrix with columns given by the d leading right singular vectors of $\boldsymbol{A}^{(t)}_n$ in the space orthogonal to the trivial solution, i.e., $\mathbf{1}\boldsymbol{D}^{(t)^{\frac{1}{2}}}_s \boldsymbol{U}^{(t)} = \mathbf{0}$ and $\mathbf{1}\boldsymbol{D}^{(t)^{\frac{1}{2}}}_d \boldsymbol{V}^{(t)} = \mathbf{0}$.

5.2 Reference embedding

We use the *local2global* algorithm to construct a reference embedding to use as a baseline for identifying anomalous behavior. As mentioned above, we treat each day of observations as a patch. Computers are largely persistent on the network from one day to the next, such that we have sufficiently many overlapping nodes between different patches to reliably estimate the relative transformations. We define the patch graph $G_p(\mathcal{P}, E_p)$ based on the delay between patches, such that $\{P_s, P_t\} \in E_p \iff |s-t| \in \{1, 7, 14, 21, 28, 35, 42, 49\}$, i.e., we compute relative transformations between patches that are one day and multiples of one week apart. In preliminary experiments, we found that including long-range connections in the patch graph is crucial to obtaining good results in this application. We separately apply the *local2global* algorithm to obtain a reference source embedding $\bar{\boldsymbol{X}}$ and a reference destination embedding $\bar{\boldsymbol{Y}}$ which are defined as the centroid over the aligned patch source embeddings $\{\bar{\boldsymbol{X}}^{(t)}\}$ and destination embeddings $\{\bar{\boldsymbol{Y}}^{(t)}\}$ (see Eq. (10)). Note that we only compute a single reference embedding based on all patches. In practical applications, one would apply the alignment in a streaming fashion, such that the reference embedding is only based on past data and updated as new data becomes available. However, given the limited snapshot of data available as part of the LANL-netflow data set, applying such a streaming algorithm is problematic.

5.3 Anomaly score

We define the raw source anomaly score $\Delta_{s_i}^{(t)} = \left\| \bar{\boldsymbol{X}}_i - \bar{\boldsymbol{X}}_i^{(t)} \right\|_2$ for source node i at time t, and raw destination anomaly score $\Delta_{d_j}^{(t)} = \left\| \bar{\boldsymbol{Y}}_j - \bar{\boldsymbol{Y}}_j^{(t)} \right\|_2$ for destination node j at time t based on the Euclidean distance between the reference embedding and the aligned patch embeddings. The raw scores are not comparable across nodes, as the behavior and hence the embedding is inherently more stable for some nodes than others.

We use a z-score transformation to standardize the scores and make them comparable between nodes to enable easier identification of anomalous behavior. We define the standardized anomaly scores as

$$\hat{\Delta}_{s_i}^{(t)} = \frac{\Delta_{s_i}^{(t)} - \mu_{-t}(\Delta_{s_i})}{\sigma_{-t}(\Delta_{s_i})}, \qquad \hat{\Delta}_{d_j}^{(t)} = \frac{\Delta_{d_j}^{(t)} - \mu_{-t}(\Delta_{d_j})}{\sigma_{-t}(\Delta_{d_j})}, \tag{12}$$

where we use μ_{-t} and σ_{-t} to denote the mean and standard deviation estimated based on all data points except the data point at time t. Leaving out the current point when estimating the mean and standard deviation has the effect of boosting the scores for outliers and thus leads to a clearer signal.

