Massively Distributed Graph Distances

Armin Moharrer, Jasmin Gao, Shikun Wang, José Bento, Stratis Ioannidis

***Abstract*—Graph distance (or similarity) scores are used in several graph mining tasks, including anomaly detection, nearest neighbor and similarity search, pattern recognition, transfer learning, and clustering. Graph distances that are metrics and, in particular, satisfy the triangle inequality, have theoretical and empirical advantages. Well-known graph distances that are metrics include the chemical or the Chartrand-Kubiki-Shultz (CKS) distances. Unfortunately, both are computationally intractable. Recent efforts propose using *convex relaxations* of the chemical and CKS distances. Though distance computation becomes a convex optimization problem under these relaxations, the number of variables is quadratic in the graph size; this makes traditional optimization algorithms prohibitive even for small graphs. We propose a distributed method for massively parallelizing this problem using the Alternating Directions Method of Multipliers (ADMM). Our solution uses a novel, distributed bisection algorithm for computing a p-norm proximal operator as a building block. We demonstrate its scalability by conducting experiments over multiple parallel environments.**

***Index Terms*—Graph Matching, Optimization, ADMM, Distributed Algorithms**

I. Introduction

Graphs are ubiquitous combinatorial objects, representing real-world phenomena from social and information networks to technological, biological, chemical, and brain networks. Graph distance (or similarity) scores find applications in varied fields, such as image processing [1], chemistry [2], [3], and social network analysis [4], [5]. Graph distances are used in several graph mining tasks, including anomaly detection [6], [7], nearest neighbor and similarity search [6], [8]–[11], pattern recognition [8], [11], transfer learning [12], and clustering [13], to name a few. Distance scores that are *metrics*–and satisfy the triangle inequality property–exhibit significant computational advantages. From a theoretical standpoint, operations such as nearest-neighbor search [14]–[16], outlier detection [17], clustering [18]–[20], and diameter computation [21] can be computed or approximated efficiently over objects embedded in a metric space. Beyond theoretical guarantees, in practice, metrics often significantly improve performance and/or quality compared to non-metrics in a variety of tasks. For example, graph clustering algorithms are better at detecting clusters over metric spaces (see, e.g., [13]). Metric graph distances are therefore highly desirable.

Armin Moharrer is with the Department of Electrical and Computer Engineering of Northeastern University, Boston, MA 02115, USA

Jasmin Gao is with the Department of Operations Research and Financial Engineering of Princeton University, Princeton, NJ 08544, USA

Shikun Wang was with the Department of Computer Science, Boston College, Chestnut Hill, MA 02467, USA

José Bento is with the Department of Computer Science, Boston College, Chestnut Hill, MA 02467, USA

Stratis Ioannidis is with the Department of Electrical and Computer Engineering of Northeastern University, Boston, MA 02115, USA

Well-known graph distances that are metrics include the so-called *chemical* [3] and the *Chartrand-Kubiki-Shultz (CKS)* [22] distances. The chemical distance between two graphs G_A and G_B with adjacency matrices $\boldsymbol{A}, \boldsymbol{B} \in \{0,1\}^{n\times n}$, respectively, is defined as:

$$\min_{\boldsymbol{P}\in\mathcal{P}^n} \|\boldsymbol{AP}-\boldsymbol{PB}\|_2, \tag{1}$$

where $\mathcal{P}^n$ is the set of permutation matrices, and $\|\cdot\|_2$ is the $p=2$ (a.k.a. Frobenius) norm. Intuitively, the solution to Prob. (1) counts the number of edges present in one graph but not the other, under a node correspondence (mapping) captured by permutation matrix $\boldsymbol{P}$. The CKS distance has the same formulation, replacing the adjacency matrices with matrices comprising shortest path distances. Unfortunately, both distances are computationally intractable [23].

To address this, Bento and Ioannidis [13] recently proposed a *convex relaxation* of these distances, which attains tractability while also naturally incorporating node features. In a nutshell, the authors define the distance between two n-node graphs G_A and G_B as the optimal value of the problem:

$$\min_{\boldsymbol{P}\in\mathcal{W}^n} \|\boldsymbol{AP}-\boldsymbol{PB}\|_p + \lambda\cdot \mathsf{tr}\left(\boldsymbol{P}^\top \boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}\right), \tag{2}$$

where $\mathcal{W}^n$ is the set of doubly stochastic matrices, $\|\cdot\|_p$ is the entry-wise p-norm, $\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}\in\mathbb{R}^{n\times n}$ denotes dissimilarities between the nodes of the two graphs, and $\lambda\geq 0$ is a hyper-parameter.

The relaxation of the chemical distance defined by Prob. (2) has several advantages. First, it is tractable, as it involves solving a convex optimization problem. Second, Bento and Ioannidis show that the distance resulting from solving Prob. (2) is a metric and, in particular, satisfies the triangle inequality. This yields the aforementioned benefits of metrics in downstream tasks such as, e.g., graph clustering or nearest-neighbor search. Third, it incorporates node features via the linear trace term. This has computational advantages (which we discuss in Section III-G), but is also important in practice: nodes in real-life graphs often contain such information (e.g., demographic information of users in a social network, atom properties in a molecule, etc.). Finally, Prob. (3) encompasses multiple p-norms and possible distance matrices $D_{A,B}$, for which both the metric property and convexity are maintained [13]. The ability to span different norms is also very important in practice, as the right value of p can be data dependent (see Tables III and IV in Sec. VI-B).

Even though Prob. (2) is a convex optimization problem, the number of variables is quadratic in the graph size n; this makes traditional optimization methods for solving (2) prohibitive even for small n. Nevertheless, for $p=1$, the problem can be solved in a distributed fashion via the Alternating Directions Method of Multipliers (ADMM) [24], since its objective decomposes into a sum of simpler objective functions.

Unfortunately, it is not clear how to efficiently distribute the solution for $p > 1$; this is precisely because, for $p > 1$, the objective of (2) cannot be written as a sum of distinct terms. Our present work directly addresses this challenge: we propose a distributed algorithm solving (2) for arbitrary $p \geq 1$. Our solution combines ADMM with a distributed proximal operator for arbitrary p-norms, which is both novel and of independent interest. Finally, we demonstrate the applicability of our algorithm via massively distributed implementations over OpenMP and Apache Spark, which we make publicly available.[1] In summary, we make the following contributions:

- We propose an ADMM-based distributed algorithm for solving (3) *for all* $p \geq 1$. Our solution for the case $p > 1$ uses a nested-ADMM (Alg. 1 and 2) in combination with a distributed bisection algorithm (Alg. 3) as building blocks.
- We describe the algorithm's parallel complexity in terms of the sparsity of graphs G_A, G_B, and additional constraints we introduce in the problem. In particular, we bound message exchanges in terms of these sparsity parameters.
- We implement our algorithm in OpenMP [25] and Spark [26]. Our publicly available implementation scales to hundreds of CPUs. Over a 448 CPU cluster, we attain speedups as much as 153×.

The remainder of the paper is organized as follows. We review related work in Sec. II. We review basic definitions, convex relaxation (2), and ADMM in Sec. III. We present our main algorithm in Sec. IV, its computational complexity in Sec. V, and our experiments in Sec. VI. We conclude in Sec. VII.

II. Related Work

Graph Distances. A distance between two graphs can be defined naturally when they are labeled, i.e., the correspondence between their nodes is known (see, e.g., [5], [27], [28]). Two classic examples are the edit distance [29], [30] and the maximum common subgraph distance [31], [32]. Some recent works focus on distances for labeled graphs that are easy to compute (e.g in linear or quadratic time) [5], [27], [28] without maintaining the properties of a metric. We study the (harder) unlabeled setting, in which the node correspondence between graphs is unknown. Examples of distances in this setting include the chemical [3] and the Chartrand-Kubiki-Shultz (CKS) [22] distances, while the edit and the maximum common subgraph distances can also be extended to the unlabeled setting. All four [31]–[34] are metrics and hard to compute, while existing heuristics (e.g., [35], [36]) do not satisfy the triangle inequality property. A simple approach to induce a metric over unlabeled graphs is to embed them in a common metric space and then measure the distance of these embeddings. Riesen et al. [37], [38] embed graphs into real vectors by computing their edit distances to a set of *prototype* graphs. The same embedding is also used to compute a median of graphs [39]. Other works [40]–[42] map graphs to spaces determined by their spectral decomposition. Such approaches are not as discriminative as the metrics considered here [13], because embeddings only summarize the graph structure.

[1] https://github.com/neu-spiral/GraphMatching

Metrics. Metrics naturally arise in data mining tasks, including clustering [43], [44], nearest neighbour search [14]–[16], and outlier detection [17]. Some of these tasks become tractable, or admit formal guarantees, precisely when performed over a metric space. For example, finding the nearest neighbor [14]–[16] or the diameter of a dataset [21] become polylogarithimic under metric assumptions; similarly, approximation algorithms for clustering (which is NP-hard) rely on metric assumptions, whose absence leads to a deterioration of known bounds [18]. Our focus on metrics is motivated by these considerations.

Graph Matching. Graph matching has a long history in machine learning and pattern recognition [1], [45], [46]. Given two graphs, the graph matching problem amounts to finding a node-to-node correspondence (or mapping) that preserves edge relationships across two graphs. This relates to distance computation, as the optimal mapping can be cast as the solution of a minimum distance computation problem. For example, graph matching is commonly formulated as a quadratic assignment problem [1], [47]–[49], which is generally NP-hard [45]. There are many works solving this problem approximately (see [1] for a thorough review). NetAlignMR [47] proposes and solves an integer linear programming relaxation. For the same linear relaxation, NetAlignBP [50] uses a more efficient belief propagation (BP) method. Natalie [49] proposes another integer linear programming relaxation. A different approach via a graduated assignment was proposed by Gold and Rangarajan [45]. IsoRank [51] finds a score matrix via a spectral algorithm. Closest to us, Lyzinski et al. [52] propose both a convex and non-convex relaxation over the set of doubly stochastic matrices: their convex relaxation is Eq. (2) with $\lambda = 0$ and $p = 2$, while the objective in the non-convex relaxation is the quadratic function $\mathsf{tr}((\boldsymbol{AP})^\top \boldsymbol{PB})$. Schellewald and Schnörr [53] propose a semi-definite programming relaxation. Though highly efficient, these approaches generally do not yield distances that are metrics (see [13]).

Proximal Operators. We use a bisection algorithm due to Liu and Ye [54] to compute the proximal operator of p-norms. The original presentation of the algorithm was serial; we show (and exploit) in our work the fact that the algorithm can be implemented in parallel via map-reduce operations. Beyond this, we also provide a convergence guarantee (Thm. IV.3), which was absent from their work. Sra [55] extends Liu and Ye's approach, proposing a bisection method for finding the proximal operators of mixed $\ell_{1,p}$ norms. The same author also provides a proximal operator algorithm for mixed $\ell_{p,q}$ norms in a follow-up work [56]. Proximal operators can be seen as generalizations of projection operators [57]; in the case of norms, they are coupled to projections via Moreau's decomposition [58]. Exploiting the latter, some works solve the problem in the dual domain via projections on the unit ball of the dual norm [57] or via gradient methods [59]. These methods are not readily parallelizable.

ADMM. The Alternating Direction Method of Multipliers (ADMM) [60] is a convex optimization algorithm. Consensus ADMM [24], which we use here, is a classic approach to distribute optimization problems; its applications are numerous [48], [61]–[66]. For strongly convex problems, its optimally-tuned convergence rate is as fast as that of the fastest first-

$\boldsymbol{x}, \boldsymbol{y}$	Vectors	$\boldsymbol{A}, \boldsymbol{B}$	Adjacency matrices
$\mathcal{Q}$	Mapping constraint set	a_{ij}, b_{ij}	Matrix elements in $\boldsymbol{A}$ and $\boldsymbol{B}$
$[n]$	Set $\{1, \ldots, n\}$	$\mathcal{R}^{(i)}, \mathcal{C}^{(j)}$	Simplex sets
$\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}$	Matrix of node distances d_{ij}	F_i	local objectives
$\|\cdot\|_0$	support size, i.e., $\|\operatorname{supp}(\cdot)\|$	$\|\cdot\|_p$	p-norm, entrywise, for $p \geq 1$
$G(\mathcal{V}, E)$	Graph, vertex set, and edge set	$\boldsymbol{z}, \boldsymbol{x}_i$	Consensus/local variable
$A_{\mathcal{S}}$	Matrix Projection on coordinate set $\mathcal{S}$	$\boldsymbol{x}_{\mathcal{S}}$	Vector Projection on coordinates set $\mathcal{S}$
$\boldsymbol{P}$	Doubly stochastic matrix	$\mathcal{I}$	Set of non-empty subsets $\mathcal{S}_{ij}$
$\boldsymbol{P}_{ij}, \boldsymbol{y}_{ij}$	Local/dual variable pair	q_{ij}, ξ_{ij}	Local/dual variable pair
$\boldsymbol{r}_i, \boldsymbol{\psi}_i$	Local/dual variable pair	$\boldsymbol{c}_j, \boldsymbol{\phi}_j$	Local/dual variable pair
$\mathcal{S}_{(\cdot)}$	Subset of coordinates on which the objectives depend	$\mathcal{P}^n, \mathcal{W}^n$	Sets of permutation and doubly stochastic matrices
x_i, y_i	Coordinates of vectors $\boldsymbol{x}, \boldsymbol{y}$	ER(n, q)	Erdős Rényi graph with n nodes and edge probability q

Table I: Summary of Notation

order method [67]. Though we focus on the simplest setting, extensions include asynchronous [64] and stochastic [66] versions, adaptive ways of updating parameter ρ [65], and faster variants that solve subproblems inexactly [68]. Applying such optimizations to our work is an interesting open question.

