Graph-based Semi-Supervised & Active Learning for Edge Flows

Junteng Jia
Cornell University
jj585@cornell.edu

Michael T. Schaub
Massachusetts Institute of Technology
University of Oxford
mschaub@mit.edu

Santiago Segarra
Rice University
segarra@rice.edu

Austin R. Benson
Cornell University
arb@cs.cornell.edu

ABSTRACT

We present a graph-based semi-supervised learning (SSL) method for learning edge flows defined on a graph. Specifically, given flow measurements on a subset of edges, we want to predict the flows on the remaining edges. To this end, we develop a computational framework that imposes certain constraints on the overall flows, such as (approximate) flow conservation. These constraints render our approach different from classical graph-based SSL for vertex labels, which posits that tightly connected nodes share similar labels and leverages the graph structure accordingly to extrapolate from a few vertex labels to the unlabeled vertices.

We derive bounds for our method's reconstruction error and demonstrate its strong performance on synthetic and real-world flow networks from transportation, physical infrastructure, and the Web. Furthermore, we provide two active learning algorithms for selecting informative edges on which to measure flow, which has applications for optimal sensor deployment. The first strategy selects edges to minimize the reconstruction error bound and works well on flows that are approximately divergence-free. The second approach clusters the graph and selects bottleneck edges that cross cluster-boundaries, which works well on flows with global trends.

ACM Reference Format:
Junteng Jia, Michael T. Schaub, Santiago Segarra, and Austin R. Benson. 2019. Graph-based Semi-Supervised & Active Learning for Edge Flows. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3292500.3330872

1 INTRODUCTION

Semi-supervised learning (SSL) has been widely studied for large-scale data mining applications, where the labeled data are often difficult, expensive, or time consuming to obtain [36, 40]. SSL utilizes both labeled and unlabeled data to improve prediction accuracy by enforcing a *smoothness* constraint with respect to the intrinsic structure among all data samples. Graph-based SSL is an important branch of semi-supervised learning. It encodes the structure of data points with a similarity graph, where each vertex is a data sample and each edge is the similarity between a pair of vertices (Fig. 1, left). Such similarity graphs can either be derived from actual relational data or be constructed from data features using k-nearest-neighbors, ϵ-neighborhoods or Gaussian Random Fields [21, 39]. Graph-based SSL is especially suited for learning problems that are naturally defined on the *vertices* of a graph, including social networks [2], web networks [23], and co-purchasing networks [14].

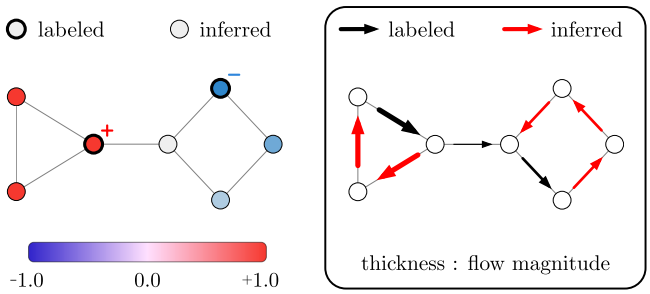

Figure 1: Left: classical graph-based semi-supervised learning for vertex labels. Right: Our framework of graph-based semi-supervised learning for edge flows.

However, in many complex networks, the behavior of interest is a dynamical process on the *edges* [30], such as a flow of energy, signal, or mass. For instance, in transportation networks, we typically monitor the traffic on roads (edges) that connect different intersections (vertices). Other examples include energy flows in power grids, water flows in water supply networks, and data packets flowing between autonomous systems. Similar to vertex-based data, edge flow data needs to be collected through dedicated sensors or special protocols and can be expensive to obtain. Although graph-theoretical tools like the line-graph [15] have been proposed to analyze graph-based data from an edge-perspective [1, 10], the problem of semi-supervised learning for edge flows has so far received little attention, despite the large space of applications

Here we consider the problem of semi-supervised learning for edge flows for networks with fixed topology. Given a network with a vertex set $\mathcal{V}$ and edge set $\mathcal{E}$, the (net) edge flows can be considered as real-valued alternating functions $f : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$, such that:

$$f(i,j) = \begin{cases} -f(j,i), & \forall\ (i,j) \in \mathcal{E} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

As illustrated in Fig. 1, this problem is related to—yet fundamentally different from—SSL in the vertex-space. Specifically, a key

KDD '19, August 4–8, 2019, Anchorage, AK, USA

ACM ISBN 978-1-4503-6201-6/19/08…$15.00
https://doi.org/10.1145/3292500.3330872

assumption underlying classical vertex-based SSL is that tightly-knit vertices are likely to share similar labels. This is often referred to as the smoothness or cluster assumption [6].

However, naively translating this notion of smoothness to learn edge flows leads to sub-optimal algorithms. As we will show in the following sections, applying classical vertex-based SSL to a line-graph [15], which encodes the adjacency relationships between *edges*, often produces worse results than simply ignoring the observed edges. Intuitively, the reason is that smoothness is not the right condition for flow data: unlike a vertex label, each edge flow carries an orientation which is represented by the sign of its numerical flow value. Enforcing smoothness on the line-graph requires the flow values on adjacent edges to be numerically close, which does *not* reflect any insight into the underlying physical system.

To account for the different nature of edge data, we assume different kinds of SSL constraints for edge flows. Specifically, we focus on flows that are almost conserved or *divergence-free*—the total amount of flow that enters a vertex should approximately equal the flow that leaves. Given an arbitrary network and a set of labeled edge flows $\mathcal{E}^{\mathrm{L}}$, the unlabeled edge flows on $\mathcal{E}^{\mathrm{U}}$ are inferred by minimizing a cost function based on the edge Laplacian $\mathbf{L}_e$ that measures divergence at all vertices. We provide the perfect recovery condition for strictly divergence-free flows, and derive an upper bound for the reconstruction error when a small perturbation is added. We further show that this minimization problem can be converted into a linear least-squares problem and thus solved efficiently. Our method substantially outperforms two competing baselines (including the line-graph approach) as measured by the Pearson correlation coefficient between the inferred edge flows and the observed ground truth on a variety of real-world datasets.

We further consider *active* semi-supervised learning for edge flows, where we aim to select a fraction of edges that is most informative for inferring the flows on the remaining edges. An important application of this problem is optimal sensor placement, where we want to deploy flow sensors on a limited number of edges such that the reconstructed edge flows are as accurate as possible. We propose two active learning strategies and demonstrate substantial performance gains over random edge selection. Finally, we discuss how our methods can be extended to other types of structured edge flows by highlighting connections with algebraic topology. We summarize our main contributions as follows: (1) a semi-supervised learning method for edge flows; (2) two active learning algorithms for choosing informative edges; and (3) analysis of real-world data that demonstrate the superiority of our method to alternatives.

2 METHODOLOGY

Given an undirected network with vertex set $\mathcal{V}$, edge set $\mathcal{E}$, and a labeled set of edge flows on $\mathcal{E}^{\mathrm{L}} \subseteq \mathcal{E}$, our goal is to predict the unlabeled edge flows $\mathcal{E}^{\mathrm{U}} \equiv \mathcal{E} \backslash \mathcal{E}^{\mathrm{L}}$. Although our assumption about the data in this problem is different from classical graph-based SSL for vertex labels, the associated matrix computations in the two problems have striking similarities. In fact, we show in Section 5 that the similarity is mediated by deep connections to algebraic topology. In this section, we first review classical vertex-based SSL and then show how it relates to our edge-based method. Table 1 summarizes notation used throughout the paper.

Background on Graph-based SSL for Vertex Labels. In the SSL problem for vertex labels, we are given the labels of a subset of vertices $\mathcal{V}^{\mathrm{L}}$, and our goal is to find a label assignment of the unlabeled vertices $\mathcal{V}^{\mathrm{U}}$ such that the labels vary smoothly across neighboring vertices. Formally, this notion of smoothness (or the deviation from it, respectively) can be defined via a loss function of the form[1]

$$\|\mathbf{B}^\intercal \mathbf{y}\|^2 = \textstyle\sum_{(i,j)\in\mathcal{E}} (y_i - y_j)^2, \tag{2}$$

where $\mathbf{y}$ is the vector containing vertex labels, and $\mathbf{B} \in \mathbb{R}^{n\times m}$ is the incidence matrix of the network, defined as follows. Consider the edge set $\mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_r, \ldots, \mathcal{E}_m\}$ and, without loss of generality, choose a reference orientation for every edge such that it points from the vertex with the smaller index to the vertex with the larger index. Then the incidence matrix $\mathbf{B}$ is defined as

$$B_{kr} = \begin{cases} 1, & \mathcal{E}_r \equiv (i,j),\ k = i,\ i < j \\ -1, & \mathcal{E}_r \equiv (i,j),\ k = j,\ i < j \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The loss function in Eq. (2) is the the sum-of-squares label difference between all connected vertices. The loss can be written compactly as $\|\mathbf{B}^\intercal \mathbf{y}\|^2 = \mathbf{y}^\intercal \mathbf{L} \mathbf{y}$ in terms of the graph Laplacian $\mathbf{L} = \mathbf{B}\mathbf{B}^\intercal$.