5.4 Results

We visualize the reference embeddings $\bar{\boldsymbol{X}}$ and $\bar{\boldsymbol{Y}}$ together with the raw anomaly scores in Fig. 7 using UMAP (McInnes et al., 2020). For most nodes, the embeddings are very stable across time with only small average error between aligned patch embeddings and the reference embedding. However, some nodes exhibit much noisier behavior with large average error. This difference in behavior is normalized out by the z-score

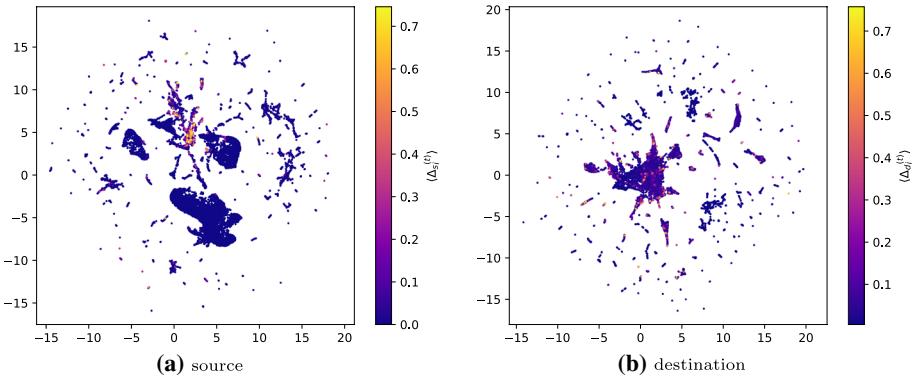

Fig. 7 UMAP visualisation of average alignment errors for (**a**) source embeddings and (**b**) destination embeddings. Only nodes with at least 21 days of observations are shown in the visualisation

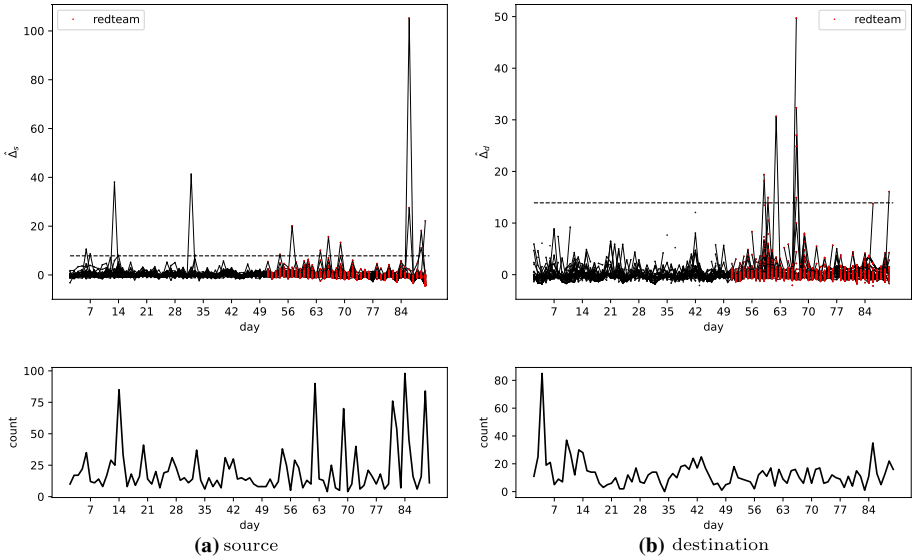

Fig. 8 Standardized anomaly scores for known red-team targets (top) and outlier countOutliers are defined as nodes with scores in the 0.999 quantile of the score distribution across days and nodes. (Outliers are defined as nodes with scores in the 0.999 quantile of the score distribution across days and nodes.) (bottom) for (**a**) source embeddings and (**b**) destination embeddings. The dashed line indicates the 0.999 quantile of the anomaly score distribution over all days and nodes with at least 21 days of observations. Days with observed redteam activity are highlighted in red

transformation of Eq. (12) such that it is possible to define a common threshold for anomalous behavior. Another source of potential issues are nodes with insufficient observations to reliably estimate normal behavior. While we use all nodes to compute the embeddings for each day and the reference embedding, we restrict the results in Figs. 7 and 8 to nodes with at least 21 days of observations.