III. Technical Preliminary

A. Basic Definitions and Notations

Graphs. We represent a graph $G(\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V} = [n] \equiv \{1, \ldots, n\}$ and edge set $\mathcal{E} \subseteq [n] \times [n]$ by its *adjacency matrix*, i.e., $\boldsymbol{A} = [a_{ij}]_{i,j\in[n]} \in \{0,1\}^{n\times n}$ s.t. $a_{ij} = 1$ iff $(i,j) \in \mathcal{E}$. A graph is *bipartite* if its node set can be partitioned into two disjoint sets $\mathcal{V}_L$ and $\mathcal{V}_R$ such that no edges exist within the same partition, i.e., $\mathcal{E} \subseteq \mathcal{V}_L \times \mathcal{V}_R$. We denote bipartite graphs by $G(\mathcal{V}_L, \mathcal{V}_R, \mathcal{E})$.

Matrix Norms and Projections. Given a matrix $\boldsymbol{A} = [a_{ij}]_{i,j\in[n]} \in \mathbb{R}^{n\times n}$ and $p \in \mathbb{R}_+$, where $p \geq 1$, its *entry-wise p-norm* is $\|\boldsymbol{A}\|_p = (\sum_{i=1}^n \sum_{j=1}^n |a_{ij}|^p)^{1/p}$. We use $\|\boldsymbol{A}\|_0$ to indicate the number of non-zero elements (a.k.a. the size of the support) of $\boldsymbol{A}$, i.e., $\|\boldsymbol{A}\|_0 \equiv |\{(i,j) : a_{ij} \neq 0\}| = |\operatorname{supp}(\boldsymbol{A})|$. Given a vector $\boldsymbol{x} \in \mathbb{R}^n$ and an ordered set $\mathcal{S} \subseteq [n]$, we denote the projection of $\boldsymbol{x}$ on a subset $\mathcal{S}$ of its coordinates by $\boldsymbol{x}_{\mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|}$. Similarly, given a matrix $\boldsymbol{A} = [a_{ij}]_{i,j\in[n]} \in \mathbb{R}^{n\times n}$ and a set $\mathcal{S} \subseteq [n] \times [n]$, we define $\boldsymbol{A}_{\mathcal{S}} \in \mathbb{R}^{|\mathcal{S}|}$ to be the projection of $\boldsymbol{A}$ on its coordinates in $\mathcal{S}$; that is, $\boldsymbol{A}_{\mathcal{S}}$ is the $|\mathcal{S}|$-dimensional vector comprising the elements $a_{ij}, (i,j) \in \mathcal{S}$. We denote by $\mathcal{P}^n = \{\boldsymbol{P} \in \{0,1\}^{n\times n} : \boldsymbol{P}\mathbf{1} = \mathbf{1}, \boldsymbol{P}^\top\mathbf{1} = \mathbf{1}\}$ the set of *permutation* matrices and by $\mathcal{W}^n = \{\boldsymbol{P} \in [0,1]^{n\times n} : \boldsymbol{P}\mathbf{1} = \mathbf{1}, \boldsymbol{P}^\top\mathbf{1} = \mathbf{1}\}$ the set of *doubly-stochastic* matrices (i.e., the *Birkhoff polytope*).

B. Chemical Distance

Let $\boldsymbol{A}, \boldsymbol{B} \in \{0,1\}^{n\times n}$ be the adjacency matrices of two graphs $G_A(\mathcal{V}, \mathcal{E}_A)$ and $G_B(\mathcal{V}, \mathcal{E}_B)$. Graphs G_A and G_B are *isomorphic* iff there exists $\boldsymbol{P} \in \mathcal{P}^n$ s.t. $\boldsymbol{P}^\top\boldsymbol{A}\boldsymbol{P} = \boldsymbol{B}$ or, equivalently, $\boldsymbol{A}\boldsymbol{P} = \boldsymbol{P}\boldsymbol{B}$. The *chemical distance* extends the latter relationship to capture graph distances. The chemical distance between G_A and G_B is defined via Prob. (1). Intuitively, Prob. (1) counts the number of edges present in one graph but not the other, under a node correspondence (mapping) captured by permutation matrix $\boldsymbol{P}$. Unfortunately, there is no poly-time algorithm for solving (1) [23].

C. Convex Relaxation

Bento and Ioannidis [13] introduce a tractable family of distances that generalizes the chemical distance. The family can be expressed via convex optimization problems, that can be solved via, e.g., barrier methods; nevertheless, the number of variables is quadratic in the graph size n, which motivates our exploration of a distributed implementation.

Formally, given the n-node graphs $G_A(\mathcal{V}, \mathcal{E}_A)$ and $G_B(\mathcal{V}, \mathcal{E}_B)$, where $\mathcal{V} = [n]$, Bento and Ioannidis suggest computing the distance between graphs as the minimum of the following problem:

$$\text{Minimize} \quad \|\boldsymbol{A}\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\|_p + \lambda \cdot \operatorname{tr}\left(\boldsymbol{P}^\top \boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}\right), \tag{3a}$$

$$\text{subj. to:} \quad \boldsymbol{P} \in \mathcal{W}^n, \quad p_{ij} = 0 \text{ for all } (i,j) \notin \mathcal{Q}, \tag{3b}$$

where $\mathcal{Q} \subseteq [n] \times [n]$ is a set of pairs constraining the support of $\boldsymbol{P}$, $\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}} = [d_{ij}]_{(i,j)\in[n]\times[n]}$ is a matrix, s.t., d_{ij} measures the dissimilarity between some features of the nodes $i \in \mathcal{V}$ and $j \in \mathcal{V}$, and $\lambda \geq 0$ is a tuning parameter.

Intuitively, Prob. (3) finds a stochastic mapping between nodes that minimizes edge discrepancy, while also taking into account node feature distances as well as hard constraints. More specifically, the doubly-stochastic matrix $\boldsymbol{P}$ can be interpreted as a stochastic mapping, where $p_{ij} \in [0,1]$ shows the probability that node i in G_A is mapped to node j in G_B. Prob. (3) thus seeks a stochastic mapping $\boldsymbol{P}$ that (a) minimizes the edge discrepancy between adjacency matrices, captured by term $\|\boldsymbol{A}\boldsymbol{P} - \boldsymbol{P}\boldsymbol{B}\|$, (b) penalizes mappings between nodes i in G_A and node j in G_B that have distinct features, captured by linear term $\operatorname{tr}\left(P^\top \boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}\right)$, and (c) further restricts mappings to have support in $\mathcal{Q}$.

We discuss examples illustrating different feature distance matrices $\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}$ and constraints $\mathcal{Q}$ below, in Sec. III-D. In short, node features can be incorporated in a *soft* manner, through the linear term in objective (3a), or as *hard* constraints in $\mathcal{Q}$ (requiring, e.g., nodes with different categorical features to never be mapped to each other).

Computing distances via Prob. (3) has several important advantages. First, under mild conditions on $\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}$ and $\mathcal{Q}$, the distance computed by Prob. (3) is a metric; this is proved by Bento and Ioannidis [13]. Second, for arbitrary p-norms, (3) is a convex optimization problem. As a result, a solution can be computed using standard methods [69]. Third, the linear term $\operatorname{tr}\left(\boldsymbol{P}^\top \boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}\right)$ and the constraints $\mathcal{Q}$ allow us to capture auxiliary information that often exists in practice, such as node features or labels. Beyond this expressive power, both have significant computational advantages, as we show in Sec. VI.

D. Constraints and Node Features

In practice, graph nodes are often endowed with features or attributes that we can leverage in graph distance computations. Here, we explain how node features can be incorporated in Prob. (3) via either the linear term or the constraint set $\mathcal{Q}$.

Node Features in $\mathbb{R}^d$. Node attributes can be represented as, e.g, k-dimensional feature vectors in $\mathbb{R}^d$. Having access to such features, we can compute the elements of the dissimilarity matrix $\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}} = [d_{ij}]_{i,j\in[n]}$ by taking, e.g., the ℓ_2 (or other vector) norm of the difference between these

vectors: that is $d_{ij} = \|x_i - x_j\|_2$, where $x_i, x_j \in \mathbb{R}^k$ are the k-dimensional feature vectors for i in G_A and j in G_B.

Node features can be *exogenous*, e.g., the demographic attributes of a user in a social network, the atomic number of an atom in a molecule, etc. Alternatively, features can be *endogenous*, i.e., computed directly from the adjacency matrix: these include, e.g., a node's degree, it's centrality, its pagerank [70], node2vec representation [71], [72], or some other vector computed via graph signal processing [71], [73]. Exogenous features are often available in practical settings, while endogenous features can have computational advantages: we observe this in Sec. VI, where adding a linear term often accelerates convergence but also produces higher quality solutions.

Categorical Features (Colors/Labels). Rather than including categorical node features as *soft* constraints, via the trace penalty, such features can also be used to produce *hard* constraints, captured by $\mathcal{Q}$. Suppose that we are given a categorical node feature, referred to as a node's *color*. We can construct the constraint set $\mathcal{Q}$ by including only pairs (i, j) s.t. the nodes i and j across the two graphs have the same color.

Colors can again be either exogenous or endogenous/structural. As examples of exogenous colors, if the graph represents an organic molecule, the color can be the node's atomic number; then, constraint $\mathcal{Q}$ requires that identical atoms are mapped to each other across the two graphs. If the graph represents a social network, colors can correspond to different demographic attributes, (e.g., gender, age group, etc.) Structural/endogenous colors, on the other hand, can be categorical variables capturing the local neighborhood structure around a node. These can be, e.g., node degrees, the number of triangles that pass through a node, or some other discrete statistic generated from a node's k-hop neighborhood. One such statistic is the output of the so-called Weisfeiler-Lehman (WL) algorithm [74], executed after k iterations.

Using categorical variables of the above nature to construct constraint set $\mathcal{Q}$ has several advantages. First, Bento and Ioannidis show that Prob. (3) remains a metric, even when incorporating such constraints. Most importantly, introducing constraints can significantly decrease the number of optimization variables and, hence, the computational complexity of Prob. (3). As we discuss in Section V, the sparsity of $\mathcal{Q}$ also dictates the communication complexity our parallel algorithm for solving Prob. (3).

E. Consensus ADMM

Consensus ADMM is an iterative optimization algorithm well-suited for solving convex optimization problems in a distributed fashion. Problems amenable to a distributed solution via consensus ADMM have a specific form: their objective can be written as a sum of functions, each depending only on a few variables. Formally, consider the optimization problem:

$$\text{Minimize} \quad F(\boldsymbol{x}) = \textstyle\sum_{i=1}^{N} F_i(\boldsymbol{x}_{\mathcal{S}_i}), \tag{4}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ and each term $F_i : \mathbb{R}^{|\mathcal{S}_i|} \to \mathbb{R}$ is convex and depends on a subset $\mathcal{S}_i \subseteq [n]$ of the coordinates of $\boldsymbol{x}$. Prob. (4) can be re-written with N *local* variables $\boldsymbol{x}_i \in \mathbb{R}^{|\mathcal{S}_i|}, i \in [N]$ and a single *consensus* variable $\boldsymbol{z} \in \mathbb{R}^n$ as:

$$\text{Minimize} \quad \textstyle\sum_{i=1}^{N} F_i(\boldsymbol{x}_i) \tag{5a}$$

$$\text{subj. to:} \quad \boldsymbol{x}_i = \boldsymbol{z}_{\mathcal{S}_i} \quad i = 1, \ldots, N, \tag{5b}$$

where $\boldsymbol{z}_{\mathcal{S}_i}$ is the projection of $\boldsymbol{z}$ on the subset $\mathcal{S}_i$. The k-th iteration of consensus ADMM for (5) is as follows:

$$\boldsymbol{x}_i^{k+1} = \arg\min_{\boldsymbol{x}_i} F_i(\boldsymbol{x}_i) + \tfrac{\rho}{2}\|\boldsymbol{x}_i - \boldsymbol{z}_{\mathcal{S}_i}^k + \boldsymbol{y}_i^k\|_2^2, \ \forall i \in [N], \tag{6a}$$

$$\boldsymbol{z}_j^{k+1} = \frac{\sum_{i:j\in\mathcal{S}_i}\left((\boldsymbol{x}_i^{k+1})_{\ell_i(j)} + (\boldsymbol{y}_i^k)_{\ell_i(j)}\right)}{|\{i \in [N] : j \in \mathcal{S}_i\}|}, \quad \forall j \in [n], \tag{6b}$$

$$\boldsymbol{y}_i^{k+1} = \boldsymbol{y}_i^k + (\boldsymbol{x}_i^{k+1} - \boldsymbol{z}_{\mathcal{S}_i}^{k+1}), \quad \forall i \in [N], \tag{6c}$$

where $\rho > 0$ is a tuning parameter and $\boldsymbol{y}_i \in \mathbb{R}^{|\mathcal{S}_i|}, i \in [n]$, are dual variables corresponding to the constraints (5b). and $\ell_i : \mathcal{S}_i \to \{1, \ldots, |\mathcal{S}_i|\}$ maps coordinates in $\mathcal{S}_i$ to their "local" representations in $\boldsymbol{x}_i$.