In vertex-based SSL, unknown vertex labels are inferred by minimizing the quadratic form $\mathbf{y}^\intercal \mathbf{L} \mathbf{y}$ with respect to $\mathbf{y}$ while keeping the labeled vertices fixed.[2] Using $\hat{y}_i$ to denote an observed label on vertex i, the optimization problem is:

$$\mathbf{y}^* = \arg\min_{\mathbf{y}} \|\mathbf{B}^\intercal \mathbf{y}\|^2 \qquad \text{s.t.} \quad y_i = \hat{y}_i,\ \forall \mathcal{V}_i \in \mathcal{V}^{\mathrm{L}}. \tag{4}$$

For connected graphs with more edges than vertices ($m > n$), Eq. (4) has a unique solution provided at least one vertex is labeled.

2.1 Graph-Based SSL for Edge Flows

We now consider the SSL problem for edge flows. The edge flows over a network can be represented with a vector $\mathbf{f}$, where $\mathrm{f}_r > 0$ if the flow orientation on edge r aligns with its reference orientation and $\mathrm{f}_r < 0$ otherwise. In this sense, we are only accounting for the *net flow* along an edge. We denote the ground truth (measured) edge flows in the network as $\hat{\mathbf{f}}$. To impose a flow conservation assumption for edge flows, we consider the divergence at each vertex, which is the sum of outgoing flows minus the sum of incoming flows at a vertex. For arbitrary edge flows $\mathbf{f}$, the divergence on a vertex i is

$$(\mathbf{B}\mathbf{f})_i = \sum_{\mathcal{E}_r \in \mathcal{E}\,:\, \mathcal{E}_r \equiv (i,j), i<j} \mathrm{f}_r - \sum_{\mathcal{E}_r \in \mathcal{E}\,:\, \mathcal{E}_r \equiv (j,i), j<i} \mathrm{f}_r.$$

To create a loss function for edge flows that enforces a notion of flow-conservation, we use the sum-of-squares vertex divergence:

$$\|\mathbf{B}\mathbf{f}\|^2 = \mathbf{f}^\intercal \mathbf{B}^\intercal \mathbf{B} \mathbf{f} = \mathbf{f}^\intercal \mathbf{L}_e \mathbf{f}. \tag{5}$$

Here $\mathbf{L}_e = \mathbf{B}^\intercal \mathbf{B}$ is the so-called edge Laplacian matrix. Interestingly, the loss function for penalizing divergence contains the transpose of the incidence matrix $\mathbf{B}$, which appeared in the measure of smoothness in the vertex-based problem [cf. Eq. (2)]. However, unlike the case for smooth vertex labels, requiring $\mathbf{f}^\intercal \mathbf{L}_e \mathbf{f} = 0$ is actually under-constrained, i.e., even when more than one edge

[1] All norms for vectors and matrices in this paper are the 2-norm.

[2] This is the formulation of Zhu, Ghahramani, and Lafferty [39]. There are other graph-based SSL methods [36]; however, most of them employ a similar loss-function based on variants of the graph Laplacian $\mathbf{L}$.

Table 1: Summary of notation used throughout the paper.

Symbol	Description
$n \in \mathbb{N}$	$\lvert\mathcal{V}\rvert$ = number of vertices
$m \in \mathbb{N}$	$\lvert\mathcal{E}\rvert$ = number of edges
$o \in \mathbb{N}$	$\lvert\mathcal{T}\rvert$ = number of triangles
$m^{\mathrm{L}} \in \mathbb{N}$	$\lvert\mathcal{E}^{\mathrm{L}}\rvert$ = number of labeled edges
$m^{\mathrm{U}} \in \mathbb{N}$	$\lvert\mathcal{E}^{\mathrm{U}}\rvert$ = number of unlabeled edges
$c \in \mathbb{N}$	$m - n + 1$ = number of independent cycles
$i, j, k \in \mathbb{N}$	vertex index
$r, s, t \in \mathbb{N}$	edge index
$u \in \mathbb{N}$	triangle index
$\alpha, \beta, \gamma \in \mathbb{N}$	index for spectral coefficient
$\mathbf{y}, \hat{\mathbf{y}} \in \mathbb{R}^n$	vertex labels, ground truth vertex labels
$\mathbf{f}, \hat{\mathbf{f}} \in \mathbb{R}^m$	edge flows, ground truth edge flows
$\mathbf{w} \in \mathbb{R}^o$	function defined on triangles
$\mathbf{B} \in \mathbb{R}^{n \times m}$	node-edge incidence matrix (see Eq. (3))
$\mathbf{C} \in \mathbb{R}^{m \times o}$	edge-triangle curl matrix (see Eq. (15))
$\mathbf{L} \in \mathbb{R}^{n \times n}$	Laplacian $\mathbf{L} = \mathbf{B}\mathbf{B}^\intercal$
$\mathbf{L}_e \in \mathbb{R}^{m \times m}$	edge Laplacian $\mathbf{L}_e = \mathbf{B}^\intercal\mathbf{B}$

is labeled, many different divergence-free edge-flow assignments may exist that induce zero loss. We thus propose to regularize the problem and solve the following constrained optimization problem:

$$\mathbf{f}^* = \arg\min_{\mathbf{f}} \|\mathbf{B}\mathbf{f}\|^2 + \lambda^2 \cdot \|\mathbf{f}\|^2 \qquad \text{s.t.} \quad \mathrm{f}_r = \hat{\mathrm{f}}_r,\ \forall \mathcal{E}_r \in \mathcal{E}^{\mathrm{L}}. \quad (6)$$

The first term in the objective function is the loss, while the second term is a regularizer that guarantees a unique optimal solution.

Computation. The equality constraints in Eq. (6) can be eliminated by reducing the number of free variables. Let $\mathbf{f}^0$ be a trivial feasible point for Eq. (6) where $\mathrm{f}^0_r = \hat{\mathrm{f}}_r$ if $r \in \mathcal{E}^{\mathrm{L}}$ and $\mathrm{f}^0_r = 0$ otherwise. Moreover, denote the set of indices for unlabeled edges as $\mathcal{E}^{\mathrm{U}} = \{\mathcal{E}^{\mathrm{U}}_1, \mathcal{E}^{\mathrm{U}}_2, \ldots, \mathcal{E}^{\mathrm{U}}_{m^{\mathrm{U}}}\}$. We define the expansion operator $\mathbf{\Phi}$ as a linear map from $\mathbb{R}^{m^{\mathrm{U}}}$ to $\mathbb{R}^m$ given by $\Phi_{rs} = 1$ if $\mathcal{E}_r = \mathcal{E}^{\mathrm{U}}_s$ and 0 otherwise. Let $\mathbf{f}^{\mathrm{U}} \in \mathbb{R}^{m^{\mathrm{U}}}$ be the edge flows on the unlabeled edges. Any feasible point for Eq. (6) can be written as $\mathbf{f}^0 + \mathbf{\Phi}\mathbf{f}^{\mathrm{U}}$, and the original problem can be converted to a linear least-squares problem:

$$\mathbf{f}^{\mathrm{U}*} = \arg\min_{\mathbf{f}^{\mathrm{U}}} \left\| \begin{bmatrix} \mathbf{B}\mathbf{\Phi} \\ \lambda \cdot \mathbf{I} \end{bmatrix} \mathbf{f}^{\mathrm{U}} - \begin{bmatrix} -\mathbf{B}\mathbf{f}^0 \\ 0 \end{bmatrix} \right\|^2. \quad (7)$$

Typically, $\mathbf{B}$ is a large sparse matrix. Thus, the least-squares problem in Eq. (7) can be solved with iterative methods such as LSQR [27] or LSMR [11], which is guaranteed to converge in m^{U} iterations. Those iterative solvers use sparse matrix-vector multiplication as subroutine, with $O(m)$ computational cost per iteration. By choosing $\lambda > 0$, Eq. (7) can be made well-conditioned, and the iterative methods will only take a small number of iterations to converge.

2.2 Spectral Graph Theory Interpretations

We first briefly review graph signal processing in the vertex-space before introducing similar tools to deal with edge flows. The eigenvectors of the Laplacian matrix have been widely used in graph signal processing for vertex labels, since the corresponding eigenvalues carry a notion of frequency that provides a sound mathematical and intuitive basis for analyzing functions on vertices [26, 32]. The spectral decomposition of the graph Laplacian matrix is $\mathbf{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^\intercal$.

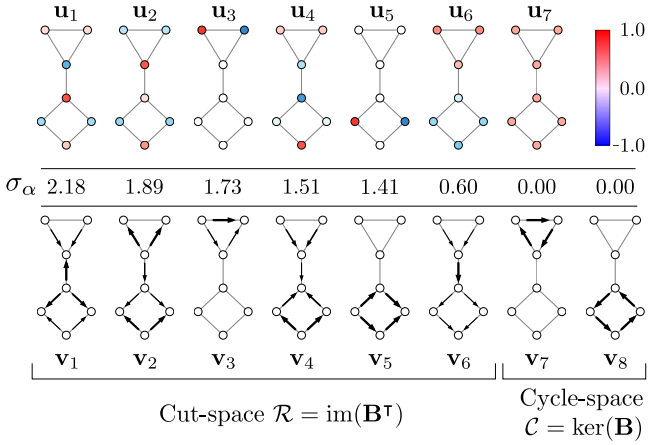

Figure 2: Singular vectors for the incidence matrix B of an example graph. Top: the left singular vectors form a basis for vertex labels. Numerical values are encoded by color of the vertices. Middle: singular values represent the "frequencies" of left singular vectors or the divergences of right singular vectors. Bottom: right singular vectors form a basis for edge flows, where the arrow points to the flow direction and the edge-width encodes the magnitude of the flow.