Figure 8 shows the standardized anomaly scores for the nodes with known red-team activity. We see a clear increase in the number of outliers for the affected nodes during the attack period, especially for the destination embedding. This suggests that red-team activity may have been responsible for some unusual network activity. However, it should be noted that there is no clear separation between red-team activity and other sources of unusual activity, as we observe outliers of similar magnitude for other nodes throughout the data. This suggests that network flows may provide an additional useful source of signal for identifying suspicious behavior. Combining network flows with node-level information from the process logs using, for example, a GCN-based embedding approach such as VGAE or DGI, could be interesting. However, extracting useful features from the process data is not straightforward, and we leave this task as future work.

6 Conclusion

In this work, we introduced a framework that can significantly improve the computational scalability of generic graph embedding methods, rendering them scalable to real-world applications that involve massive graphs, potentially with millions or even billions of nodes. At the heart of our pipeline is the *local2global* algorithm, a divide-and-conquer approach that first decomposes the input graph into overlapping clusters (using one's method of choice), computes entirely-local embeddings via the preferred embedding method for each resulting cluster (exclusively using information available at the nodes within the cluster), and finally stitches the resulting local embeddings into a globally consistent embedding, using established machinery from the *group synchronization* literature. Our results on small-scale data sets achieve comparable accuracy on graph reconstruction and semi-supervised clustering as globally trained embeddings. We have also demonstrated that *local2global* achieves a good trade-off between scalability and accuracy on large-scale data sets using a variety of embedding techniques and downstream tasks. Our results also suggest that the application of *local2global* to the task of anomaly detection is fruitful with the potential for several future work directions.

More specifically, a first direction is to further increase the size of node and edge sets by another order of magnitude, and consider web-scale graphs with billions of nodes, which many existing algorithms will not be able to handle and for which distributed training is crucial. For very large data sets, or in other settings where we have a large number of patches, solving the eigenvalue problem implied by Eq. (6) to synchronise the orthogonal transformations can become a computational bottleneck. One could consider a hierarchical approach for the patch alignment step to address this issue. In a hierarchical approach, one would cluster the patch graph and apply *local2global* to align the patches within each cluster and compute the aligned node embedding for each cluster. One could then apply *local2global* again to align the patch clusters embeddings.[15] Such a hierarchical approach would make it trivial to parallelize the patch alignment, but may come with a penalty in terms of the embedding quality. It would be interesting to explore the trade-off between

[15] A preliminary implementation of this hierarchical approach is available in https://github.com/LJeub/Local2Global_embedding.

scale and embedding quality within this hierarchical framework. A related direction is to explore the interplay of our pipeline with different clustering algorithms, and in particular hierarchical clustering, and assess how the patch graph construction mechanism affects the overall performance.

A second direction is to further demonstrate particular benefits of locality and asynchronous parameter learning. These have clear advantages for privacy preserving and federated learning setups. It would also be particularly interesting to identify further settings in which this *local2global* approach can outperform global methods. The intuition and hope in this direction stems from the fact that asynchronous locality can be construed as a regularizer (much like sub-sampling, and similar to dropout) and could potentially lead to better generalization and alleviate the oversmoothing issues of deep GCNs, as observed in Chiang et al. (2019).

Finally, the application of *local2global* to anomaly detection is promising and shows that network flows could provide useful additional signal for identifying network intrusions, which is a particularly challenging task given the noisy nature of the data and the rarity of malicious events. It would be interesting to apply the methodology in Sect. 5 to further data sets (Highnam et al., 2021; DARPA OpTC, 2020), and to incorporate node-level attributes into the embedding techniques. Generating meaningful node-level features for anomaly detection (e.g., based on device logs) and incorporating a spatio-temporal dimension into the analysis also warrant further investigation.

Appendix: Execution time comparison

The main goal of our *local2global* method is to enable training on large-scale networks where full-batch training is not feasible. As mentioned in Sect. 4.5, for networks of the sizes shown bellow, full-batch training is very fast and memory requirements are not an issue. As such, we do not expect to see significant benefits from using *local2global* in terms of execution times. However, for completeness, we include a comparison of execution times (Tables 3, 4, 5).