Incorporating Constraints. We can include constraints in ADMM by adding them to the objective (5a) via their characteristic functions: a constraint $\boldsymbol{x} \in \mathcal{D}$, where $\mathcal{D}$ is a convex set, is added to (5a) as a term $\chi_{\mathcal{D}}(\boldsymbol{x})$, where $\chi_{\mathcal{D}}$ is the characteristic function of $\mathcal{D}$ (0 if $\boldsymbol{x} \in \mathcal{D}$, $+\infty$ o.w.). Then, the corresponding step (6a) becomes a Euclidean projection onto convex set $\mathcal{D}$.

A Parallel Implementation. All the above steps in (6) can be parallelized. To see this, suppose that we have $N + n$ processors, as illustrated in Fig. 1. The N processors in $\mathcal{V}_{\texttt{obj}} \equiv [N]$ are responsible for solving problems (6a) and performing the dual variable adaptation (6c), in parallel. To do so, they store functions F_i as well as "local" primal and dual variables $\boldsymbol{x}_i$,$\boldsymbol{y}_i$, $i \in [N]$. The remaining n processors $\mathcal{V}_{\texttt{var}} = [n]$ store the coefficients z_j, $j \in [n]$, of the consensus variable $\boldsymbol{z}$ and perform the averaging (6b). In each iteration, the processors in $\mathcal{V}_{\texttt{var}}$ send the consensus variables to the corresponding processors in $\mathcal{V}_{\texttt{obj}}$. Subsequently, the latter perform adaptations (6c) and (6a), and then send their new local variables to the processors in $\mathcal{V}_{\texttt{var}}$ for averaging.

The communication complexity of each step (6), as well as the dependencies between steps, are determined by the bipartite graph $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$ shown in Fig. 1: each processor $i \in [N]$ on the left needs to receive the $|\mathcal{S}_i|$ consensus variables z_j, $j \in \mathcal{S}_i$ to perform (6a) and (6c), while processors $j \in [n]$ on the right need to collect $|\{i \in [N] : j \in \mathcal{S}_i\}|$ local variables $(x_i)_{\ell_i(j)}$. As a result, the number of messages exchanged is proportional to the number of edges in $\mathcal{G}$, namely, $\sum_{i\in[N]} |\mathcal{S}_i|$.

F. Map-Reduce

Given an N-dimensional vector $\boldsymbol{x} \in \mathcal{X}^N$, for some domain $\mathcal{X}$, a `map` operation applies a function to every element $\boldsymbol{x}$. That is, given $f : \mathcal{X} \to \mathcal{X}'$, the operation $\boldsymbol{x}' = \boldsymbol{x}.\texttt{map}(f)$ creates a vector $\boldsymbol{x}'$ in which every element x_i, $i \in [N]$, is replaced with $f(x_i)$. A `reduce` operation aggregation over $\boldsymbol{x}$; a canonical example is, e.g., computing the sum of $\boldsymbol{x}$'s coordinates. Formally, let $\oplus$ be a commutative, associative, binary operator $\oplus : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$. Then, $\boldsymbol{x}.\texttt{reduce}(\oplus)$ iteratively applies the binary operator $\oplus$ on $\boldsymbol{x}$, returning $\bigoplus_{i\in[N]} x_i = x_1 \oplus \ldots \oplus x_N$.

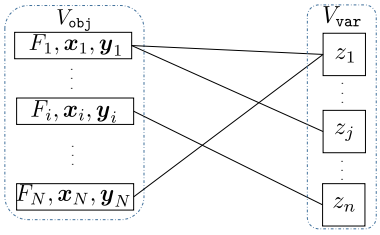


Figure 1: A bipartite graph $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$ showing the dependencies of the functions F_i on the coordinates of the global consensus variable z, as well as communication pattern during parallelism. Each node in the graph corresponds a processor. Processors in $\mathcal{V}_{\texttt{obj}}$ store F_i, $\boldsymbol{x}_i$, $\boldsymbol{y}_i$, $i \in [N]$, and perform steps (6a) and (6c), while processors in $\mathcal{V}_{\texttt{var}}$ store z_i, $i \in [n]$, and perform (6b).

Both `map` and `reduce` operations are "embarrassingly parallel". Presuming that $\boldsymbol{x}$ is distributed over N processors, a `map` can be executed without any communication among processors, other than the one required to broadcast the code that executes f. This broadcast can be done in $\log_2 N$ rounds via the transmission of $N-1$ messages, when the N processors are connected in a hypercube network. Again, under a hypercube network, `reduce` operations have the same parallel complexity (can be computed in $\log_2 N$ rounds via $N-1$ messages) [75].

G. Importance of p-norms and Linear Term

Given the size of both the input graphs, our goal is to produce a distributed algorithm for solving Prob. (3). As we discuss in Sec. IV, the main challenge arises from presence of the p-norm in combination with the linear term. One possible solution is to limit objective (3a) to the case $p = 1$. This leads to an objective paralellizable via consensus ADMM. This, on the other hand, is unsatisfactory, as the ideal norm may depend on the underlying graphs; we elaborate on this in Sec. VI, where we see how inherent noise can effect our norm choice (see p-norms in Sec. VI-B). Another solution is to modify objective (3a), replacing $\|\cdot\|_p$ with $\|\cdot\|_p^p$. This has two significant drawbacks. First, under this modification, the distance is no longer a metric; in particular, it fails to satisfy the triangle inequality, which is a significant disadvantage for downstream applications, as mentioned earlier. Second, from an optimization standpoint, it is important to keep the two terms in objective (3a) balanced; this is harder in this case as $\|\cdot\|_p^p$ is not absolutely homogeneous (in contrast to both the trace and norms).

A final alternative is to remove the linear term altogether. In this case, minimizing $\|\cdot\|_p$ is equivalent to minimizing $\|\cdot\|_p^p$. This annuls any benefits of incorporating features, both in terms of modeling, e.g., exogenous node attributes, but also in terms of efficiency: as our experiments demonstrate (see, e.g., Fig. 2 in Sec. VI-B), including the linear term can significantly accelerate convergence.

IV. Main Results

We now turn our attention to solving (3) via ADMM. We incorporate constraints (3b) in (3a), yielding objective:

$$\|\boldsymbol{A}\boldsymbol{P}-\boldsymbol{P}\boldsymbol{B}\|_p+\lambda\,\mathsf{tr}\left(\boldsymbol{P}^\top\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}\right)+\chi_{\mathcal{R}}(\boldsymbol{P})+\chi_{\mathcal{C}}(\boldsymbol{P}), \quad (7)$$

where the sets

$$\mathcal{R} = \{\boldsymbol{P} \in [0,1]^{n\times n} : \boldsymbol{P}\mathbf{1} = \mathbf{1}, p_{ij} = 0\ \forall (i,j) \notin \mathcal{Q}\},\ \text{and}$$
$$\mathcal{C} = \{\boldsymbol{P} \in [0,1]^{n\times n} : \boldsymbol{P}^\top\mathbf{1} = \mathbf{1}, p_{ij} = 0\ \forall (i,j) \notin \mathcal{Q}\},$$

correspond to the (doubly stochastic) constraints on the rows and columns, respectively. With the exception of the first term, all remaining terms in (7) can be written as sums. Indeed, the following lemma holds:

Lemma IV.1. *There exists a set $\mathcal{I} \subseteq [n]\times[n]$ as well as sets $\mathcal{S}_{ij} \subseteq [n]\times[n]$, $\mathcal{S}_i \subseteq [n]\times[n]$, $\mathcal{S}_j \subseteq [n]\times[n]$, for $i,j \in [n]$, such that the terms in* (7) *can be written as:*

$$\|\boldsymbol{A}\boldsymbol{P}-\boldsymbol{P}\boldsymbol{B}\|_p = \big(\textstyle\sum_{(i,j)\in\mathcal{I}} |f_{ij}(\boldsymbol{P}_{\mathcal{S}_{ij}})|^p\big)^{\frac{1}{p}}, \quad (8a)$$
$$\mathsf{tr}(\boldsymbol{P}^\top\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}) = \textstyle\sum_{(i,j)\in\mathcal{Q}} p_{ij}d_{ij}, \quad (8b)$$
$$\chi_{\mathcal{R}}(\boldsymbol{P}) = \textstyle\sum_{i\in[n]} \chi_{\mathcal{R}^{(i)}}(\boldsymbol{P}_{\mathcal{S}_i}), \quad (8c)$$
$$\chi_{\mathcal{C}}(\boldsymbol{P}) = \textstyle\sum_{j\in[n]} \chi_{\mathcal{C}^{(j)}}(\boldsymbol{P}_{\mathcal{S}_j}), \quad (8d)$$

where $f_{ij}(\cdot)$, $(i,j) \in \mathcal{I}$, are affine functions and

$$\mathcal{R}^{(i)} = \{\boldsymbol{p} \in [0,1]^{|\mathcal{S}_i|} | \mathbf{1}^\top\boldsymbol{p} = 1\}, \quad (9)$$
$$\mathcal{C}^{(j)} = \{\boldsymbol{p} \in [0,1]^{|\mathcal{S}_j|} | \mathbf{1}^\top\boldsymbol{p} = 1\}, \quad (10)$$

for $i \in [n], j \in [n]$, are the $|\mathcal{S}_i|$-dimensional and $|\mathcal{S}_j|$-dimensional simplices, respectively.

The proof can be found in Appendix A. Under this characterization, Prob. 3 becomes:

$$\min_{\boldsymbol{P}\in\mathcal{W}^n}\Bigg[\Big(\sum_{(i,j)\in\mathcal{I}} |f_{ij}(\boldsymbol{P}_{\mathcal{S}_{ij}})|^p\Big)^{\frac{1}{p}} + \sum_{(i,j)\in\mathcal{Q}} p_{ij}d_{ij} \quad (11a)$$
$$+ \sum_{i\in[n]} \chi_{\mathcal{R}^{(i)}}(\boldsymbol{P}_{\mathcal{S}_i}) + \sum_{j\in[n]} \chi_{\mathcal{C}^{(j)}}(\boldsymbol{P}_{\mathcal{S}_j})\Bigg]. \quad (11b)$$

The first term in (11a) (i.e., (8a)) *cannot* be written as a sum of functions, except when $p = 1$. Hence, it is not immediately obvious how to parallelize ADMM when $p \neq 1$. For $p = 1$, however, the entire objective can be written as a sum of constituent "local" objectives; hence, in this case, algorithm (6) can be directly parallelized. In all other cases however, we need a specialized implementation to parallelize the optimization of the term (8a).

The application of ADMM (6) to all the terms in Prob. (11) is summarized Alg. 1; primal-dual variable pairs:

$$(\boldsymbol{p}_{ij}, \boldsymbol{y}_{ij})_{(i,j)\in\mathcal{I}},\ (q_{ij}, \xi_{ij})_{(i,j)\in\mathcal{Q}},\ (\boldsymbol{r}_i, \boldsymbol{\psi}_i)_{i\in[n]},\ (\boldsymbol{c}_j, \boldsymbol{\phi}_j)_{j\in[n]},$$

correspond to terms (8a)-(8d), respectively. We note that Alg. 1 requires special care to handle term (8a) in the case $p > 1$; we describe how to address this case in the next two subsections.