Because $\mathbf{L} = \mathbf{B}\mathbf{B}^\intercal$, the orthonormal basis $\mathbf{U} \in \mathbb{R}^{n \times n}$ for vertex labels is formed by the *left* singular vectors of the incidence matrix: $\mathbf{B} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^\intercal$, where $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$ is the diagonal matrix of ordered singular values with $m - n$ columns of zero-padding on the right, and the right singular vectors $\mathbf{V} \in \mathbb{R}^{m \times m}$ is an orthonormal basis for edge flows. To simplify our discussion, we will say that the basis vectors in the last $m - n$ columns of $\mathbf{V}$ also have singular value 0.

The divergence-minimizing objective in Eq. (6) can be rewritten in terms of the right singular vectors of $\mathbf{B}$, thus providing a formal connection between the vertex-based and the edge-based SSL problem. Let $\mathbf{p} = \mathbf{V}^\intercal\mathbf{f} \in \mathbb{R}^m$ represent the *spectral coefficients* of $\mathbf{f}$ expressed in terms of the basis $\mathbf{V}$. Then, we can rewrite Eq. (6) as

$$\begin{aligned} \mathbf{f}^* &= \mathbf{V} \cdot \arg\min_{\mathbf{p}} (\mathbf{V}\mathbf{p})^\intercal \mathbf{B}^\intercal\mathbf{B}\,(\mathbf{V}\mathbf{p}) + \lambda^2 \cdot (\mathbf{V}\mathbf{p})^\intercal(\mathbf{V}\mathbf{p}) \\ &= \mathbf{V} \cdot \arg\min_{\mathbf{p}} \mathbf{p}^\intercal \left(\mathbf{\Sigma}^\intercal\mathbf{\Sigma} + \lambda^2 \cdot \mathbf{I}\right)\mathbf{p} \\ &= \mathbf{V} \cdot \arg\min_{\mathbf{p}} \lambda^2 \cdot \sum_{\alpha} \frac{\sigma_\alpha^2 + \lambda^2}{\lambda^2} p_\alpha^2 \\ &\qquad \text{s.t.} \quad (\mathbf{V}\mathbf{p})_r = \hat{\mathrm{f}}_r,\ \forall \mathcal{E}_r \in \mathcal{E}^{\mathrm{L}}, \end{aligned} \quad (8)$$

which minimizes the weighted sum-of-square of the spectral coefficients under equality constraints for measured edge flows.

Signal smoothness, cut-space, and cycle space. The connection between the vertex and edge-based problem is in fact not just a formal relationship, but can be given a clear (physical) interpretation. By construction $\mathbf{V}$ is a complete orthonormal basis for the space of edge flows (here identified with $\mathbb{R}^m$). This space can be decomposed into two orthogonal subspaces (see also Fig. 2).

The first subspace is the *cut-space* $\mathcal{R} = \mathrm{im}(\mathbf{B}^\intercal)$ [15], spanned by the singular vectors $\mathbf{V}_{\mathcal{R}}$ associated with nonzero singular values. The space $\mathcal{R}$ is also called the space of gradient flows, since any vector may be written as $\mathbf{B}^\intercal\mathbf{y}$, where $\mathbf{y}$ is a vector of vertex scalar potentials that induce a gradient flow. The second subspace is the

cycle-space $C = \ker(\mathbf{B})$ [15], spanned by the remaining right singular vectors $\mathbf{V}_C$ associated with zero singular values. Note that any vector $\mathbf{f} \in C$ corresponds to a circulation of flow, and will induce zero cost in our loss function Eq. (5). In fact, for a connected graph, the number of right singular vectors with zero singular values equals $c = m - n + 1$, which is the number of independent cycles in the graph. We denote the spectral coefficients for basis vectors in these two spaces as $\mathbf{p}_{\mathcal{R}} \in \mathbb{R}^{m-c}$ and $\mathbf{p}_C \in \mathbb{R}^c$.

Let $\mathbf{u}_\alpha, \sigma_\alpha, \mathbf{v}_\alpha$ denote a triple of a left singular vector, singular value, and right singular vector. The singular values $\mathbf{u}_\alpha^\mathsf{T}\, \mathbf{L}\, \mathbf{u}_\alpha = \sigma_\alpha^2$ provide a notion of "unsmoothness" of basis vector $\mathbf{u}_\alpha$ representing vertex labels, while $\mathbf{v}_\alpha^\mathsf{T}\, \mathbf{L}_e\, \mathbf{v}_\alpha = \sigma_\alpha^2$ gives the sum-of-squares divergence of basis vector $\mathbf{v}_\alpha$ representing edge flows. As an example, Fig. 2 displays the left and right singular vectors of a small graph. The two singular basis vectors $\mathbf{v}_7$ and $\mathbf{v}_8$ associated with zero singular values correspond to cyclic edge flows in the network. The remaining flows $\mathbf{v}_i$, $i = 1, \ldots, 6$—corresponding to non-zero singular values—all have a non-zero divergence. Note also how the left singular vectors $\mathbf{u}_i$ associated with non-zero singular values give rise to the right singular vectors $\mathbf{v}_i = 1/\sigma_i \cdot \mathbf{B}^\mathsf{T}\mathbf{u}_i$ for $i = 1, \ldots 6$. As the singular vectors $\mathbf{u}_i$ can be interpreted as potential on the nodes, this highlights that the cut-space is indeed equivalent to the space of gradient flows (note that $\mathbf{u}_7$ induces no gradient).

2.3 Exact and perturbed recovery

From the above discussion, we can derive an exact recovery condition for the edge flows in the divergence-free setting.

Lemma 2.1. *Assume the ground truth flows are divergence-free. Then as $\lambda \to 0$, the solution of Eq. (8) can exactly recover the ground truth from some labeled edge set $\mathcal{E}^{\mathrm{L}}$ with cardinality $c = m - n + 1$.*

Proof. If the ground truth edge flows are divergence free (cyclic), the spectral coefficients of the basis vectors with non-zero singular values must be zero. Recall that $\mathbf{p}_C \in \mathbb{R}^c$ are the spectral coefficients of a basis $\mathbf{V}_C$ in the cycle-space, then the ground truth edge flows can be written as $\hat{\mathbf{f}} = \mathbf{V}_C \mathbf{p}_C$ (the singular vectors $\mathbf{V}_C$ that form the basis of the the cycle space are not unique as the singular values are "degenerate"; any orthogonal transformation is also valid). On the other hand, in the limit $\lambda \to 0$, the spectral coefficients of basis vectors with non-zero singular values have infinite weights and are forced to zero [cf. Eq. (8)]. Therefore, by choosing the set of labeled edges corresponding to $c = m - n + 1$ linearly independent rows from $\mathbf{V}_C$, the ground truth $\hat{\mathbf{f}}$ is the unique optimal solution. □

Furthermore, when a perturbation is added to divergence-free edge flows $\mathbf{f}$, the reconstruction error can be bounded as follows.

Theorem 2.2. *Let $\mathbf{V}_C^{\mathrm{L}}$ denote c linearly independent rows of the $\mathbf{V}_C$ that correspond to labeled edges. If the divergence-free edge flows $\mathbf{f}$ are perturbed by δ, then as $\lambda \to 0$, the reconstruction error of the proposed algorithm is bounded by $[\sigma_{\min}^{-1}(\mathbf{V}_C^{\mathrm{L}}) + 1] \cdot \|\delta\|$.*

Proof. The ground truth edge flows can be written as,

$$\hat{\mathbf{f}} = \mathbf{f} + \delta = \begin{bmatrix} \mathbf{f}^{\mathrm{L}} \\ \mathbf{f}^{\mathrm{U}} \end{bmatrix} + \begin{bmatrix} \delta^{\mathrm{L}} \\ \delta^{\mathrm{U}} \end{bmatrix}, \tag{9}$$

where $\mathbf{f}^{\mathrm{L}}$, $\mathbf{f}^{\mathrm{U}}$ are the divergence-free edge flows on labeled and unlabeled edges, while δ^{L}, δ^{U} are the corresponding perturbations.

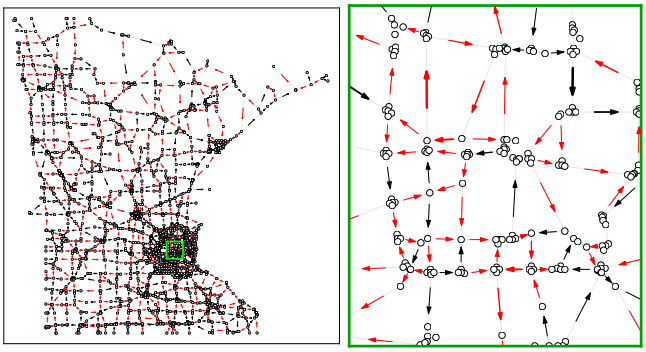

Figure 3: Synthetic traffic flow in Minnesota road network; 40% of the edges are labeled and their flow is plotted in black. The remaining red edge flows are inferred with our algorithm. The width of each arrow is proportional to the magnitude of flow on the edge. The Pearson correlation coefficient between the inferred flows $\mathbf{f}^*$ and the ground truth $\hat{\mathbf{f}}$ is 0.956.