In order to provide a more meaningful comparison, we disable early stopping for the timing comparisons and train all embedding models for 200 epochs in order to reduce the variance in execution times. For the training and alignment steps, the reported timings correspond to the median result over 10 runs. The execution time for patch training corresponds to the time needed when training all patches in parallel. The timings for the clustering and patch graph construction preprocessing steps correspond to a single run, the results of which are used for the subsequent training. The processing times do not include the overhead for loading data and saving results. We note that the large difference in training time between VGAE and DGI is due to an inefficient implementation of VGAE in torch-geometric 1.7.0 which we used for all experiments.

Table 3 Preprocessing and training times for Cora with 10 patches and training fixed to 200 epochs

Preprocessing							
	Time (s)						
Clustering	3.33						
Patches	12.94						
Training							
	Time (s)						
$d =$	2	4	8	16	32	64	128
VGAE							
Full	3.89	3.94	3.97	4.00	4.04	4.15	4.32
Patch	2.72	2.74	2.78	2.83	2.86	2.90	3.09
l2g align	0.13	0.15	0.29	0.30	0.47	1.44	4.69
DGI							
Full	1.31	1.34	1.37	1.41	1.44	1.48	1.61
Patch	1.30	1.32	1.34	1.35	1.40	1.42	1.49
l2g align	0.14	0.17	0.26	0.26	0.46	1.40	4.87

Table 4 Preprocessing and training times for Amazon photo with 10 patches and training fixed to 200 epochs

Preprocessing							
	Time (s)						
Clustering	3.43						
Patches	12.88						
Training							
	Time (s)						
$d =$	2	4	8	16	32	64	128
VGAE							
Full	55.24	55.70	55.92	56.12	56.92	58.24	60.93
Patch	23.78	23.57	23.61	23.50	23.91	24.44	25.32
l2g align	0.14	0.24	0.40	0.36	0.53	1.87	7.95
DGI							
Full	1.38	1.36	1.42	1.45	1.48	1.63	2.59
Patch	1.35	1.37	1.38	1.42	1.41	1.44	1.62
l2g align	0.24	0.22	0.43	0.34	0.58	1.82	6.11

Table 5 Preprocessing and training times for Pubmed with 20 patches and training fixed to 200 epochs

Preprocessing							
	Time (s)						
Clustering	3.37						
Patches	12.98						
Training							
	Time (s)						
$d =$	2	4	8	16	32	64	128
VGAE							
Full	24.19	24.26	24.30	24.60	24.72	25.65	27.59
Patch	5.63	5.59	5.62	5.68	5.76	5.77	6.03
l2g align	0.28	0.46	0.43	0.76	1.44	5.71	25.58
DGI							
Full	1.44	1.48	1.51	1.51	1.53	1.91	3.55
Patch	1.32	1.36	1.36	1.38	1.41	1.42	1.56
l2g align	0.36	0.57	0.91	0.83	1.53	4.32	24.99

Acknowledgements The authors would like to acknowledge support from the Defence and Security Programme at The Alan Turing Institute, funded by the Government Communications Headquarters (GCHQ).

Author Contributions LJ, GC, XD, MB, and MC designed the study and wrote the manuscript. LJ implemented the methods and performed all numerical experiments.

Funding This work was supported by the Defence and Security Programme at The Alan Turing Institute, funded by the Government Communications Headquarters (GCHQ).

Availability of data and material All data sets used in this work are publicly available: Cora, Amazon photo, Pubmed: Accessible through pytorch-geometric https://github.com/pyg-team/pytorch_geometric. MAG240m: Part of the open graph benchmark project https://ogb.stanford.edu. LANL Unified Host and Network Data Set: https://csr.lanl.gov/data/2017/

Declarations

Conflict of interest The authors declare no conflicts of interest.

Ethics approval Not Applicable.

Consent to participate Not Applicable.

Consent for publication Not Applicable.