A. Distributing Consensus ADMM for $p > 1$.

Applying consensus ADMM directly on (7) stumbles on the fact that the first term in the objective cannot be written as a sum; although the "local" optimization step (6a) of ADMM can be parallelized for all other terms, (6a) for this term (i.e., Line 12 of Alg. 1) takes the following form:

$$\min_{\boldsymbol{p}_{ij},i,j\in[n]}\Big(\sum_{(i,j)\in\mathcal{I}} |f_{ij}(\boldsymbol{p}_{ij})|^p\Big)^{\frac{1}{p}} + \frac{\rho}{2}\sum_{(i,j)\in\mathcal{I}} \|\boldsymbol{p}_{ij} - \boldsymbol{Z}^k_{\mathcal{S}_{ij}} + \boldsymbol{y}^k_{ij}\|_2^2 \quad (12)$$

Algorithm 1 Outer ADMM

1: **Input:** $A, B \in \{0,1\}^{n\times n}$, $D = D_{A,B} \in \mathbb{R}_+^{n\times n}$, $\mathcal{Q} \subseteq [n]\times[n]$
2: **Local primal & dual variables at processors in $V_{\texttt{obj}}$:** $(\boldsymbol{p}_{ij}, \boldsymbol{y}_{ij})_{(i,j)\in\mathcal{I}}$, $(q_{ij}, \xi_{ij})_{(i,j)\in\mathcal{Q}}$, $(\boldsymbol{r}_i, \boldsymbol{\psi}_i)_{i\in[n]}$, $(\boldsymbol{c}_j, \boldsymbol{\phi}_j)_{j\in[n]}$
3: **Consensus variables at processors in $V_{\texttt{var}}$:** $Z = [z_{ij}]_{(i,j)\in\mathcal{Q}}$
4: Initialize consensus variables and local/dual variables to 0;
5: Send copies of consensus variables z_{ij} to processors in $V_{\texttt{obj}}$
6: **while** not converged **do**
7: **if** $p = 1$ **then**
8: **for all** $(i,j) \in \mathcal{I}$ **in parallel do**
9: $\boldsymbol{p}_{ij} \leftarrow \underset{\boldsymbol{p}_{ij}\in\mathbb{R}^{|\mathcal{S}_{ij}|}}{\arg\min} \left(|f_{ij}(p_{ij})| + \frac{\rho}{2}\|\boldsymbol{p}_{ij} - \boldsymbol{Z}_{\mathcal{S}_{ij}} + \boldsymbol{y}_{ij}\|_2^2\right)$
10: **end for**
11: **else if** $p > 1$ **then**
12: Compute $\boldsymbol{p}_{ij}$, $(i,j) \in \mathcal{I}$, by solving (12) via Alg. 2
13: **end if**
14: **for all** $(i,j) \in \mathcal{Q}$ **in parallel do**
15: $q_{ij} \leftarrow \arg\min_{q_{ij}\in\mathbb{R}} \left(\lambda \cdot q_{ij} d_{ij} + (q_{ij} - z_{ij} + \xi_{ij})^2\right)$
16: **end for**
17: **for all** rows $i \in [n]$ and **all** columns $j \in [n]$ **in parallel do**
18: $\boldsymbol{r}_i \leftarrow \arg\min_{\boldsymbol{r}_i\in\mathbb{R}^{|\mathcal{S}_i|}} \left(\chi_{\mathcal{R}^{(i)}}(\boldsymbol{r}_i) + \frac{\rho}{2}\|\boldsymbol{r}_i - \boldsymbol{Z}_{\mathcal{S}_i} + \boldsymbol{\psi}_{ij}\|_2^2\right)$
19: $\boldsymbol{c}_j \leftarrow \arg\min_{\boldsymbol{c}_j\in\mathbb{R}^{|\mathcal{S}_j|}} \left(\chi_{\mathcal{C}^{(j)}}(\boldsymbol{c}_j) + \frac{\rho}{2}\|\boldsymbol{c}_j - \boldsymbol{Z}_{\mathcal{S}_j} + \boldsymbol{\phi}_{ij}\|_2^2\right)$
20: **end for**
21: Send local variables to processors in $V_{\texttt{var}}$
22: Update z_{ij}, $(i,j) \in \mathcal{Q}$, via averaging (6b)
23: Send copies of consensus variables z_{ij} to processors in $V_{\texttt{obj}}$
24: Update all dual variables via (6c)
25: **end while**
26: **return** consensus variables Z

Algorithm 2 Inner ADMM

1: **Input:** $\{\bar{z}_{ij} : (i,j) \in \mathcal{I}\}$
2: **Local primal & dual variables at $|\mathcal{I}|$ processors in $V_{\texttt{obj}}$:** $(\boldsymbol{p}_{ij}, u_{ij}, v_{ij})_{(i,j)\in\mathcal{I}}$,
3: Initialize $\boldsymbol{p}_{ij}$ to their previous values at the outer iteration, and dual variables v_{ij} to 0
4: **while** not converged **do**
5: Compute $\boldsymbol{u}$ by solving (14a) via Alg. 3
6: **for all** $(i,j) \in \mathcal{I}$ **in parallel do**
7: $\boldsymbol{p}_{ij} \leftarrow \underset{\boldsymbol{p}_{ij}\in\mathbb{R}^{|\mathcal{S}_{ij}|}}{\arg\min} \left(\frac{\rho}{2}\|\boldsymbol{p}_{ij} - \bar{\boldsymbol{z}}_{ij}\|_2^2 + \frac{\rho'}{2}(u_{ij} - f_{ij}(\boldsymbol{p}_{ij}) + v_{ij})^2\right)$
8: Update dual variable v_{ij} via (14c).
9: **end for**
10: **end while**
11: **return** consensus variables Z

where $\boldsymbol{p}_{ij} \in \mathbb{R}^{|\mathcal{S}_{ij}|}$ is the local vector containing coefficients corresponding to $\boldsymbol{Z}_{\mathcal{S}_{ij}}$ and $\boldsymbol{y}_{ij}^k \in \mathbb{R}^{|\mathcal{S}_{ij}|}$ is the dual variable corresponding to $\boldsymbol{p}_{ij} = \boldsymbol{Z}_{\mathcal{S}_{ij}}$. We rewrite this as:

$$\text{Minimize:} \quad \|\boldsymbol{u}\|_p + \frac{\rho}{2}\sum_{(i,j)\in\mathcal{I}} \|\boldsymbol{p}_{ij} - \bar{\boldsymbol{z}}_{ij}\|_2^2 \tag{13a}$$

$$\text{subj. to:} \quad u_{ij} = f_{ij}(\boldsymbol{p}_{ij}), \text{ for } (i,j) \in \mathcal{I}, \tag{13b}$$

where $\boldsymbol{u} = [u_{ij}]_{(i,j)\in\mathcal{I}} \in \mathbb{R}^{|\mathcal{I}|}$ is a vector of auxiliary variables corresponding to the the affine terms $f_{ij}(\boldsymbol{p}_{ij})$, and $\bar{\boldsymbol{z}}_{ij} \equiv \boldsymbol{Z}_{\mathcal{S}_{ij}}^k - \boldsymbol{y}_{ij}^k \in \mathbb{R}^{|\mathcal{S}_{ij}|}$, for $(i,j) \in \mathcal{I}$. As $f_{ij}(\cdot)$ are affine functions, so are the set of constraints. Then, we can also solve (13) w.r.t. $\boldsymbol{u}$ and $\boldsymbol{p}_{ij}$ via ADMM, where the steps are

$$\boldsymbol{u}^k = \underset{\boldsymbol{u}\in\mathbb{R}^{|\mathcal{I}|}}{\arg\min} \|\boldsymbol{u}\|_p + \frac{\rho'}{2}\sum_{(i,j)\in\mathcal{I}} (u_{ij} - f_{ij}(\boldsymbol{p}_{ij}^k) + v_{ij}^k)^2 \tag{14a}$$

$$\boldsymbol{p}_{ij}^{k+1} = \underset{\boldsymbol{p}_{ij}\in\mathbb{R}^{|\mathcal{S}_{ij}|}}{\arg\min} \frac{\rho}{2}\|\boldsymbol{p}_{ij} - \bar{\boldsymbol{z}}_{ij}\|_2^2 + \frac{\rho'}{2}(u_{ij}^{k+1} - f_{ij}(\boldsymbol{p}_{ij}) + v_{ij}^k)^2 \tag{14b}$$

$$v_{ij}^{k+1} = v_{ij}^k + (u_{ij}^{k+1} - f_{ij}(\boldsymbol{p}_{ij}^{k+1})) \quad (i,j) \in \mathcal{I}, \tag{14c}$$

Algorithm 3 p-norm Prox. Operator

1: **Input:** $\boldsymbol{w} \in \mathbb{R}^d$, $p \geq 1$, $\rho > 0$, $\varepsilon > 0$
2: Set $\hat{w}_i \leftarrow \rho|w_i|$ for $i = 1, \ldots, d$.
3: **if** $\|\hat{\boldsymbol{w}}\|_q \leq 1$ **then**
4: **return** $\boldsymbol{u}^* \leftarrow \boldsymbol{0}$
5: **end if**
6: Set $\boldsymbol{u} \leftarrow \boldsymbol{0}$, $s_L \leftarrow 0$, and $s_U \leftarrow \|\hat{\boldsymbol{w}}\|_p$
7: **for** $k = 1, \ldots, \log_2 \lceil \frac{1}{\varepsilon} \rceil$ **do**
8: Set $s \leftarrow (s_L + s_U)/2$
9: Compute $u_i \leftarrow \hat{w}_i g\left(s \cdot (\hat{w}_i)^{\frac{2-p}{p-1}}\right)$ for all $i \in \text{supp}(\hat{\boldsymbol{w}})$;
10: Compute $\|\boldsymbol{u}\|_p$;
11: **if** $\|\boldsymbol{u}\|_p < s$ **then**
12: Set $s_U \leftarrow s$
13: **else**
14: Set $s_L \leftarrow s$
15: **end if**
16: **end for**
17: Set $u_i^* \leftarrow \frac{\text{sign}\, w_i}{\rho} u_i$ for $i = 1, \ldots, d$.
18: **return** $\boldsymbol{u}^*$

where $\rho' > 0$ is a tuning parameter and $v_{ij} \in \mathbb{R}$, $i,j \in [n]$, are the dual variables corresponding to linear constraints (13b). Step (14b) comprises $|\mathcal{I}|$ quadratic problems, while (14c) is a simple adaptation; both can be executed *in parallel across the* $|\mathcal{I}|$ *processors* that store $\boldsymbol{p}_{ij}$, $\boldsymbol{y}_{ij}$, and which have received $\boldsymbol{Z}_{\mathcal{S}_{ij}}$ from the (outer) consensus ADMM step (line 23 of Alg. 1). In contrast, it is not apriori clear how to parallelize step (14a); as in the case of the outer ADMM, this is due to the $\|\cdot\|_p$ term: we present our algorithm solving (14a) in parallel (Alg. 3) next.

The pseudocode for this inner ADMM step is presented in Alg. 2. The code is executed in parallel across the $|\mathcal{I}|$ machines described above. Note that steps (14b) and (14c) are executed in parallel but require no communication; hence, all communication in Alg. 2 is the one needed by Alg. 3 to compute $\boldsymbol{u}$; as we discuss in the next section, this amounts to a logarithmic number of map and reduce operations.

B. Parallel p-norm Proximal Operator

For $p > 1$, motivated by (14a), we consider the problem:

$$\min_{\boldsymbol{u}\in\mathbb{R}^d} \|\boldsymbol{u}\|_p + \tfrac{\rho}{2}\|\boldsymbol{u} - \boldsymbol{w}\|_2^2, \tag{15}$$

for a given $\boldsymbol{w} \in \mathbb{R}^d$ where $d \equiv |\mathcal{I}|$. In doing so, we assume that, as is the case in (14a), the elements of vector $\boldsymbol{w}$ are distributed across d machines, that need to collectively solve (15) in parallel. Following Liu and Ye [54], we define first a non-negative vector $\hat{\boldsymbol{w}}$ via

$$\hat{w}_i = \rho|w_i|. \tag{16}$$

We then consider the following simpler problem:

$$\min_{\boldsymbol{u}\in\mathbb{R}_+^d} \|\boldsymbol{u}\|_p + \tfrac{1}{2}\|\boldsymbol{u} - \hat{\boldsymbol{w}}\|_2^2. \tag{17}$$

Note that this differs from Prob. (15) in that (a) $\rho = 1$ and (b) vector $\boldsymbol{w} \in \mathbb{R}^d$ replaced with non-negative vector $\boldsymbol{w} \in \mathbb{R}_+^d$, and (c) optimization happens over $\boldsymbol{u} \in \mathbb{R}_+^d$. Nevertheless, Prob. (15) is equivalent to Prob. (17) (see Lemma B.4 below). In particular, if $\hat{\boldsymbol{u}}$ is the optimal solution of (17), the optimal solution to (15) is given by $\boldsymbol{u}^*$ such that:

$$u_i^* = \frac{\text{sign}(w_i)}{\rho}\hat{u}_i, \quad \text{for } i \in [d]. \tag{18}$$

We therefore turn our attention to solving Prob. (17). To do so, we define first an auxiliary function. Given $\alpha \in (0, \infty)$, define the function $\alpha \mapsto g(\alpha)$, as the unique solution of the following equation over $x \geq 0$:

$$(x/\alpha)^{p-1} + x - 1 = 0, \tag{19}$$

We extend g to $[0, \infty)$ by setting $g(0) \equiv 0$ for $\alpha = 0$, by definition. Function g is hard to express in closed form,[2] but it is well-defined. This is because, for $\alpha > 0$, the l.h.s is -1 for $x = 0$ and positive for $x = \min(1, \alpha)$. Hence, by the intermediate value theorem, Eq. (19) always has a positive solution between 0 and $\min(1, a)$; uniqueness is implied by the strict monotonicity of the l.h.s. of Eq. (19) in x. Hence, $g : \mathbb{R}_+ \to [0, 1]$ is indeed well-defined.

Having defined g, given a vector $\hat{\boldsymbol{w}} \in \mathbb{R}^d_+$, we define functions $g_i : \mathbb{R}_+ \to \mathbb{R}_+, i \in [d]$ as:

$$g_i(s) = \hat{w}_i \cdot g\big(s \cdot (\hat{w}_i)^{\frac{2-p}{p-1}}\big), \tag{20}$$

as well as function $h : \mathbb{R}_+ \to \mathbb{R}$ as:

$$h(s) = \big(\textstyle\sum_{i=1}^d g_i(s)^p\big)^{\frac{1}{p}} - s. \tag{21}$$

The optimal solution to Prob. (17) can be determined w.r.t. a root of equation $h(s) = 0$. In particular, the following holds:

Theorem IV.2 (Liu and Ye [54])**.** *Given $\hat{\boldsymbol{w}} \in \mathbb{R}^d_+$ and $p > 1$, let $\hat{\boldsymbol{u}} \in \mathbb{R}^d_+$ be an optimal solution to Prob.* (17)*. Let also $q \in \mathbb{R}_+$ be such that $\frac{1}{p} + \frac{1}{q} = 1$. Then, $\hat{\mathbf{u}}$ is unique, and:*

- *If $\|\hat{\boldsymbol{w}}\|_q \leq 1$, then $\hat{\boldsymbol{u}} = \mathbf{0}$.*
- *If $\|\hat{\boldsymbol{w}}\|_q > 1$, then*

$$\hat{u}_i = g_i(s^*), \quad \textit{for } i \in [d], \tag{22}$$

where s^* is the unique value in $(0, \|\hat{\boldsymbol{w}}\|_p]$ s.t. $h(s^*) = 0$.