Further, the reconstructed edge flows from Eq. (8) are given by $\mathbf{V}_C(\mathbf{V}_C^{\mathrm{L}})^{-1}(\mathbf{f}^{\mathrm{L}} + \delta^{\mathrm{L}})$. Therefore, we can bound the norm of the reconstruction error as follows:

$$\begin{aligned} \|\mathbf{V}_C(\mathbf{V}_C^{\mathrm{L}})^{-1}(\mathbf{f}^{\mathrm{L}} + \delta^{\mathrm{L}}) - (\mathbf{f} + \delta)\| &= \|\mathbf{V}_C(\mathbf{V}_C^{\mathrm{L}})^{-1}\delta^{\mathrm{L}} - \delta\| \\ &\le \|\mathbf{V}_C(\mathbf{V}_C^{\mathrm{L}})^{-1}\delta^{\mathrm{L}}\| + \|\delta\| = \|(\mathbf{V}_C^{\mathrm{L}})^{-1}\delta^{\mathrm{L}}\| + \|\delta\| \\ &\le \|(\mathbf{V}_C^{\mathrm{L}})^{-1}\| \cdot \|\delta^{\mathrm{L}}\| + \|\delta\| \le [\|(\mathbf{V}_C^{\mathrm{L}})^{-1}\| + 1] \cdot \|\delta\|. \end{aligned} \tag{10}$$

The first equality in Eq. (10) comes from Lemma 2.1, and the second equality is due to the orthonormal columns of $\mathbf{V}_C$. Finally, the norm of a matrix equals its largest singular value, and the singular values of the matrix inverse are the reciprocals of the singular values of the original matrix. Therefore, we can rewrite Eq. (10) as follows

$$\begin{aligned} [\|(\mathbf{V}_C^{\mathrm{L}})^{-1}\| + 1] \cdot \|\delta\| &= [\sigma_{\max}((\mathbf{V}_C^{\mathrm{L}})^{-1}) + 1] \cdot \|\delta\| \\ &= [\sigma_{\min}^{-1}(\mathbf{V}_C^{\mathrm{L}}) + 1] \cdot \|\delta\|. \end{aligned} \tag{11}$$

□

3 SEMI-SUPERVISED LEARNING RESULTS

Having discussed the theory and computations underpinning our method, we now examine its application on a collection of networks with synthetic and real-world edge flows. As our method is based on a notion of divergence-free edge flows, naturally we find the most accurate edge flow estimates when this assumption approximately holds. For experiments in this section, the labeled sets of edges are chosen uniformly at random. In Section 4, we provide active learning algorithms for selecting *where* to measure.

3.1 Learning Synthetic Edge Flows

Flow Network Setup. In our first set of experiments, the network topology comes from real data, but we use synthetic flows to demonstrate our method. Later, we examine edge flows from real-world measurements. We use the following four network topologies in our synthetic flow examples: (1) The Minnesota road network where edges are roads and vertices are intersections ($n = 2642, m = 3303$) [13]; (2) The US power grid network from KONECT, where vertices are power stations or individual consumers and edges are

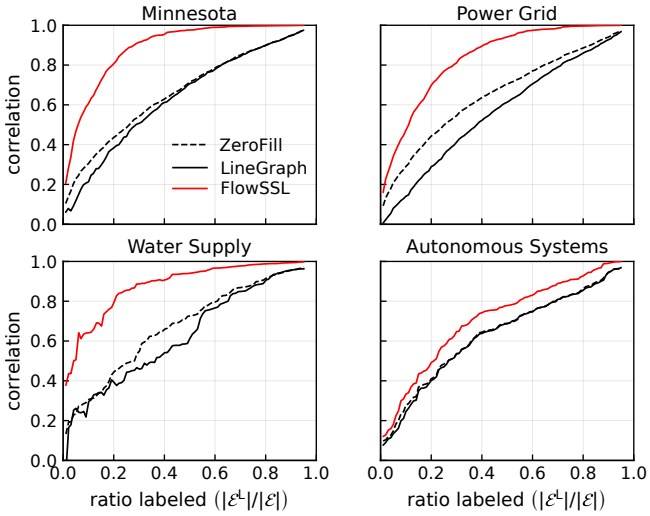

Figure 4: Graph-based SSL for synthetic flows. The plots show the correlation between the estimated flow vector $\mathbf{f}^*$ and the synthetic ground truth edge flows $\hat{\mathbf{f}}$ as a function of the ratio of labeled edges.

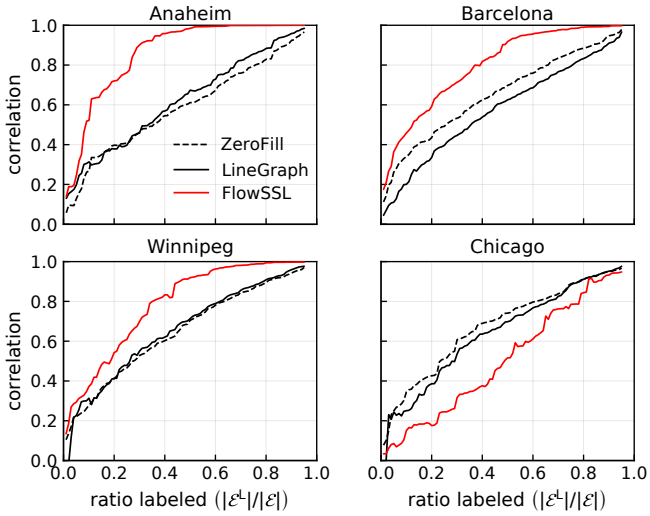

Figure 5: Graph-based SSL for real-world traffic flows. We plot the correlation between the estimated flow $\mathbf{f}^*$ and the ground truth $\hat{\mathbf{f}}$ measured in four transportation networks, as a function of the ratio of labeled edges. Our FlowSSL out-performs the baselines except in Chicago, which has a large flow component in the cut space (Fig. 6).

transmission lines ($n = 4941, m = 6593$) [22]; (3) The water irrigation network of Balerma city in Spain where vertices are water supplies or hydrants and edges are water pipes ($n = 447, m = 454$) [28]; and (4) An autonomous system network ($n = 520, m = 1280$) [24].

For each network, we first perform an SVD on its incidence matrix to get the edge-space basis vectors $\mathbf{V}$. The synthetic edge flows are then created by specifying the spectral coefficients $\mathbf{p}$, i.e., the mixture of these basis vectors. Recall from Section 2.2 that the singular values associated with the basis vectors measure the magnitude of the divergence of each of these flow vectors. To obtain a divergence-free flow, the spectral coefficients for all basis vectors $\mathbf{V}_{\mathcal{R}}$ spanning the cut space (associated with a nonzero singular value) should thus be set to zero. However, to mimic the fact that most real-world edge flows are not perfectly divergence-free, we do not set the spectral coefficients for the basis vectors in $\mathbf{V}_{\mathcal{R}}$ to zero. Instead, we create synthetic flows with spectral coefficients for each basis vector (indexed by α) that are inversely proportional to the associated singular values σ_α:

$$p_\alpha = \frac{b}{\sigma_\alpha + \epsilon}, \tag{12}$$

where b is a parameter that controls the overall magnitude of the edge flows and ϵ is a damping factor. We choose $b = 0.02$, $\epsilon = 0.1$ in all examples shown in this paper.

Performance Measurement and Baselines. Using the synthetic edge flows as our ground truth $\hat{\mathbf{f}}$, we conduct numerical experiments by selecting a fraction of edges *uniformly at random* as the labeled edges $\mathcal{E}^L$, and using our method to infer the edge flow on the unlabeled edges $\mathcal{E}^U$. To quantify the accuracy of the inferred edge flows, we use the Pearson correlation coefficient ρ between the ground truth edge flows $\hat{\mathbf{f}}$ and the inferred edge flow $\mathbf{f}^*$.[3] The regulation parameter λ in Eq. (6) is 0.1. To illustrate the results, Fig. 3 shows inferred traffic flows on the Minnesota road network.

[3]Consistent results are obtained with other accuracy metrics, e.g., the relative L^2 error.

We compare our algorithm against two baselines. First, the *ZeroFill* baseline simply assigns 0 edge flows to all unlabeled edges. Second, the *LineGraph* baseline uses a line-graph transformation of the network and then applies standard vertex-based SSL on the resulting graph. More specifically, the original network is transformed into an undirected line-graph, where there is a vertex for each edge in the original network; two vertices in the line-graph are connected if the corresponding two edges in the original network share a vertex. Flow values (including sign) on the edges in the original network are the labels on the corresponding vertices in the transformed line-graph. Unlabeled edge flows are then inferred with a classical vertex-based SSL algorithm on the line-graph [39].

Results. We test the performance of our algorithm *FlowSSL* and the two baseline methods for different ratios of labeled edges (Fig. 4). The LineGraph approach performs no better than ZeroFill. This should not be surprising, since the LineGraph approach does not interpret the sign of an edge flow as an orientation but simply as part of a numerical label. On the other hand, our algorithm out-performs both baselines considerably. *FlowSSL* works especially well on the Minnesota road network and the Balerma water supply network, which have small average degree $\langle d \rangle$. The intuitive reason is that the dimension of the cycle space is $m-n+1 = n(\langle d \rangle/2-1)+1$; therefore, low-degree graphs have fewer degrees of freedom associated with a zero penalty in the objective Eq. (6).