Code availability All code associated with this paper is publicly available from https://github.com/LJeub/Local2Global_embedding and https://github.com/LJeub/Local2Global.

References

Bojchevski, A., Klicpera, J., Perozzi, B., Kapoor, A., Blais, M., & Rózemberczki, B., et al. (2020). Scaling graph neural networks with approximate PageRank. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 2464–2473). New York: ACM.

Busch, J., Pi, J., & Seidl, T. (2020). PushNet: Efficient and adaptive neural message passing. In *Proceedings of the 24th European conference on artificial intelligence* (pp. 1039–1046). The Netherlands: IOS Press.

Chen, J., Ma, T., & Xiao, C. (2018). FastGCN: Fast learning with graph convolutional networks via importance sampling. In *Proceedings of the 6th international conference on learning representations*

Chen, J., Zhu, J., & Song, L. (2018). Stochastic training of graph convolutional networks with variance reduction. In *Proceedings of the 35th international conference on machine learning* (Vol. 80, pp. 942–950). PMLR.

Chen, M., Wei, Z., Ding, B., Li, Y., Yuan, Y., Du, X., & Wen, J.-R. (2020). Scalable graph neural networks via bidirectional propagation. *Advances in Neural Information Processing Systems, 33*, 14556–14566.

Chen, T., Zhou, K., Duan, K., Zheng, W., Wang, P., Hu, X., & Wang, Z. (2022). Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. https://doi.org/10.1109/TPAMI.2022.3174515

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S., & Hsieh, C.-J. (2019). Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 257–266). New York: ACM.

Cucuringu, M., Lipman, Y., & Singer, A. (2012). Sensor network localization by eigenvector synchronization over the Euclidean group. *ACM Transactions on Sensor Networks, 8*(3), 1–42.
Cucuringu, M., Singer, A., & Cowburn, D. (2012). Eigenvector synchronization, graph rigidity and the molecule problem. *Information and Inference, 1*(1), 21–67.
DARPA OpTC (2020). Darpa operationally transparent cyber (optc) data. https://github.com/FiveDirections/OpTC-data.
Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining—KDD '01*. ACM Press. https://doi.org/10.1145/502512.502550
Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the second international conference on knowledge discovery and data mining* (pp. 226–231). AAAI Press.
Fey, M., Lenssen, J. E., Weichert, F., & Leskovec, J. (2021). GNNAutoScale: Scalable and expressive graph neural networks via historical embeddings. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (Vol. 139, pp. 3294–3304). PMLR. Retrieved from https://proceedings.mlr.press/v139/fey21a.html
Frasca, F., Rossi, E., Eynard, D., Chamberlain, B., Bronstein, M., & Monti, F. (2020). Sign: Scalable inception graph neural networks. arXiv:2004.11198 [cs.LG].
Hamilton, W. L., Ying, R., & Leskovec, J. (2018). Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems, 31*, 1025–1035.
Highnam, K., Arulkumaran, K., Hanif, Z., & Jennings, N. R. (2021). BETH dataset: Real cybersecurity data for anomaly detection research. In *ICML 2021 workshop on uncertainty and robustness in deep learning*
Horn, B. K. P., Hilden, H. M., & Negahdaripour, S. (1988). Closed-form solution of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America A, 5*(7), 1127–1135.
Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., & Leskovec, J. (2021). OGB-LSC: A large-scale challenge for machine learning on graphs. arXiv:2103.09430 [cs.LG].
Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd international conference on international conference on machine learning*–(Vol. 37, pp. 448–456). JMLR.org.
Jeub, L. G. S., Colavizza, G., Dong, X., Bazzi, M., & Cucuringu, M. (2021). Local2global: Scaling global representation learning on graphs via local training. Kdd 2021 workshop on deep learning on graphs.
Kairouz, P., & McMahan, H.B. (Eds.). (2021). Advances and open problems in federated learning. *Foundations and Trends in Machine Learning, 14*(1).
Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing, 20*(1), 359–392.
Kingma, D.P., & Ba, J.L. (2015). Adam: A method for stochastic optimization. In *Proceedings of the 3rd international conference on learning representations*.
Kipf, T. N., & Welling, M. (2016). Bayesian deep learning workshop (NIPS: Variational graph auto-encoders).
Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th international conference on learning representations*.
Klicpera, J., Bojchevski, A., Günnemann, S. (2019). Predict then propagate: Graph neural networks meet personalized pagerank. In *Proceedings of the 7th international conference on learning representations*.
McInnes, L., Healy, J., Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction. arXiv:1802.03426 [stat.ML].
Shchur, O., Mumme, M., Bojchevski, A., & Günnemann, S. (2019). Pitfalls of graph neural network evaluation. arXiv:1811.05868 [cs.LG].
Singer, A. (2011). Angular synchronization by eigenvectors and semidefinite programming. *Applied and Computational Harmonic Analysis, 30*(1), 20–36.
Spielman, D. A., & Srivastava, N. (2011). Graph sparsification by effective resistances. *SIAM Journal on Computing, 40*(6), 1913–1926.
Tsourakakis, C., Gkantsidis, C., Radunovic, B., Vojnovic, M. (2014). FENNEL: Streaming graph partitioning for massive scale graphs. In *Proceedings of the 7th ACM international conference on web search and data mining* (pp. 333–342). New Yark: ACM.
Turcotte, M. J. M., Kent, A. D., & Hash, C. (2018). Unified host and network data set. *Security Science and Technology*. (pp. 1–20). https://doi.org/10.1142/9781786345646001
Veličkovč, P., Fedus, W., Hamilton, W. L., Lió, P., Bengio, Y., & Hjelm, R. D. (2019). Deep graph infomax. In *Proceedings of the 7th international conference on learning representations*.