Intuitively, the above theorem suggests that there are two cases we need to consider. The first, "easy" case, is when $\|\hat{\boldsymbol{w}}\|_q \leq 1$: then, the optimal solution is $\mathbf{0}$. If $\|\hat{\boldsymbol{w}}\|_q > 1$, i.e., on the "hard case", solving Prob. (17) is tantamount to finding the unique, scalar root $s^* \in (0, \|\hat{\boldsymbol{w}}\|_p]$ of the equation:

$$h(s) = 0, \text{ where } h \text{ is given by Eq. (21).}$$

This is because, once this root s^* is computed, the optimal solution $\hat{\boldsymbol{u}} \in \mathbb{R}^d$ can be constructed via Eq. (22), by computing $g_i(s^*)$ for every $i \in [d]$.

Crucially, a root of h can be found with a simple bisection algorithm, *the steps of which can be easily parallelized via map and reduce operations.* This bisection algorithm, summarized in Alg. 3, proceeds as follows: given $\hat{\boldsymbol{w}} \in \mathbb{R}^d_+$, we test whether the condition $\|\hat{\boldsymbol{w}}\|_q \leq 1$ holds; if so, we return $\boldsymbol{u}^* = \mathbf{0}$. Otherwise, we find s^* via bisecting $[0, \|\hat{\boldsymbol{w}}\|_p]$. That is, at each iteration, we maintain an upper (s_U) and lower (s_L) bound on s^*, initialized at the above values. By construction, function h alternates signs on each of the two bounds: i.e., $h(s_L)h(s_U) \leq 0$; at each iteration, we (a) compute the average $s = 0.5(s_L + s_U)$, between the two bounds, (b) find the sign of h on this average, and then (c) update the bounds accordingly.

[2]Though its inverse g^{-1} is easy to describe explicitly; see Eq. (31).

As signs alternate, by the intermediate value theorem, s^* is guaranteed to be in $[s_L, s_U]$ at all times.

Another way to get some intuition behind how Alg. 3 behaves in the "hard" case is the following. At any iteration, s is compared to p-norm of the current solution $\boldsymbol{u} \in \mathbb{R}^d$. If $\|\boldsymbol{u}\| < s$, then s is too big, and we search at a smaller value; if the opposite is true, we search for a larger value, always adjusting the bounds accordingly. At all times, we set $\boldsymbol{u}$ by following the trajectory in $\mathbb{R}^d$ determined by functions g_i, linking the current s to the $\boldsymbol{u}$.

Since Alg. 3 ensures that the root s^* is be within $[s_L, s_U]$ at all times; Liu and Ye show that the algorithm thus approximate s^* within ε accuracy by performing $\log_2 \varepsilon^{-1}$ bisections [54]. We show this implies the following convergence guarantee:

Theorem IV.3. *Alg. 3 outputs a solution $\boldsymbol{u} \in \mathbb{R}^d$ such that $\|\boldsymbol{u} - \hat{\boldsymbol{u}}\|_p \leq \sqrt[p-1]{\|\hat{\boldsymbol{w}}\|_q} \cdot \|\hat{\boldsymbol{w}}\|_p \cdot \varepsilon$.*

Hence, Alg. 3 can approximate the optimal solution within arbitrary accuracy within a logarithmic number of iterations. Finally, it is easy to see that the computations involved in Alg. 3 can be parallelized across the d processors that store the values w_i, $i \in [d]$. Given an s, the computation of values $\hat{u}_i$ can happen in parallel via an application of Eq. (22) at each $\hat{w}_i$ (Line 9). Moreover, the p-norm of u (Line 10), needed to compute the sign of $h(s)$, can be computed via a reduce; the updated value of s can subsequently be broadcast to processors, to initiate the next iteration. We further elaborate on parallelism in Section V, where we discuss the computation and communication complexity of the entire process, combining Algorithms 1,2, and 3. For completeness, we provide proofs of Theorems IV.2 and IV.3 in Appendix B.

V. Parallel Complexity

ADMM is a first-order method, and its convergence is $O(\frac{1}{k})$ [76]. All dual variable adaptations are linear in their input sizes and so are averaging operations involved in consensus variable computations; both are parallelized. All primal variable adaptations are convex optimization problems with self-concordant objectives, either unconstrained or linearly constrained; as such, they can generically be solved within accuracy ϵ by interior point methods in steps that are polylogarithmic in $1/\epsilon$, with each step being polynomial in the input size. In particular:

- When $p = 1$, updating $\boldsymbol{p}_{ij}$, $(i, j) \in \mathcal{I}$ involves solving a generalized lasso regression problem (Line 9 of Alg. 1), which can be solved using the algorithm proposed by Tibshirani [77]. Alternatively, the inner ADMM Eq. (14) can again be applied; Eq. (14a) then amounts to $|\mathcal{I}|$ softmax operations (each at $O(n)$ cost for computing f_{ij}). All all trivially parallelizable via a `map`.
- The row and column updates (Lines 18 and 19 of Alg. 1) amount to orthogonal projections on the simplex; we use the strongly polynomial algorithm by Michelot [78], which has complexity $O(n \log n)$. There are a total of n (one per row/column) such operations, all of which can again be parallelized via a `map` applying Michelot's algorithm.
- The optimization of the trace term involves $|\mathcal{Q}|$ one-dimensional quadratic problems (Line 15 of Alg. 1), which

have a closed form and can be computed in $O(1)$ time. Again, these operations can be parallelized via a `map`; in practice, however, we avoid these computations altogether by "completing the squares" and incorporating these terms along with the column and row projections, as adjustments to the vectors projected to the simplices.

- The update of $\boldsymbol{p}_{ij}$ in Alg. 2 is an unconstrainted convex quadratic program that has a closed form solution. Each of these $|\mathcal{I}|$ such operations can be computed in $O(|S_{ij}|) = O(n)$ time; again, they can be parallelized over $|\mathcal{I}|$ processors via a `map`.
- The norm computations in Alg. 3 (Lines 3 and 10) depend on vector size $|\mathcal{I}|$ and can be parallelized via a `reduce`. Updates in Line 9 are $O(1)$ for each of the $|\mathcal{I}|$ coordinates;[3] this is parallelizable over $|\mathcal{I}|$ processors, via a `map`.
- There are a total of $|\mathcal{I}| + |\mathcal{Q}| + 2n$ outer and $|\mathcal{I}|$ inner dual adaptations, they are all $O(1)$ and parallelizable via a `map`.
- The consensus averaging step involves $|\mathcal{V}_{\texttt{var}}| = |\mathcal{Q}|$ scalar summations, adding in total of $|\mathcal{E}_{\mathcal{G}}|$ terms, where $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$ is the bipartite graph (illustrated in Fig. 1) induced by our problem. Parallelizing this involves message passing between the nodes storing all $\mathcal{V}_{\texttt{obj}}$ objectives and the $|\mathcal{Q}|$ processors storing the consensus values, with the total number of messages passed being $|\mathcal{E}_{\mathcal{G}}|$. Below, we establish bounds on all these quantities.

Putting everything together, assuming the number of iterations of the outer and inner ADMM are $k_1, k_2 \in \mathbb{N}$, respectively, and that the accuracy used in Alg. 3 is ε, the serial complexity of our algorithm is:

$$k_1 k_2 \big[O\big(|\mathcal{I}|\big(n + \log \frac{1}{\varepsilon}\big)\big) + O\big(n^2 \log n\big)\big] + k_1 O(|\mathcal{E}_{\mathcal{G}}|). \quad (23)$$

Assuming access to $\max(|\mathcal{I}|, n, |\mathcal{Q}|)$ processors, each inner iteration (first term in Eq. (23)) can be fully implemented via a constant number of `map` and `reduce` operations over these processors, with maps involving operations of at most $O(n \log n)$ complexity, and reduces terminating within $O(\log |\mathcal{I}|)$ rounds. On the other hand, the consensus step (second step in Eq. (23)) can be done via message passing between the processors corresponding to nodes of graph $\mathcal{G}$. The parallel complexity of the algorithm depends on the size of set $\mathcal{E}_{\mathcal{G}}$. In particular, we would like to determine conditions under which $\mathcal{G}$ is sparse. We therefore turn our attention to bounding the sparsity of $\mathcal{G}$ of the problem input size.

A. Characterizing the Sparsity of $\mathcal{G}$

The induced bipartite graph $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$, as illustrated in Fig. 1, depends on the number of terms that appear in the problem objective (determining $V_{\texttt{obj}}$) as well as on the number of times each variable appears in each such term (determining $\mathcal{E}_{\mathcal{G}}$). We first bound the size of $\mathcal{V}_{\texttt{obj}}$:

Lemma V.1. *Let $\boldsymbol{E} \in \{0,1\}^{n \times n}$ be the binary matrix whose support is $\mathcal{Q}$, and let $m_0 \equiv \|\boldsymbol{AE} + \boldsymbol{EB}\|_0$. Then, the summation inside the first term* (8a) *of objective* (7) *contains $|\mathcal{I}| \leq m_0$ terms; collectively, the remaining three terms* (8b)-(8d) *contain at most $|\mathcal{Q}| + 2n$ terms.*

The proof is in Appendix C-A. Lemma V.1 immediately implies that the bipartite graph $\mathcal{G}(\mathcal{V}_{\texttt{obj}}, \mathcal{V}_{\texttt{var}}, \mathcal{E}_{\mathcal{G}})$ satisfies:

$$|\mathcal{V}_{\texttt{obj}}| \leq m_0 + |\mathcal{Q}| + 2n \quad \text{and} \quad |\mathcal{V}_{\texttt{var}}| = |\mathcal{Q}|. \quad (24)$$

Our next lemma characterizes $|\mathcal{E}_{\mathcal{G}}|$:

Lemma V.2. *The supports of $f_{ij}(\cdot), \chi_{\mathcal{R}}(\cdot), \chi_{\mathcal{C}}(\cdot)$ satisfy:*

$$\textstyle\sum_{(i,j)\in\mathcal{I}} |\mathcal{S}_{ij}| \leq \min(n|\mathcal{E}_A|, n|\mathcal{Q}|) + \min(n|\mathcal{E}_B|, n|\mathcal{Q}|), \quad (25a)$$

$$\textstyle\sum_{i\in[n]} |\mathcal{S}_i| \leq |\mathcal{Q}|, \ \sum_{j\in[n]} |\mathcal{S}_j| \leq |\mathcal{Q}|. \quad (25b)$$

The support of functions $f_{ij}(\cdot)$ is also bounded by:

$$\textstyle\sum_{i,j\in[n]} |\mathcal{S}_{ij}| \leq m_0 \left(\max(d_A, d_{\mathcal{Q}}) + \max(d_B, d_{\mathcal{Q}})\right), \quad (25c)$$

where d_A, d_B, and $d_{\mathcal{Q}}$ denote the maximum degrees of graphs G_A, G_B, and $G([n], [n], \mathcal{Q})$, respectively.

The proof is in Appendix C-B. Lemma V.2 implies that the number of edges in $\mathcal{G}$ is:

$$|\mathcal{E}_{\mathcal{G}}| \leq M + 3|\mathcal{Q}|, \quad (26)$$

where M is the minimum among the bounds in (25a) and (25c). Hence, Eq. (24) and (26) together provide conditions under which when $\mathcal{G}$ is sparse. This happens if, e.g., $m_0 = O(n^2)$ and both G_A and G_B are sparse: by Eq. (25a), graph $\mathcal{G}$ would then have a number of edges that is $O(\mathcal{V}_{\texttt{obj}} + \mathcal{V}_{\texttt{var}})$. Alternatively, the same occurs when d_A, d_B, and $d_{\mathcal{Q}}$ are bounded (by Eq. (25c)).

VI. Experiments

A. Experimental Setup

Implementation. We implemented Alg. 1 over Spark (version 2.3.2), an open-source cluster-computing framework [26], via its Python interface (version 2.7.15). We also implemented Alg. 1 in Ansi C (glibc version 2.23), using OpenMP (version 4.0) and Atlas (version 3.10.2).

Execution environment. We run Spark on a local cluster that comprises 8 machines. Each machine has 2 Intel(R) Xeon(R) CPUs (E5-2680 v4) with 14 cores, and the cluster has $8 \times 28 = 224$ cores in total. We run OpenMP on the Google Cloud Platform[4] and on a `n1-standard-96` machine with 96 (virtual) cores and 360GB RAM.