3.2 Learning Real-World Traffic Flows

We now consider actual, measured flows rather than synthetically generated flows. Accordingly, our assumption of approximately divergence-free flows may or may not be valid. We consider transportation networks and associated measured traffic flows from four cities (Anaheim, Barcelona, Winnipeg, and Chicago) [35]. To test our method, we repeat the same procedure we used for processing

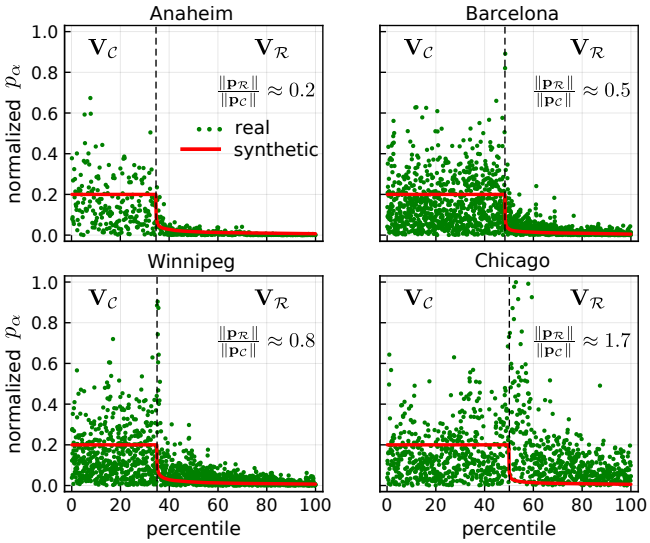

Figure 6: The normalized spectral coefficients of real-world and synthetic edge flows (Eq. (12)). The spectral coefficients are ordered by increasing singular values and plotted as a function of the percentile ranking. The basis vectors in the cycle-space $\mathbf{V}_C$ (lower percentile) all have zero singular values. The real-world traffic flow's spectral coefficients are taken in absolute value and normalized so that the root-mean-square of $\mathbf{p}_C$ (the spectral coefficients in cycle-space) equals 0.2. The different rates of decay in spectral coefficients leads to different performance of our method (Fig. 5).

synthetic flows in Section 3.1 with these real-world measured flows. Figure 5 displays the results.

Our algorithm performs substantially better than the baselines on three out of four transportation networks with real-world flows. It performs worse than the baseline on the Chicago road network. To understand this phenomenon, we compare the spectral coefficients of the real-world traffic flows in four cities with the "damped-inverse" synthetic spectral coefficients from Eq. (12) (see Fig. 6). We immediately see that the real-world spectral coefficients $\hat{p}_\alpha$ do not significantly decay with increasing singular value in the Chicago network, in contrast to the other networks. Formally, we measure how much the real-world edge flows deviate from our divergence-free assumption by computing the spectral ratio $\|\mathbf{p}_\mathcal{R}\|/\|\mathbf{p}_C\|$ between the norms of the spectral coefficients in the cut-space and the cycle-space. The ratios in the first three cities are all below 1.0, indicating divergence-free flow is the dominating component. However, the spectral ratio of traffic flows in Chicago is approximately 1.7, which explains why our method fails to give accurate predictions. Moreover, in the Chicago network, the spectral coefficients $\hat{p}_\alpha$ with the largest magnitude are actually concentrated in the cut-space basis vectors (with smallest singular values). Later, we show how to improve our results by strategically choosing edges on which to measure flow, rather than selecting edges at random (Section 4.1).

3.3 Information Flow Networks

Thus far we have focused on networks embedded in space, where the edges represent some media through which physical units flow between the vertices. Now we demonstrate the applicability of our method to information networks by considering the edge-based SSL problem for predicting transitions among songs in a user's play sequence on Last.fm[4]. A user's play sequence consists of a chronologically ordered sequence of songs he/she listened to, and a song may repeatedly show up. Taking the playlist of a user, we represent each unique song as a vertex in graph, and we connect two vertices if they are adjacent somewhere in the playlist. The ground truth flows is constructed as follows: every time the user plays song A followed by song B, add one unit of flow from A to B. We similarly constructed a flow network that records the transition among the artists of songs. The flow networks constructed here are close to divergence-free, since every time a user transitions to a song or artist, he/she typically transition out by listening to other ones. We used the same set of experiments to evaluate flow prediction for these networks (Fig. 7). Our method outperforms the baselines, despite the flows are not from physical systems.

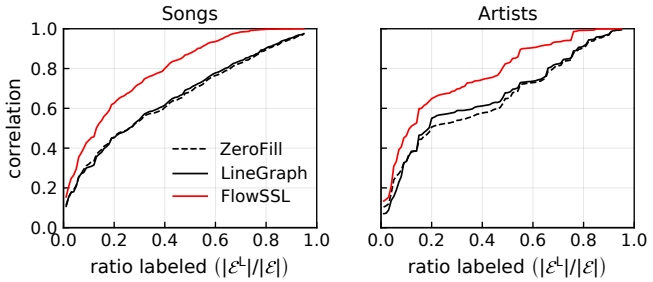

Figure 7: Graph-based SSL for real-world flows among songs and artists in a music playlists. The plots show the correlation between the estimated flow vector $\mathbf{f}^*$ and the ground truth $\hat{\mathbf{f}}$ on flow networks of songs and artists on a music streaming service. Even though the flows are not physical, FlowSSL method still outperforms the baselines.

4 ACTIVE SEMI-SUPERVISED LEARNING

We now focus on the problem of selecting the set of labeled edges that is most helpful in determining the overall edge flows in a network. While selecting the most informative set of labeled vertices has been well studied in the context of vertex-based semi-supervised learning [12, 18], active learning in the edge-space remains largely under-explored. Traffic flows are typically monitored by road detectors, but installation and maintenance costs often prevent the deployment of these detectors on the entire transportation network. In this scenario, solving the active learning problem in the edge-space enables us to choose an optimal set of roads to deploy sensors under a limited budget.

4.1 Two active learning algorithms

We develop two active semi-supervised learning algorithms for selecting edges to measure. These algorithms improve the robustness of our method for learning edge flows.

Rank-revealing QR (RRQR). According to Theorem 2.2, the upper bound of the reconstruction error decreases as the smallest singular value of $\mathbf{V}_C^L$ increases. Therefore, one strategy for selecting $\mathcal{E}^L$ is to choose m^L rows from $\mathcal{V}_0$ that maximize the smallest singular value of the resulting submatrix. This problem is known

[4]This dataset is from https://www.last.fm/.

as optimal column subset selection (maximum submatrix volume) and is NP-hard [7]. However, a good heuristic is the rank revealing QR decomposition (RRQR) [5], which computes

$$\mathbf{V}_C^\mathsf{T}\,\Pi = Q\begin{bmatrix} R_1 & R_2 \end{bmatrix}. \tag{13}$$

Here, Π is a permutation matrix that keeps R_1 well-conditioned. Each column permutation in Π corresponds to an edge, and the resulting edge set $\mathcal{E}^\mathrm{L}$ for active learning chooses the first m^L columns of Π. This approach is mathematically similar to graph clustering algorithms that use RRQR to select representative vertices for cluster centers [9]. The computational cost of RRQR is $O(m^3)$.

Recursive Bisection (RB). In many real-world flow networks, there exist a global trend of flows across different cluster of vertices. For example, traffic during morning rush hour flows from rural to urban regions or electricity flows from industrial power plants to residential households. The spectral projection of such global trends is concentrated on singular vectors $\mathbf{v} \in \mathbf{V}_\mathcal{R}$ with small singular values corresponding to gradient flows (e.g., $\mathbf{v}_6$ in Fig. 2), as was the case with the Chicago traffic flows (Fig. 6).

Building on this observation, our second active learning algorithm uses a heuristic recursive bisection (RB) approach for selecting labeled edges.[5] The intuition behind this heuristic is that edge flows on bottleneck-edges, which partition a network, are able to capture global trends in the networks' flow pattern. We start with an empty labeled set $\mathcal{E}^\mathrm{L}$, a target number of labeled edges m^L, and the whole graph as one single cluster. Next, we recursively partition the largest cluster in the graph with spectral clustering and add every edge that connects the two resulting clusters into $\mathcal{E}^\mathrm{L}$, until we reach the target number of labeled edges. Similar methods have been shown to be effective in semi-supervised active learning for vertex labels [18]; in these cases, the graph is first clustered, and then one vertex is selected from each cluster. While any other graph partitioning algorithm could be used and greedy recursive bisection approaches can be sub-optimal [33], we find that this simple methods works well in practice on our datasets, and its iterative nature is convenient for selecting a target number of edges. The computational cost of the recursive bisection algorithm is $O(m \log n)$.

4.2 Results

We repeat the experiments in Section 3 on traffic networks with labeled edges $\mathcal{E}^\mathrm{L}$ selected by our RRQR and RB algorithms. For comparison, the ZeroFill approach with randomly selected labeled edges is included as a baseline. Our RRQR algorithm outperforms both recursive bisection and random selection for networks with synthetic edge flows, where the divergence-free condition on vertices approximately holds (Fig. 8). However, for networks with real-world edges flows, RRQR performs poorly, indicating that the divergence-free condition is too strong an assumption (Fig. 9). In this case, our RB method consistently outperforms the baselines, especially for small numbers of labels.

To provide additional intuition for the RB algorithm, we plot the selected labeled edges and the final clusters in the Winnipeg and Chicago network when allowing 10% of edges to be labeled (Fig. 10). The correlation coefficients resulting from random edge selection and RB active learning are $\rho_\mathrm{rand} = 0.371$ and $\rho_\mathrm{RB} = 0.580$ for the Winnipeg road network, respectively. For the Chicago road network we obtain correlations of $\rho_\mathrm{rand} = 0.151$ and $\rho_\mathrm{RB} = 0.718$. Thus, the active learning strategy alleviates our prior issues with learning on the Chicago road network by strategically choosing where to measure edge flows.

[5] We call this algorithm recursive *bisection* although it does not necessarily gives two clusters with the same number of vertices.