Wu, F., Zhang, T., de Souza Jr., A. H., Fifty, C., Yu, T., & Weinberger, K. Q. (2019). Simplifying graph convolutional networks. In *Proceedings of the 36th international conference on machine learning* (Vol. 97, pp. 6861– 6871). PMLR.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems, 32*(1), 4–24. https://doi.org/10.1109/tnnls.2020.2978386

Yang, Z., Cohen, W. W., Salakhutdinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd international conference on machine learning* (Vol. 48, pp. 40–48). PMLR.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., & Prasanna, V. (2019). Accurate, efficient and scalable graph embedding. In *2019 IEEE international parallel and distributed processing symposium* (pp. 462–471). New York: IEEE.

Zeng, H., Zhou, H., Srivastava, A., Kannan, R., & Prasanna, V. (2020). Graphsaint: Graph sampling based inductive learning method. In *Proceedings of the 8th international conference on learning representations*.

Zou, D., Hu, Z., Wang, Y., Jiang, S., Sun, Y., & Gu, Q. (2019). Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in Neural Information Processing Systems* (Vol. 32). New York: Curran Associates, Inc.

Authors and Affiliations

Lucas G. S. Jeub[1] · Giovanni Colavizza[1,2] · Xiaowen Dong[3] · Marya Bazzi[1,4] · Mihai Cucuringu[1,5]

Giovanni Colavizza
g.colavizza@uva.nl

Xiaowen Dong
xdong@robots.ox.ac.uk

Marya Bazzi
marya.bazzi@warwick.ac.uk

Mihai Cucuringu
mihai.cucuringu@stats.ox.ac.uk

[1] The Alan Turing Institute, 96 Euston Road, London NW1 2DB, UK

[2] Institute for Logic, Language and Computation (ILLC), University of Amsterdam, Science Park 107, Amsterdam 1098 XG, Netherlands

[3] Department of Engineering Science, University of Oxford, Eagle House, Walton Well Road, Oxford OX2 6ED, UK

[4] Department of Mathematics, University of Warwick, Coventry CV4 7HP, UK

[5] Department of Statistics & Mathematical Institute, University of Oxford, 24-29 St Giles', Oxford OX1 3LB, UK