Metrics. We report the objective as well as the primal and dual residuals as the iterations of our ADMM algorithm progress. The latter measure convergence. We evaluate the optimality of our solution by a parameter $\epsilon \in \mathbb{R}$ defined as: $\epsilon = \max\left(\frac{\|r^K\|_2}{\sqrt{|\mathcal{I}|}}, \frac{\|s^K\|_2}{\sqrt{|\mathcal{I}|}}\right)$, where r^K and s^K are the primal and dual residuals [24] at the last iteration . The smaller ϵ is, the closer the solution is to the optimal.

Datasets. We experiment on several real graphs from the Network Repository[5] and the Stanford Large Network Dataset Collection[6], which we summarize in Table II. Four graphs `bnm1`, `bnm2`, `bnc1`, and `bnc2` are brain networks. `ptn1`

[3] Function g can be computed efficiently at an arbitrary accuracy as it is strictly monotone and g^{-1} has a closed form.

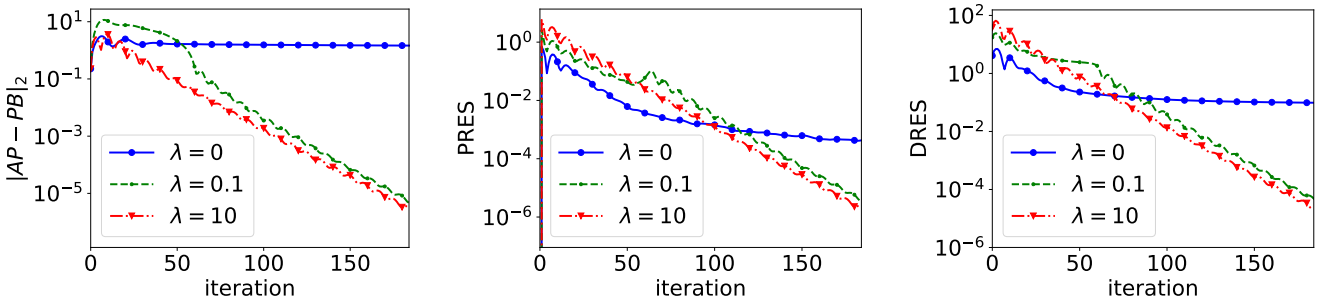
[4] https://cloud.google.com

[5] http://networkrepository.com

[6] https://snap.stanford.edu/data/index.html

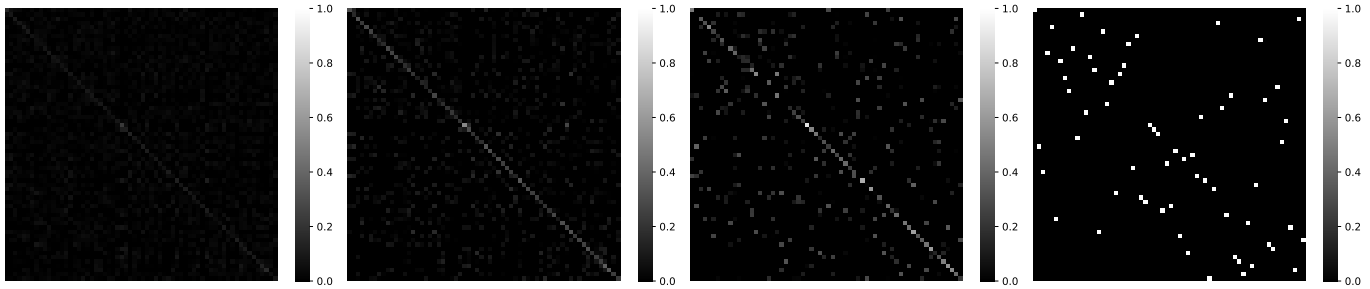
pair	(G_A, G_B)	$\lvert\mathcal{V}_A\rvert, \lvert\mathcal{V}_B\rvert$	$\lvert\mathcal{E}_A\rvert, \lvert\mathcal{E}_B\rvert$	$\mathcal{Q}$	$\lvert\mathcal{Q}\rvert$	$\lvert\mathcal{I}\rvert$
cortex	(bnc1, bnc2)	91, 93	1.9K, 2.6K	all	8.6K	8.6K
monkey	(bnm1, bnm2)	242, 91	4K, 628	all	58.5K	58.5K
protein	(ptn1, ptn2)	1.5K, 1.8K	2K, 4K	degree	3.3M	3.3M
retweet	(rt1, rt2)	3.2K, 3.2K	3.4K, 3.9K	degree	5.9M	2.7M
deezer	(dzr1, dzr2)	41.8K, 47.5K	125.8, 222.8K	WL2	1.1M	2.9M
slashdot	(sld1, sld2)	77K, 82K	828K, 870K	WL3	98K	2M

Table II: A summary of real graph pairs along with the preprocessing method for generating $\mathcal{Q}$.



(a) Norm **(b)** Primal Residual **(c)** Dual Residual

Figure 2: Effects of adding the linear term on the convergence. The plot shows the traces for the objective and the primal and dual residual throughout ADMM iterations (Alg. 1 for $p = 2$) for different λ. We observe that giving a larger coefficient to the linear term makes the convergence considerably faster.



(a) $p = 5, \lambda = 0.0$, **DPM**=0.097 **(b)** $p = 5, \lambda = 0.1$, **DPM**=0.31 **(c)** $p = 5, \lambda = 1.0$, **DPM**=0.36 **(d)** only linear term, **DPM**=0.32

Figure 3: Effects of adding linear term on solutions for $p = 5$ and BRN noise. We assess that adding the linear term $\lambda = 0.1$ increases the values on the diagonal. However, increasing λ further to 1.0 makes solution highly biased on the extracted node features; this is obvious from the non-diagonal gray elements.

	BRN		OUT1		OUT2	
(p, λ)	DPM	DPMP	DPM	DPMP	DPM	DPMP
(1, 0.0)	**0.99**	**1.0**	0.32	1.0	0.08	0.98
(1, 0.1)	0.99	1.0	0.33	1.0	0.05	0.45
(1, 1.0)	0.97	0.97	0.33	1.0	0.05	0.45
(1.5, 0.0)	0.14	1.0	0.42	1.0	**0.24**	**0.99**
(1.5, 0.1)	0.27	0.98	**0.43**	**1.0**	0.23	0.98
(1.5, 1.0)	0.49	0.97	0.38	1.0	0.10	0.82
(2, 0.0)	0.12	1.0	0.35	1.0	0.18	0.98
(2, 0.1)	0.25	0.97	0.08	0.97	0.06	0.90
(2, 1.0)	0.3	0.95	0.04	0.45	0.06	0.91
(3, 0.0)	0.11	1.0	0.25	0.98	0.04	0.24
(3, 0.1)	0.29	0.97	0.01	0.03	0.01	0.05
(3, 1.0)	0.42	0.82	0.01	0.02	0.01	0.05
(5, 0.0)	0.097	0.98	0.3	1.0	0.04	0.24
(5, 0.1)	0.31	0.92	0.01	0.006	0.01	0.04
(5, 1.0)	0.36	0.58	0.01	0.01	0.01	0.04
(N/A, 1.0)	0.32	0.32	0.02	0.02	0.01	0.009

Table III: Comparison of different p-norms and λ coefficients for the linear term for BRN, OUT1, and OUT2. When considering only the linear term, we denote p value by N/A and report the corresponding results in the last row.

and ptn2 are biological networks, while rt1 and rt2 are re-tweet networks. The nodes in dzr1 and dzr2 are users of the music streaming service Deezer for two different countries, where edges show friendship. The graphs sld1 and sld2 represent social interactions between the users of the website Slashdot. We also experiment on synthetic Erdős Rényi, ER(n, q), graphs with n nodes and edge probability q, where n ranges from 2^6 to 2^{17}.

	GSS		LPC		MIX	
(p, λ)	DPM	DPMP	DPM	DPMP	DPM	DPMP
(1, 0.0)	0.12	0.99	0.08	0.87	0.16	1.0
(1.5, 0.0)	0.14	1.0	0.8	1.0	0.19	1.0
(2, 0.0)	0.15	1.0	0.09	1.0	0.20	1.0
(3, 0.0)	0.16	1.0	0.09	0.98	0.21	1.0
(5, 0.0)	**0.163**	**1.0**	**0.1**	**1.0**	**0.22**	**1.0**

Table IV: Comparison of different norms for GSS, LPC, and MIX. Here we do not report results for $\lambda \neq 0$, as the second graph is weighted and fully connected.

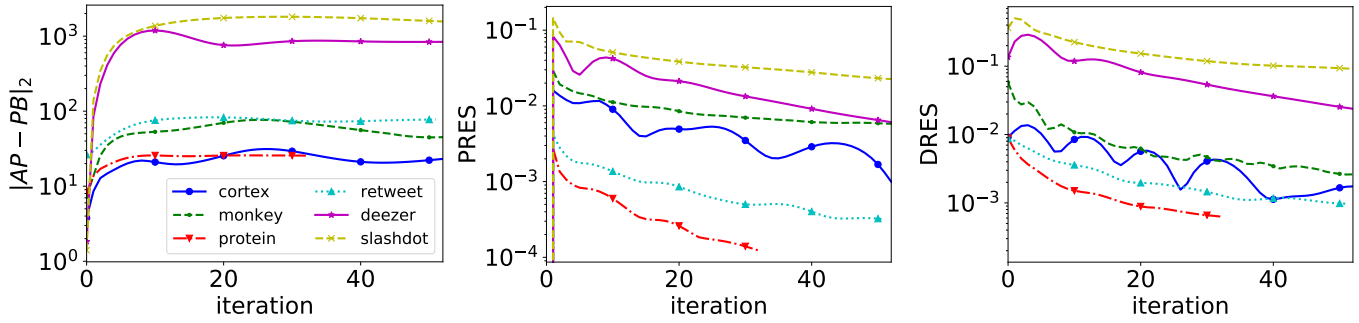
Preprocessing. For real graphs we use 4 node features: the size of the first-hop and second-hop neighborhoods, the number of paths of length 2, and their pagerank. For synthetic graphs, in addition to these 4 features, we also compute the number of paths and cycles of length 3 along with the size of the third-hop neighborhood. Given two graphs, we construct the dissimilarity matrix $\boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}$ using the ℓ_2 distance between features. We construct the constraint set $\mathcal{Q}$ for real graphs

	Dense						Sparse									
n	2^{10}		2^{11}		2^{12}		2^{13}		2^{14}		2^{15}		2^{16}		2^{17}	
$\lvert\mathcal{E}_A\rvert,\lvert\mathcal{E}_B\rvert$	5.2K,5.2K		20.9K,21K		84K,83.8K		49K,48.7K		103K,104K		220K,221K		464K,465K		980K,981K	
$\lvert\mathcal{Q}\rvert,\lvert\mathcal{I}\rvert$	10.4K,194.9K		42K,1.4M		168K,9.4M		48.7K,1.1M		103K,2.5M		219K,5.8M		465K,13M		979K,29M	
CPUs	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)
OpenMP																
1	8	4	126	39	1076	412	△		-	-	-	-	-	-	-	-
10	10	0.4	145	4	1074	43	△		-	-	-	-	-	-	-	-
20	9	0.23	121	2	1045	19	△		-	-	-	-	-	-	-	-
30	9	0.17	121	1	953	12	△		-	-	-	-	-	-	-	-
Spark																
1	803	680	6095	9570	■		4173	2891	11155	6339	22183	15250	■		■	
10	288	88	639	1139	■		2147	357	6419	769	12009	1896	■		■	
20	257	56	370	688	■		2029	216	6208	463	11435	1175	■		■	
30	257	56	302	559	■		1984	177	5869	392	10762	975	■		■	
56	294	79	293	402	■		1963	154	6000	338	10713	910	■		■	
448	75	26	52	62	1123	437	290	34	806	67	1409	135	4200	250	10990	790

Table V: Scaling results for Alg. 1 and $\|\boldsymbol{AP}-\boldsymbol{PB}\|_2^2$ objective ($\lambda=0$). In this case Alg. 1 skips the inner loop Alg. 2 as the objective is separable. We run our ADMM algorithms using both OpenMP and Spark implementations on the synthetic graphs. t_{IT} is the average over 5 iterations. We denote the cases that the execution ran out of memory by ■ and ones that produced a segmentation fault with △.

	Dense						Sparse									
n	2^{10}		2^{11}		2^{12}		2^{13}		2^{14}		2^{15}		2^{16}		2^{17}	
$\lvert\mathcal{E}_A\rvert,\lvert\mathcal{E}_B\rvert$	5.2K,5.2K		20.9K,21K		84K,83.8K		49K,48.7K		103K,104K		220K,221K		464K,465K		980K,981K	
$\lvert\mathcal{Q}\rvert,\lvert\mathcal{I}\rvert$	10.4K,194.9K		42K,1.4M		168K,9.4M		48.7K,1.1M		103K,2.5M		219K,5.8M		465K,13M		979K,29M	
CPUs	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)	t_{SU}(s)	t_{IT}(s)
56	63	642	1566	11468	■		101	3040	218	6475	495	15359	■		■	
448	21	222	60	1490	584	5463	35	455	44	907	98	2070	196	4327	565	9121

Table VI: Scaling results for Alg. 1 and $\|\boldsymbol{AP}-\boldsymbol{PB}\|_2$ objective ($\lambda=0$). In comparison to Table V, we see that in general the iteration time is longer because each iteration of Alg. 1 executes the inner ADMM loop Alg 2 (for 60 iterations).