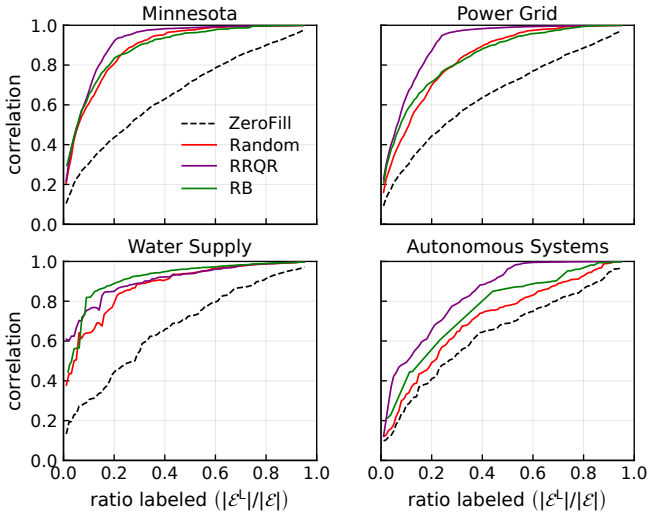

Figure 8: Graph-based semi-supervised active learning for *synthetic* flows. The plots show the Pearson correlation coefficients between the estimated flow vector $\mathbf{f}^*$ and the synthetic ground truth edge flows $\hat{\mathbf{f}}$ as a function of the ratio of labeled edges. Our rank-revealing QR (RRQR) active learning performs well on synthetic datasets.

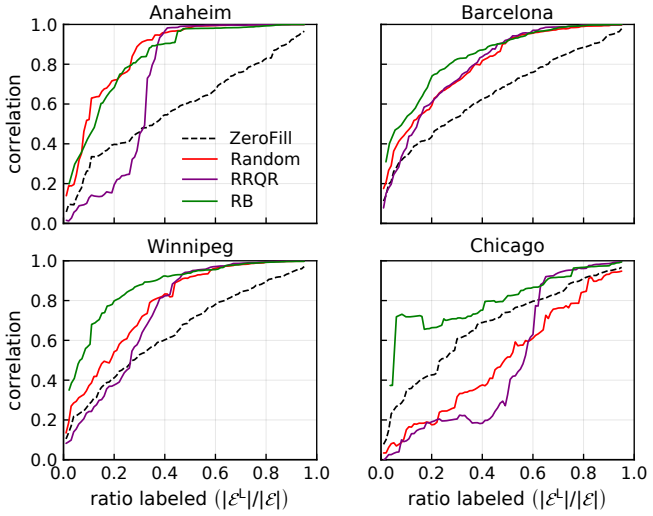

Figure 9: Graph-based semi-supervised active learning for *real-world* traffic flows. The plots show the Pearson correlation coefficients between the estimated flow vector $\mathbf{f}^*$ and the ground truth $\hat{\mathbf{f}}$ measured in four transportation networks, as a function of the ratio of labeled edges. With our Recursive Bisection (RB) active learning method to select edges, we now perform better than the baseline (ZeroFill) on the Chicago traffic dataset (cf. Fig. 5).

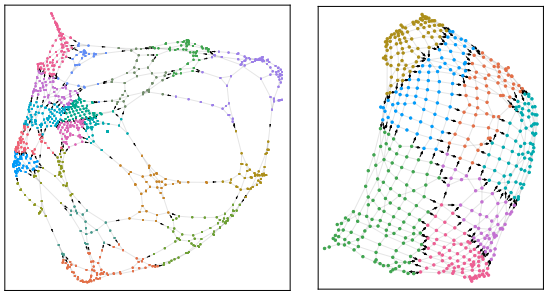

Figure 10: The clusters discovered by our recursive bisection algorithm in Winnipeg (left) and Chicago (right) road networks, where the vertex coordinates are computed by spectral-embedding. In each network, 10% of the edges are selected as the labeled set $\mathcal{E}^{\mathrm{L}}$ and the ground truth edge flows on those edges are plotted as black arrows.

5 EXTENSIONS TO CYCLIC-FREE FLOWS

Thus far, our working assumption has been that edge flows are approximately divergence-free. However, we might also be interested in the opposite scenario, if we study a system in which circular flows should *not* be present. Below we view our method through the lens of basic combinatorial Hodge theory. This viewpoint illuminates further how our previous method is a projection onto the space of divergence-free edge flows. By projecting into the complementary subspace, we can therefore learn flows that are (approximately) cycle-free. We highlight the utility of this idea through a problem of fair pricing of foreign currency exchange rates, where the cycle-free condition eliminates arbitrage opportunities.

5.1 The Hodge decomposition

Let $\mathbf{f}$ be a vector of edge flows. The *Hodge decomposition* provides an orthogonal decomposition of $\mathbf{f}$ [25, 29]:

$$\underbrace{\mathbf{f}}_{\text{edge flow}} = \overbrace{\mathbf{B}^{\mathsf{T}}\mathbf{y}}^{\text{gradient flow}} \oplus \overbrace{\underbrace{\mathbf{C}\mathbf{w}}_{\text{curl flow}} \oplus \underbrace{\mathbf{h}}_{\text{harmonic flow}}}^{\text{divergence-free flow}} \tag{14}$$

where the matrix $\mathbf{C} \in \mathbb{R}^{m \times o}$ (called the curl operator) maps edge flows to curl around a triangle,

$$C_{ru} = \begin{cases} 1, & \mathcal{T}_u \equiv (i,j,k),\ \mathcal{E}_r \equiv (i,j),\ i<j<k \\ 1, & \mathcal{T}_u \equiv (i,j,k),\ \mathcal{E}_r \equiv (j,k),\ i<j<k \\ -1, & \mathcal{T}_u \equiv (i,j,k),\ \mathcal{E}_r \equiv (i,k),\ i<j<k \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

and $(\mathbf{B}^{\mathsf{T}}\mathbf{B} + \mathbf{C}\mathbf{C}^{\mathsf{T}})\mathbf{h} = 0$. Here, the gradient flow component is zero if and only if $\mathbf{f}$ is a divergence-free flow. Thus far, we have focused on controlling the gradient flow; specifically, the objective function in Eq. (6) penalizes a solution $\mathbf{f}$ where $\|\mathbf{B}\mathbf{f}\|$ is large.

We can alternatively look at other components of the flow given by the Hodge decomposition. In Eq. (14), the "curl flow" captures all flows that can be composed of flows around triangles in the graph. This component is zero if the sum of edge flows given by $\mathbf{f}$ around every triangle is 0. Combining Eqs. (1) and (15), the curl of a flow on triangle $\mathcal{T}_u = (i,j,k)$ with oriented edges (i,j), (j,k), and (i,k) is $[\mathbf{C}^{\mathsf{T}}\mathbf{f}]_u = f(i,j) + f(j,k) - f(i,k) = f(i,j) + f(j,k) + f(k,i)$.

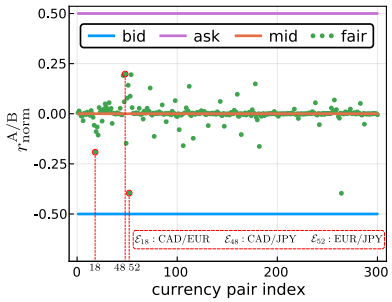

Figure 11: Exchange rates and fair rates in a currency exchange market. Rates are normalized so that bid and ask are 0.5 and -0.5. We find fair trading prices by penalizing curl flow in the exchange.

Finally, the vector $\mathbf{h}$ is called the harmonic flow and measures flows that cannot be constructed from a linear combination of gradient and curl flows. Projecting edge flows onto the space of gradient flows is the HodgeRank method for ranking with pairwise comparisons [20]. In the next section, we use $\|\mathbf{C}^{\mathsf{T}}\mathbf{f}\|$ as part of an objective function to alternatively learn flows that have small curl.

5.2 An application to Arbitrage-Free Pricing

We demonstrate an application of edge-based learning with a different type of flow constraint. In this case study, edge flows are currency exchange rates, where participants buy, sell, exchange, and speculate on foreign currencies. Every pair of currencies has two exchange rates: the bid is the rate at which the market is prepared to buy a specific currency pair. The ask is the rate at which the market is prepared to sell a specific currency pair. There is also a third widely used exchange rate called the "middle rate," which is the average of the bid and ask, is often used as the price to facilitate a trade between currencies. An important principle in an efficient market is the no-arbitrage condition, which states that it is not possible to obtain net gains by a sequence of currency conversions. Although the middle rates are widely accepted as a "fair" rate, they do not always form an arbitrage-free market. For example, in a dataset of exchange rates between the 25 most traded currencies at 2018/10/05 17:00 UTC [8], the middle rates for CAD/EUR, EUR/JPY and JPY/CAD were 0.671200, 130.852 and 0.0113876, respectively. Therefore, a trader successively executing these three transactions would yield 1.000146 CAD from a 1 CAD investment.

Here we show how to construct a "fair" set of exchange rates that is arbitrage-free. We first encode the exchange rates as a flow network. Each currency is a vertex, and for each pair of currencies A and B with exchange rate $r^{\mathrm{A/B}}$, we connect A and B with $\log(r^{\mathrm{A/B}})$ units of edge flow from A to B, which ensures that $f(A,B) = -f(B,A)$. The resulting exchange network is fully connected. Under this setup, the arbitrage-free condition translates into requiring the edge flows in the exchange network to be cycle-free.