(a) Norm **(b)** Primal Residual **(c)** Dual Residual

Figure 4: Traces of our ADMM algorithm for $p=2$ and real graphs pairs. For `cortex`, `retweet`, `deezer`, and `slashdot` pairs λ is set to 0.001, while for `monkey` and `protein` it is 0.1. We run the inner loop (Alg. 2) for 60 iterations. The average iteration time (of Alg. 1) for `cortex`, `monkey`, `protein`, `retweet`, `deezer`, and `slashdot` is 364(s), 712(s), 3836(s), 4375(s), 3797(s), and 2290(s), respectively.

by one of the following methods, which we report along with the size of the set $\mathcal{Q}$ in Table II.

- `all`: $\mathcal{Q}$ includes all pairs $(i,j)\in\mathcal{V}^2$, i.e., $|\mathcal{Q}|=n^2$.
- `degree`: $\mathcal{Q}$ includes pairs $(i,j)\in\mathcal{V}^2$, s.t., the nodes i and j have the same node degree.
- `WL`k: $\mathcal{Q}$ includes pairs $(i,j)\in\mathcal{V}^2$, s.t. i,j have the same color: we generate node colors by running the Weisfeiler-Lehman (WL) algorithm [74] (see also Sec. 2 in [13]) for k iterations.

For synthetic graphs we use `all` for generating $\mathcal{Q}$.

B. Experimental Results

Linear Term. We begin by studying the effects of the linear term in (3) on convergence. G_A is an ER$(64,0.1)$ and G_B is a random permutation of G_A. We show the trace of residuals and the norm throughout iterations of ADMM for $\lambda=0,0.1,10$ in Fig. 2. We run ADMM for each λ value for a fixed number of iterations (160). In all cases the optimal solution is the permutation matrix used to generate G_B from G_A, so the optimal objective value is 0. Note that in Fig. 2 all of the objective values as well as the residuals indeed converge to zero. We see that non-zero λ values significantly accelerate convergence, as the linear term directs the algorithm faster to the correct solution. The logarithm-scaled plot accentuates the faster linear convergence of ADMM for $\lambda=0.1,10$ in comparison with slower sub-linear convergence for $\lambda=0$.

***p*-norms.** We next assess the quality of the computed doubly stochastic matrix $\boldsymbol{P}$ w.r.t. p-norms. We let again G_A be ER$(64,0.1)$. We generate another graph G_B by adding different noise types to G_A according to the following scenarios:

- Bernoulli noise (`BRN`): We flip each element in $\boldsymbol{A}$ with probability 0.01.
- Outliers (`OUT`k): We choose k nodes (*outliers*) from G_A uniformly at random and make them connected to every other node in G_B. We experimented with $k=1,2$ outliers, (i.e, `OUT1` and `OUT2`, respectively).

- Gaussian noise (GSS): We add i.i.d. zero-mean Gaussian noise with variance 0.01 to the elements of A.
- Laplacian noise (LPC): We add i.i.d. zero-mean Laplacian noise with variance 0.01 to the elements of A.
- Mixture of Gaussian and Laplacian noise (MIX): We add a mixture of i.i.d. zero-mean Gaussian and Laplacian random variables with 0.01 variance; the mixture coefficients are equally set to $\sqrt{0.5}$ so that the variance is 0.01.

We obtain the matrix $\boldsymbol{P}$ by solving (3) for different pair (p, λ) values. As G_B is a perturbed version of G_A, i.e., generated via the addition of noise, we measure the quality of the resulting solution by how far it deviates from the identity. As metrics, we report the Diagonal Probability Mass ($\mathbf{DPM}$) of $\boldsymbol{P}$, defined as the sum of the diagonal elements normalized by $n = 64$, and the Diagonal Probability Mass after Projection ($\mathbf{DPMP}$) on the set of permutation matrices $\mathcal{P}^n$, i.e., $\mathbf{DPM} = \frac{1}{n}\sum_{i\in[n]} \boldsymbol{P}_{ii}$ and $\mathbf{DPMP} = \frac{1}{n}\sum_{i\in[n]} \boldsymbol{P}^{\boldsymbol{\pi}}_{ii}$, where $\boldsymbol{P^\pi} = \arg\min_{\boldsymbol{P'}\in\mathcal{P}^n} \|\boldsymbol{P} - \boldsymbol{P'}\|_2^2$.

We report results for the first three cases with unweighted edges, i.e., BRN, OUT1, and OUT2 in Table III. Note that for other cases, i.e., GSS, LPC, and MIX, the edges in G_B are weighted and thusly G_B is fully connected, so we do not add the linear term ($\lambda = 0$). We report results for the latter in Table IV. The reported results are averaged over 5 random runs; graphs G_A are generated independently at random for each run, then G_B is generated following the scenarios outlined above.

We make the following observations from Table III. We first concentrate on BRN, reported in the first column of the table. We see that in the absence of the linear term ($\lambda = 0$), $p = 1$ has a superior performance and recovers the identity matrix. We further observe that for other p values adding the linear term ($\lambda = 0.1, 1$) improves the solution. For instance, by comparing metrics for $\lambda = 0$ and $\lambda = 0.1$ in the first column (BRN case) we see that adding the linear term generally increases $\mathbf{DPM}$, while $\mathbf{DPMP}$ stays almost the same (above 0.9). However, increasing λ further to 1 decreases $\mathbf{DPMP}$ significantly. The reason is that increasing λ makes the solution highly biased on the extracted features. Motivated by this observation we also tested the case with only the linear term, where p is denoted by N/A, we see that $\mathbf{DPM}$ and $\mathbf{DPMP}$ values are grossly inferior, in comparison to other cases. These observations suggest that there is a trade-off between the first norm term, and the second linear term in (3); the linear term can improve the solution by incorporating node features; however, using a high λ values makes the results highly dependent on the crafted node features.

To make this point more vivid we visualize solutions as heatmaps for $p = 5$ and different λ values and the case with only linear term in Fig. 3; from the figure we see that increasing λ from 0 to 0.1 increases the overall diagonal values (see Fig 3a and 3b). We see that increasing λ further to 1 slightly increases the diagonal mass but also increases non-diagonal values (see Fig. 3c); by comparing Fig. 3c and Fig. 3d we see that the non-diagonal elements in the former corresponds to the solution generated by adding the linear term (see the non-diagonal elements in Fig. 3c and Fig. 3d).

For the two types of outliers (OUT1 and OUT2) reported in next columns of Table III, we see that $p = 1.5$ outperforms other p values. This is in contradiction to the previous case (BRN), where $p = 1$ outperformed other p values. We further observe that despite the case of BRN adding the linear term only deteriorates solutions. The reason is that adding outlier nodes adversely interfere with the extracted node features that are all dependent on degree and neighborhood information.

Finally we report results for other noise types with weighted edges, i.e., GSS, LPC, and MIX in Table IV. Here all p norms have comparable performances; they all achieve $\mathbf{DPMP} = 1$, but $\mathbf{DPM}$ is slightly higher for higher p values.

Scalability. We evaluate the scalability of our proposed ADMM algorithm w.r.t. the graph size n and the number of CPUs. We report the results for two different objectives $\|\boldsymbol{AP} - \boldsymbol{PB}\|_2^2$ and $\|\boldsymbol{AP} - \boldsymbol{PB}\|_2$ with $\lambda = 0$, in Tables V and VI, respectively. The two problems are mathematically equivalent. However, $\|\boldsymbol{AP} - \boldsymbol{PB}\|_2^2$ has a separable form, therefore, Alg. 1 skips the inner loop (Alg. 2); in this case, Line 9 in Alg. 1 is an unconstrained quadratic problem, which has a closed form solution. In particular, we report setup time $t_{\texttt{SU}}$ and iteration time $t_{\texttt{IT}}$. $t_{\texttt{SU}}$ includes the time spent creating and initializing the variables, i.e., Lines 2 to 4 in Alg. 1. $t_{\texttt{IT}}$ is the average iteration time of Alg. 1, i.e., Lines 7 to 24. ADMM is a first-order-method and it usually needs ≈100 iterations to converge. Therefore, the iteration time dominates the setup time, but for completeness we report setup times too. In this experiment G_A, G_B, and $\mathcal{Q}$ are Erdős Rényi graphs. We experiment with two settings, i.e., (a) "dense" graphs ER$(n, 0.01)$ ($n = 2^{10}$ to 2^{12}) and (b) "sparse" graphs ER$(n, 1.1 \log n/n)$ for $n = 2^{13}$ to 2^{17}.

For $\|\boldsymbol{AP} - \boldsymbol{PB}\|_2^2$ we use both the OpenMP (in C) and the Spark (in Python) implementations. For Spark, when running with 56 cores or less, we use a single machine out of the cluster. From Table V we see that for dense graphs OpenMP has excellent speedup; for example, we see that $t_{\texttt{IT}}$ for 30 CPUs is $30\times$ smaller than $t_{\texttt{IT}}$ for 1 CPU, matching the level of parallelism. However, the Spark implementation is slower for these dense graphs. This is due to both the high-level programming language (Python) and the Spark overheads, e.g., the cost of communicating the consensus variables across machines, which is more considerable in the dense graphs. We report speedups for running over a Spark cluster with 480 CPUs based on Gustafoson's law [79] that computes speedup as follows, $s_{speedup} = 1 - \gamma + \gamma s^{par}_{speedup}$, where $\gamma \in [0, 1]$ is the portion of the serial program that can be parallelized (ADMM iterations in Alg. 1) and $s^{par}_{speedup}$ is the speedup for the portion of the program that benefits from parallelism. We compute γ as the ratio of running time for ADMM iterations (we consider 100 iterations) to the total running time for serial execution (1 CPU), i.e., $\gamma = \frac{100\times t_{\texttt{IT}}}{100\times t_{\texttt{IT}} + t_{\texttt{SU}}}$, where $t_{\texttt{IT}}, t_{\texttt{SU}}$ correspond to 1 CPU in Table V. We compute $s^{par}_{speedup}$ by comparing $t_{\texttt{IT}}$ values for 1 CPU and 480 CPUs. The speedups for Table V are 26, 153, 84, 92, and, 110, respectively for $n = 2^{10}, 2^{11}, 2^{13}, 2^{14}$, and 2^{15}.

As expected from Lemma V.2, the Spark scales better for sparse graphs. For each n, by increasing the number of CPUs from 1 to 56, on a single machine, we see a consistent speedup

in both $t_{\texttt{IT}}$ and $t_{\texttt{SU}}$. Moreover, when running over cluster (448CPUs) it is 4.5, 5, and 6.7 times faster then a single machine (56CPUs) for $n = 2^{13}, 2^{14}$, and 2^{15}, respectively.

For $\|\boldsymbol{AP} - \boldsymbol{PB}\|_2$ we only report the results for Spark implementation in Table VI. By comparing the running times for 448 CPUs and 56 CPUs we see speedups of 2.89 and 7.69, for the dense graphs with $n = 2^{10}$ and 2^{11}, respectively, and speedups of 6.68, 7.13, and 7.41 for the sparse graphs of size $n = 2^{13}, 2^{14}$, and 2^{15}, respectively. We again observe that for sparse graphs our algorithm scales better than dense graphs: for spares graphs the running times almost consistently double as we double n. In comparison to Table V, we see that setup times are lower as for the case of $\|\boldsymbol{AP} - \boldsymbol{PB}\|_2^2$ our implementation pre-computes some matrices.

Real Graph Pairs. We use our proposed ADMM algorithm to compute distances for the real graph pairs summarized in Table II. We force a pair of graphs G_A and G_B to have the same number of nodes $n = \max(|\mathcal{V}_A|, |\mathcal{V}_B|)$ by adding isolated (degree 0) dummy nodes to the smaller graphs.

For brevity, we only report results for $p = 2$. Fig. 4 shows the trace of residuals and norm. We see that our algorithm converges for all these graph pairs; for `cortex`, `monkey`, `protein`, `retweet`, `deezer`, and `slashdot`, the parameter ϵ is $0.01, 0.005, 0.0006, 0.001, 0.3$, and 0.06, respectively. The average iteration time of Alg. 1 for these pairs are 364(s), 712(s), 3836(s), 4375(s), 3797(s), and 2290(s), respectively, scaling well with $|\mathcal{Q}|$ and $|\mathcal{I}|$.

VII. Conclusions

We present a massively parallel algorithm for graph distance computation via ADMM. We can consider penalty terms beyond the trace. Accelerating this method further, via, e.g., optimally partitioning the data, are important open problems. Our approach allows introducing additional penalty terms beyond the trace we considered here. Identifying means of accelerating this method further, as well as how to optimally partition the data, are important open problems.

VIII. Acknowledgment

The authors gratefully acknowledge the support of the National Science Foundation (grants IIS-1741197, IIS-1741129, and CCF-1750539) and the National Institutes of Health (grant 1U01AI124302).