We can constrain the edge flows on every triangle to sum to 0 by the curl-free condition $\|\mathbf{C}^{\mathsf{T}}\mathbf{f}\| = 0$. Moreover, in a fully connected network, curl-free flows are cycle-free. Thus, we propose to set fair rates by minimizing the curl over all triangles, subject to the constraint that the fair price lies between the bid and ask prices:

$$\mathbf{f}^* = \arg\min_{\mathbf{f}} \|\mathbf{C}^{\mathsf{T}}\mathbf{f}\|^2 + \lambda^2 \cdot \|\mathbf{f} - \mathbf{f}^{\mathrm{mid}}\|^2 \text{ s.t. } \mathbf{f}^{\mathrm{bid}} \le \mathbf{f} \le \mathbf{f}^{\mathrm{ask}}. \tag{16}$$

The second term in the objective ensures that the minimization problem is not under-determined. In our experiments, $\lambda = 1.0 \cdot 10^{-3}$ and we solve the convex quadratic program with linear constraints

using the Gurobi solver. Unlike the middle rate, which is computed only using the bid/ask rates of one particular currency pair, the solution to the optimization problem above accounts for *all* bid/ask rates in the market to determine the fair exchange rate. Figure 11 shows the fair exchange rates, and the computed rates for CAD/EUR, EUR/JPY and JPY/CAD are 0.671169, 130.845 and 0.0113871, respectively, removing the arbitrage opportunity.

6 RELATED WORK

The Laplacian $\mathbf{L}$ appears in many graph-based SSL algorithms to enforce smooth signals in the vertex space of the graph. Gaussian Random Fields [39] and Laplacian Regulation [4] are two early examples, and there are several extensions [34, 37]. However, these all focus on learning vertex labels and, as we have seen, directly applying ideas from vertex-based SSL to learn edge flows on the line-graph does not perform well. In the context of signal processing on graphs, there exist preliminary edge-space analysis [3, 31, 38], but semi-supervised or active learning are not considered.

In terms of active learning, several graph-based active semi-supervised algorithms have been designed for learning vertex labels, based on error bound minimization [16], submodular optimization [17], or variance minimization [19]. Graph sampling theory under spectral assumptions has also been an effective strategy [12]. Similarly, in Section 2.2, we give exact recovery conditions and derive error bounds for our method assuming the spectral coefficients of the basis vectors representing potential flows are approximately zero, which motivated our use of RRQR for selecting representative edges. There are also clustering heuristics for picking vertices [18]; in contrast, we use clustering to choose informative edges.

7 DISCUSSION

We developed a graph-based semi-supervised learning method for edge flows. Our method is based on imposing interpretable flow constraints to reflect properties of the underlying systems. These constraints may correspond to enforcing divergence-free flows in the case of flow-conserving transportation systems, or non-cyclic flows as in the case of efficient markets. Our method permits spectral analysis for deriving exact recovery condition and bounding reconstruction error, provided that the edge flows are indeed (nearly) divergence free. On a number of synthetic and real-world problems, our method substantially outperforms competing baselines. Furthermore, we explored two active semi-supervised learning algorithms for edge flows. The RRQR strategy works well for synthetic flows, while a recursive partitioning approach works well on real-world datasets. The latter result hints at additional structure in the real-world data that we can exploit for better algorithms.

ACKNOWLEDGEMENTS

This research was supported in part by European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No 702410; NSF Award DMS-1830274; and ARO Award W911NF-19-1-0057.

REFERENCES

[1] Yong-Yeol Ahn, James P. Bagrow, and Sune Lehmann. 2010. Link communities reveal multiscale complexity in networks. *Nature* (2010).
[2] Kristen M. Altenburger and Johan Ugander. 2018. Monophily in social networks introduces similarity among friends-of-friends. *Nature Human Behaviour* (2018).
[3] Sergio Barbarossa and Mikhail Tsitsvero. 2016. An introduction to hypergraph signal processing. In *ICASSP*.
[4] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. 2006. Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. *JMLR* (2006).
[5] T. F. Chan. 1987. Rank revealing QR factorizations. *Linear Algebra Appl.* (1987).
[6] Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. 2003. Cluster Kernels for Semi-Supervised Learning. *NeurIPS*.
[7] Ali Çivril and Malik Magdon-Ismail. 2009. On selecting a maximum volume sub-matrix of a matrix and related problems. *Theoretical Computer Science* (2009).
[8] Oanda Corporation. 2018. Foreign Exchange Data. https://www.oanda.com/.
[9] Anil Damle, Victor Minden, and Lexing Ying. 2016. Robust and efficient multi-way spectral clustering. *arXiv:1609.08251* (2016).
[10] T. S. Evans and R. Lambiotte. 2010. Line graphs of weighted networks for overlapping communities. *The European Physical Journal B* 77, 2 (2010), 265–272.
[11] David Chin-Lung Fong and Michael Saunders. 2011. LSMR: An Iterative Algorithm for Sparse Least-Squares Problems. *SIAM J. Sci. Comp.* (2011).
[12] Akshay Gadde, Aamir Anis, and Antonio Ortega. 2014. Active Semi-supervised Learning Using Sampling Theory for Graph Signals. In *KDD*.
[13] David F. Gleich. 2009. *Models and Algorithms for PageRank Sensitivity.* Ph.D. Dissertation. Stanford University.
[14] David F. Gleich and Michael W. Mahoney. 2015. Using Local Spectral Methods to Robustify Graph-Based Learning Algorithms. In *KDD*.
[15] Chris Godsil and Gordon Royle. 2001. *Algebraic Graph Theory.* Springer.
[16] Quanquan Gu and Jiawei Han. 2012. Towards Active Learning on Graphs: An Error Bound Minimization Approach. In *ICDM*.
[17] Andrew Guillory and Jeff Bilmes. 2011. Active Semi-supervised Learning Using Submodular Functions. In *UAI*.
[18] Andrew Guillory and Jeff A Bilmes. 2009. Label Selection on Graphs. In *NeurIPS*.
[19] Ming Ji and Jiawei Han. 2012. A Variance Minimization Criterion to Active Learning on Graphs. In *AISTATS*.
[20] Xiaoye Jiang, Lek-Heng Lim, Yuan Yao, and Yinyu Ye. 2010. Statistical ranking and combinatorial Hodge theory. *Mathematical Programming* (2010).
[21] Thorsten Joachims. 2003. Transductive learning via spectral graph partitioning. In *ICML*.
[22] Jérôme Kunegis. 2013. KONECT: The Koblenz Network Collection. In *WWW*.
[23] Rasmus Kyng, Anup Rao, Sushant Sachdeva, and Daniel A Spielman. 2015. Algorithms for Lipschitz learning on graphs. In *COLT*.
[24] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *KDD*.
[25] Lek-Heng Lim. 2015. Hodge Laplacians on graphs. In *Proceedings of Symposia in Applied Mathematics, Geometry and Topology in Statistical Inference.*
[26] Antonio Ortega, Pascal Frossard, Jelena Kovačević, José M. F. Moura, and Pierre Vand ergheynst. 2018. Graph Signal Processing. *Proc. IEEE* (2018).
[27] Christopher C. Paige and Michael A. Saunders. 1982. LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares. *ACM TOMS* (1982).
[28] Juan Reca and Juan Martinez. 2006. Genetic algorithms for the design of looped irrigation water distribution networks. *Water Resources Research* (2006).
[29] Michael T. Schaub, Austin R. Benson, Paul Horn, Gabor Lippner, and Ali Jadbabaie. 2018. Random walks on Simplicial Complexes and the normalized Hodge Laplacian. *arXiv:1807.05044* (2018).
[30] Michael T. Schaub, Jörg Lehmann, Sophia N. Yaliraki, and Mauricio Barahona. 2014. Structure of complex networks: Quantifying edge-to-edge relations by failure-induced flow redistribution. *Network Science* (2014).
[31] Michael T. Schaub and Santiago Segarra. 2018. Flow smoothing and denoising: graph signal processing in the edge-space. In *GlobalSIP*.
[32] David I. Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. 2013. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine* (2013).
[33] Horst Simon and Shang-hua Teng. 1998. How Good is Recursive Bisection? *SIAM J. Sci. Comp.* (1998).
[34] Justin Solomon, Raif Rustamov, Leonidas Guibas, and Adrian Butscher. 2014. Wasserstein Propagation for Semi-Supervised Learning. In *ICML*.
[35] Ben Stabler, Hillel Bar-Gera, and Elizabeth Sall. 2016. Transportation Networks for Research. https://github.com/bstabler/TransportationNetworks
[36] Amarnag Subramanya and Partha Pratim Talukdar. 2014. *Graph-Based Semi-Supervised Learning.* Morgan & Claypool Publishers.
[37] Xiao-ming Wu, Zhenguo Li, Anthony M. So, John Wright, and Shih-fu Chang. 2012. Learning with Partially Absorbing Random Walks. In *NeurIPS*.
[38] Daniel Zelazo and Mehran Mesbahi. 2011. Edge Agreement: Graph-Theoretic Performance Bounds and Passivity Analysis. *IEEE Trans. Automat. Control* (2011).
[39] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. 2003. Semi-supervised Learning Using Gaussian Fields and Harmonic Functions. In *ICML*.
[40] Xiaojin Zhu, Andrew B. Goldberg, Ronald Brachman, and Thomas Dietterich. 2009. *Introduction to Semi-Supervised Learning.* Morgan and Claypool Publishers.

A APPENDIX

Here we provide some implementation details of our method to help readers reproduce and further understand the algorithms and experiments in this paper. First, we present the solvers for learning edge flows in divergence-free and curl-free networks. Then, we further discuss the active learning algorithms used for selecting informative edges. All of the algorithms used in this paper are implemented in Julia 1.0.

A.1 Graph-Based SSL for Edge Flows

We created a Julia module *NetworkOP* for processing edge flows in networks. It contains a *FlowNetwork* class which records the vertices, edges and triangles of a graph as three ordered dictionaries.[6] The NetworkOP also provides many convenience functions, such as computing the incident matrix of a FlowNetwork object. Such functions greatly simplify our implementations for learning unlabeled edge flows. In this part, we assume the labeled edges are given. In Appendix A.2 we show algorithms for selecting edges.

```
1  using SparseArrays, NetworkOP;
2  using LinearMaps, IterativeSolvers;
3
4  function ssl_df(A::SparseMatrixCSC{Float64,Int64},
5                  F::SparseMatrixCSC{Float64,Int64},
6                  IdU::Vector{Int64}, lambda=1.0e-1)
7    # A: adjacent matrix of the graph
8    # F: anti-symmetric matrix for ground truth flows
9    # IdU: list of indices for unlabeled edges
10
11   # create a flow network object
12   FN = NetworkOP.FlowNetwork(A);
13   n = length(FN.VV); # n: number of vertices
14   m = length(FN.EE); # m: number of edges
15
16   # assemble edge flows to vector
17   fhat = NetworkOP.mat2vec(FN,F);
18   # the trival solution for Eq.(6)
19   f0 = collect(fhat); f0[IdU] = zeros(length(IdU));
20   # the incidence matrix for network
21   B = NetworkOP.mat_div(FN);
22   # expansion operator \Psi and its transpose
23   expand = ff->collect(sparsevec(IdU,ff,m));
24   select = ff->ff[IdU];
25
26   # the operator in iterative least-squares problem
27   map = LinearMap{Float64};
28   op = map(ff->vcat(B*expand(ff), lambda*ff),
29            pp->select(B'*pp[1:n]) + lambda*pp[n+1:end],
30            n+length(IdU), length(IdU);
31            ismutating=false);
32   # infer edge flows on unlabeled edges with LSQR
33   fU = lsqr(op, vcat(-B*f0, zeros(length(IdU))));
34   # fstar: the inferred edge flow vector
35   fstar = f0 + expand(fU);
36
37   return fstar;
38 end
```

Figure 12: Code snippet for inferring edge flows in a divergence-free network by solving a least-squares problem.

[6] Although we considered unweighted graphs in this work, we choose ordered dictionaries over lists for future exploration of weighted graphs.

```
1  using SparseArrays, NetworkOP;
2  using JuMP, Gurobi;
3
4  function ssl_cf(A::SparseMatrixCSC{Float64,Int64},
5                  bid::Vector{Float64},
6                  mid::Vector{Float64},
7                  ask::Vector{Float64}, lambda=1.0e-3)
8    # A: adjacent matrix of the graph
9    # bid: flow vector f^{bid} representing bid rate
10   # mid: flow vector f^{mid} representing middle rate
11   # ask: flow vector f^{ask} representing ask rate
12
13   # create a flow network object
14   FN = NetworkOP.FlowNetwork(A);
15   n = length(FN.VV); # n: number of vertices
16   m = length(FN.EE); # m: number of edges
17   # map from edge to edge-index
18   e2id = Dict(e=>i for (i,e) in enumerate(keys(FN.EE)));
19
20   model = Model(solver=GurobiSolver(Presolve=0));
21   # variables are the fair exchange rates
22   @variable(model, bid[i] <= f[i=1:m] <= ask[i]);
23   # objective function in Eq.(20)
24   @objective(model, Min,
25     sum((f[e2id[(i,j)]]+f[e2id[(j,k)]]-f[e2id[(i,k)]])^2
26         for i=1:n, j=i+1:n, k=j+1:n) +
27     sum((f[i]-mid[i])^2 for i=1:m))*lambda^2;
28
29   status = solve(model);
30   # fair: flow vector f^{fair} representing fair rate
31   fair = getvalue(f);
32
33   return fair;
34 end
```

Figure 13: Code snippet for fair pricing in a arbitrage-free foreign exchange network by solving a quadratic program with linear constraints (QPLC).

Learning flows in divergence-free networks. Our algorithm for solving Eq. (6) is presented in Fig. 12. This algorithm takes three arguments as input: (1) $\mathbf{A} \in \mathbb{R}^{n\times n}$ is the adjacency matrix of a graph; (2) $\mathbf{F} \in \mathbb{R}^{n\times n}$ is the matrix containing ground truth edge flows, where $\mathrm{F}_{ij} = f(i,j) = -\mathrm{F}_{ji}$; and (3) $\mathbf{IdU} \in \mathbb{N}^{m^{\mathrm{U}}}$ is the edge indices of unlabeled edges. The edge flow matrix $\mathbf{F}$ is transformed into an edge flow vector $\hat{\mathbf{f}}$ with the `mat2vec` function (line 17). Following the formulation in Eq. (7), we define the incidence matrix $\mathbf{B}$ (line 21), the expansion operator $\mathbf{\Phi}$ and its transpose (line 23,24), then we assemble them into a linear map (line 27-31) as the main operator in the least-squares problem. Finally, we solve the least-squares problem with an iterative LSQR solver from the `IterativeSolvers.jl` package.

Learning flows in cycle-free networks. Our algorithm for computing the fair rate in a foreign exchange market is given in Fig. 13. It takes as input the (fully connected) adjacency matrix $\mathbf{A}$ as well as the flow vectors $\mathbf{bid}, \mathbf{mid}, \mathbf{ask} \in \mathbb{R}^m$ representing the corresponding logarithmic exchange rates between currencies. Then we use the `JuMP.jl` package to set up the optimization problem in Eq. (16). The linear constraints and quadratic objective function are specified in line 22 and 24-27 respectively. Finally we solve the QPLC problem with the `Gurobi.jl` package (line 29).

```
using Clustering, NetworkOP;

function al_rb(A::SparseMatrixCSC{Float64,Int64},
               ndim::Int64,
               ratio::Float64)
  # A: adjacency matrix of the graph
  # ndim: number of vectors in spectral clustering
  # ratio: ratio of labeled edges

  # create a flow network object
  FN = NetworkOP.FlowNetwork(A);
  m = length(FN.EE); # m: number of edges

  # compute the embedded vertex coordinates
  X = spectral_embedding(FN, dims);

  IdL = []; # the indices for labeled edges
  clusters = [[1:m]]; # list with current clusters
  # map from edge to edge index
  e2id = Dict(e=>i for (i,e) in enumerate(keys(FN.EE)));

  while (length(IdL) < ratio*m)
    # find the cluster with maximum cardinality
    max_id = argmax([length(cc) for cc in clusters]);
    cluster = clusters[max_id];
    Xc = X[:,cluster];
    km = kmeans(Xc,2,init=:kmpp); # perform k-means
    cid = assignments(km); # get cluster assignment
    # split the cluster into two, update cluster list
    clusters[max_id] = cluster[cid .== 1];
    push!(clusters, cluster[cid .== 2]);
    # add the edges connecting two clusters to IdL
    for i in clusters[max_id]
      for j in clusters[end]
        e = i < j ? (i,j) : (j,i);
        if (e in keys(e2id))
          push!(IdL, e2id[e]);
        end
      end
    end
  end
end
```

Figure 15: Code snippet of the recursive bisection active learning algorithm for selecting informative edges.

```
using LinearAlgebra, NetworkOP;

function al_rrqr(A::SparseMatrixCSC{Float64,Int64},
                 ratio::Float64)
  # A: adjacency matrix of the graph
  # ratio: ratio of labeled edges

  # create a flow network object
  FN = NetworkOP.FlowNetwork(A);
  n = length(FN.VV); # n: number of vertices
  m = length(FN.EE); # m: number of edges
  # the incidence matrix for network
  B = NetworkOP.mat_div(FN);

  # basis for cyclic edge flows
  VC = nullspace(B);

  # permutation order from RRQR
  od = qr(VC',Val(true)).p;
  # the indices for labeled edges
  IdL = od[1:Int64(ceil(m*ratio))];

  return IdL;
end
```

Figure 14: Code snippet of the RRQR active learning algorithm for selecting informative edges.

A.2 Active Learning Strategies

Now we look at the implementation of the two active learning algorithms. Given the adjacency matrix $\mathbf{A}$ as input, those active learning algorithms output the selected indices $\mathbf{IdL} \in \mathbb{N}^{m^{\mathrm{L}}}$ of edges to be labeled.

RRQR algorithm. We present our RRQR active learning algorithm in Fig. 14. First, we compute an orthonormal basis $\mathbf{V}_C$ for the cycle-space $C = \ker(\mathbf{B})$ representing cyclic edge flows (line 16). Then, we perform pivoted QR decomposition on the rows of $\mathbf{V}_C$ (line 19), and the edge indices for labeled edges are given by the first m^{L} permutations (line 21).

Recursive bisection algorithm. We present our recursive bisection active learning algorithm in Fig. 15. This algorithm first uses a spectral embedding to compute the vertex coordinates (line 15). Then it repeatedly chooses the largest cluster in the graph (line 24-26), uses the k-means (Lloyd's) algorithm to divide the chosen cluster into two (line 27-31), and adds the edges that connect the two resulting clusters into the labeled edges indices (line 33-40). Note that the k-mean algorithm sometimes fails due to bad initial cluster centers or vertices with same embedded coordinates; however, we omit the code for dealing with those corner cases here due to limited space.

The complete implementation of all of our algorithms can be found at https://github.com/000Justin000/ssl_edge.git.