Appendix A
Proof of Lemma IV.1

For $(i,j) \in [n] \times [n]$ the (i,j)-th element of $\boldsymbol{AP} - \boldsymbol{PB}$ is:

$$(\boldsymbol{AP}-\boldsymbol{PB})_{ij} = \sum_{k:(k,j)\in\mathcal{S}_L^{(i,j)}} a_{ik}p_{kj} - \sum_{k:(i,k)\in\mathcal{S}_R^{(i,j)}} p_{ik}b_{kj},$$

where $\mathcal{S}_L^{(i,j)} = \{(k,j) \in \mathcal{Q} | (i,k) \in \mathcal{E}_A\}$ and $\mathcal{S}_R^{(i,j)} = \{(i,k) \in \mathcal{Q} | (k,j) \in \mathcal{E}_B\}$. We can write $(\boldsymbol{AP} - \boldsymbol{PB})_{ij}$ as $f_{ij}(\boldsymbol{P}_{\mathcal{S}_{ij}})$, where $f_{ij}(\boldsymbol{P}_{\mathcal{S}_{ij}}) \triangleq \sum_{k:(k,j)\in\mathcal{S}_L^{(i,j)}} a_{ik}p_{kj} - \sum_{k:(i,k)\in\mathcal{S}_R^{(i,j)}} p_{ik}b_{kj}$, where $\mathcal{S}_{ij} = \mathcal{S}_L^{(i,j)} \cup \mathcal{S}_R^{(i,j)}$ and $f_{ij} : \mathbb{R}^{|\mathcal{S}_{ij}|} \to \mathbb{R}$ is a linear function. The entry-wise p norm of $AP - PB$ is thus (8a), where $\mathcal{I}$ comprises pairs (i,j) for which $\mathcal{S}_{ij} \neq \emptyset$. On the other hand, $\mathsf{tr}(\boldsymbol{P}^\top \boldsymbol{D}_{\boldsymbol{A},\boldsymbol{B}}) = \sum_{(i,j)\in\mathcal{Q}} p_{ij}d_{ij}$ by the fact that $p_{ij} = 0$ for $(i,j) \notin \mathcal{Q}$. The function $\chi_{\mathcal{R}}(\boldsymbol{P})$ states that each row i of $\boldsymbol{P}$ belongs to the set: $\mathcal{R}^{(i)} = \{\boldsymbol{p}^{(i)} \in [0,1]^{|\mathcal{S}_i|} | \mathbf{1}^\top \boldsymbol{p}^{(i)} = 1\}$, where $\mathcal{S}_i = (\{i\} \times \mathcal{V}_B) \cap \mathcal{Q}$. As a result, we can write $\chi_{\mathcal{R}}(P)$ as the sum of the corresponding characteristic functions. Similarly, $\chi_{\mathcal{C}}(P)$ can be written as the sum of the characteristic functions for the sets $\mathcal{C}^{(j)} = \{\boldsymbol{p}^{(j)} \in [0,1]^{|\mathcal{S}_j|} | \mathbf{1}^\top \boldsymbol{p}^{(j)} = 1\}$, corresponding to each column $j \in [n]$ of P. □

Appendix B
Proofs of Theorems IV.2 and IV.3.

We present here the proofs of Theorems IV.2 and IV.3. We first give high-level proofs as derived from key lemmas, and then prove these lemmas.

A. Proof of Theorem IV.2

The objective of Prob. (17) is strongly convex. Hence, the optimal solution $\hat{\boldsymbol{u}} \in \mathbb{R}_+^d$ is unique. We first show that $\hat{\boldsymbol{u}}$ lies, coordinate-wise, between $\mathbf{0}$ and $\hat{\boldsymbol{w}} \in \mathbb{R}_+^d$:

Lemma B.1. *Let $\hat{\boldsymbol{u}}$ be the optimal solution of Prob.* (17)*. Then $\hat{u}_i \in [0, \hat{w}_i]$, for all $i \in [d]$.*

The proof can be found in Appendix B-D. The lemma implies that we only need to look for a solution in a bounded domain. Note also that, as an immediate implication of Lemma B.1, if $\hat{w}_i = 0$, then necessarily also $\hat{u}_i = 0$.

The optimal solution $\hat{\boldsymbol{u}}$ satisfies the KKT conditions:

$$\mathbf{0} \in \{\boldsymbol{g} + \boldsymbol{u} - \hat{\boldsymbol{w}} - \boldsymbol{\alpha} | \boldsymbol{g} \in \partial f_p(\boldsymbol{u})\}, \tag{27a}$$

$$\alpha_i u_i = 0 \quad \text{for all } i \in [d], \quad \boldsymbol{u} \geq 0, \quad \boldsymbol{\alpha} \geq 0, \tag{27b}$$

where $\partial f_p(\boldsymbol{u})$ is the subdifferential[7] of the function $f_p(\boldsymbol{u}) = \|\boldsymbol{u}\|_p$ at the point $\boldsymbol{u}$. Eq. (27a) implies a condition on $\hat{\boldsymbol{w}}$ under which the optimal solution to (17) is the zero vector:

Lemma B.2. *Given $p \geq 1$, let $q \geq 1$ be such that $\frac{1}{p} + \frac{1}{q} = 1$. If $\hat{\boldsymbol{w}} \in \mathbb{R}_+^d$ satisfies $\|\hat{\boldsymbol{w}}\|_q \leq 1$, the optimal solution to Prob.* (17) *is $\hat{\boldsymbol{u}} = \mathbf{0}$.*

The proof can be found in Appendix B-E. Intuitively, we prove this by verifying that if $\|\hat{\boldsymbol{w}}\|_q \leq 1$ then $\boldsymbol{u} = \boldsymbol{\alpha} = \mathbf{0}$ satisfy the KKT conditions in Eq. (27). Lemma B.2 therefore immediately implies the first ("easy") case of Theorem IV.2.

We therefore turn to the case where $\|\hat{\boldsymbol{w}}\|_q > 1$ (the "hard" case). For $\boldsymbol{u} \neq 0$, f_p is differentiable and its subdifferential is a singleton, i.e., $\partial f_p(\boldsymbol{u}) = \{\nabla f_p(\boldsymbol{u})\}$, where

$$\partial f_p(\boldsymbol{u})/\partial u_i = \left(\frac{u_i}{\|\boldsymbol{u}\|_p}\right)^{p-1},$$

for $i \in [d]$. Suppose that the i-th element of the optimal point $\hat{\boldsymbol{u}}$ is positive, i.e., $\hat{u}_i > 0$. Then, $\alpha_i = 0$ by Eq. (27b), and Eq. (27a) implies that $\hat{\boldsymbol{u}}$ satisfies the following equation:

$$\left(\hat{u}_i/\|\hat{\boldsymbol{u}}\|\right)^{p-1} + (\hat{u}_i - \hat{w}_i) = 0, \text{ for all } i \text{ s.t. } \hat{u}_i > 0. \tag{28}$$

The optimality condition (28) can equivalently be written as

$$\hat{u}_i = \hat{w}_i g\left(\|\hat{\boldsymbol{u}}\|_p (\hat{w}_i)^{\frac{2-p}{p-1}}\right), \quad \text{for all } i \text{ s.t. } \hat{u}_i > 0. \tag{29}$$

[7]Note that f_p is not differentiable at $\boldsymbol{u} = \mathbf{0}$, hence the need to refer to its subdifferential.

where $g : \mathbb{R}_+ \to \mathbb{R}_+$ is defined via Eq. (19). Recall that Lemma B.1 implies that, if $i \notin \text{supp}(\hat{\boldsymbol{w}})$, then necessarily $\hat{u}_i = 0$. We can therefore consider w.l.o.g. a vector $\hat{\boldsymbol{w}}$ for which $\text{supp}(\hat{\boldsymbol{w}}) = [d]$; if not, we can compute the optimal solution by setting $\hat{u}_i = 0$ for $i \notin \text{supp}(\hat{\boldsymbol{w}})$, and focus on what happens on the remainder of the coordinates, that have full support. Not surprisingly, the optimal solution in this case is characterized by Eq. (29). In particular, the following lemma holds:

Lemma B.3. *Consider a $\boldsymbol{w} \in \mathbb{R}^d_+$ s.t. (a) $\text{supp}(\hat{\boldsymbol{w}}) = [d]$, and (b) $\|\boldsymbol{w}\|_q > 1$, where $\frac{1}{p} + \frac{1}{q} = 1$. Let $g_i : \mathbb{R} \to \mathbb{R}^d$, $i \in [d]$, and $h : \mathbb{R}_+ \to \mathbb{R}$ be given by Eqs.* (20) *and* (21)*, respectively. Then, the unique solution $\hat{\boldsymbol{u}}$ to Prob.* (17) *is given by $\hat{u}_i = g_i(s^*)$, for all $i \in [d]$, where s^* is the unique value in $(0, \|\boldsymbol{w}\|_p]$ s.t. $h(s^*) = 0$.*

The proof can be found in Appendix B-F. Intuitively, we show this by first establishing that $h(0) = 0$ and $h(\|\hat{\boldsymbol{w}}\|_p) \leq 0$. We show that, if $\|w\|_q > 1$, then $h'(0) > 0$; thus, h must be strictly positive in the neighborhood of 0. As $h(\|\hat{\boldsymbol{w}}\|_p) \leq 0$, there must exist a root of h in $(0, \|\hat{\boldsymbol{w}}\|_p]$; any such root corresponds to an optimum of Prob. (17) via (29); uniqueness is implied by strong convexity.

Lemma B.3, along with our observation on cases where $i \notin \text{supp}(\hat{\boldsymbol{w}})$, immediately imply Theorem IV.2. To see this, note first that for $i \notin \text{supp}(\hat{\boldsymbol{w}})$, $g_i(s) = 0$ for all $s \in \mathbb{R}_+$. Moreover, these 0 coordinates do not contribute to $\|\boldsymbol{u}\|_p$ or $\|\hat{\boldsymbol{w}}\|_p$; as such, they do not affect h and, thereby, the root s^*: the latter is fully determined by only elements in $\text{supp}(\hat{\boldsymbol{w}})$. Hence, Eq. (22) holds for all coordinates in $[d]$. □

B. Proof of Theorem IV.3

We begin by showing the equivalence of Problems (15) and (17):

Lemma B.4. *Let $\hat{\boldsymbol{u}}$ be an optimal solution to Prob.* (17)*, where $\hat{\boldsymbol{w}}$ is given by Eq.* (16)*, then $\boldsymbol{u}^*$, given by Eq.* (18)*, is an optimal solution to Prob.* (15)*.*

The proof can be found in Appendix B-C. Armed with this result, we next show that Alg. 3 correctly bounds the root s^*, whenever the corresponding for-loop is executed:

Lemma B.5. *If $\|\hat{\boldsymbol{w}}\|_q > 1$, at every iteration of Alg. 3, $s^* \in [s_L, s_U]$.*

The proof is in Appendix B-G. Observe that the distance between the two bounds is halved at each iteration. Hence, at the last $(\log_2 \lceil \frac{1}{\varepsilon} \rceil)$ iteration,

$$|s - s^*| \leq |s_L - s_U| \leq \|\hat{\boldsymbol{w}}\|_p \varepsilon. \tag{30}$$

On the other hand, functions $g_i : \mathbb{R}_+ \to \mathbb{R}_+$ are Lipschitz:

Lemma B.6. *Each function $g_i : \mathbb{R}_+ \to \mathbb{R}_+$ is Lipschitz continuous with Lipschitz parameter $\hat{w}_i^{\frac{1}{p-1}}$.*

The proof is in Appendix B-H. Lemma B.6 immediately implies that, for $\boldsymbol{u}$ the output of the algorithm, and s the last estimate of the root: $\left(\sum_{i\in[d]}(u_i - \hat{u}_i)^p\right)^{1/p} \leq \left(\sum_{i\in[d]} \hat{w}_i^{\frac{p}{p-1}}\right)^{1/p} \cdot |s - s^*|$, and the theorem follows from Eq. (30), as $q = \frac{p}{p-1}$. □

C. Proof of Lemma B.4

We have that: $\min_{\boldsymbol{u}\in\mathbb{R}^d} \rho(\|\boldsymbol{u}\|_p + \frac{\rho}{2}\|\boldsymbol{u} - \boldsymbol{w}\|_2^2) = \min_{\boldsymbol{u}'\in\mathbb{R}^d} \|\boldsymbol{u}'\|_p + \frac{1}{2}\|\boldsymbol{u}' - \boldsymbol{w}'\|_2^2$ for $\boldsymbol{u}' = \rho\boldsymbol{u}$, $\boldsymbol{w}' = \rho\boldsymbol{w}$. One can show that the coordinates of the optimal solution to the latter problem will have the same sign as the coordinates of $\boldsymbol{w}$. Let $\odot : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ indicate the element-wise multiplication between two vectors, and $\text{sign} : \mathbb{R}^d \to \{-1, +1\}^d$ be the vector resulting from element-wise application of the sign operator. Then, under the transformation $\boldsymbol{v} = \text{sign}(\boldsymbol{w}) \odot \boldsymbol{u}' \in \mathbb{R}^d_+$, Prob. (15) is equivalent to: $\min_{\boldsymbol{v}\in\mathbb{R}^d_+} \|\text{sign}(w)\odot\boldsymbol{v}\|_p + \frac{1}{2}\left\|\text{sign}(\boldsymbol{w})\odot\boldsymbol{v} - \text{sign}(\boldsymbol{w})\odot\hat{\boldsymbol{w}}\right\|_2^2 = \min_{\boldsymbol{v}\in\mathbb{R}^d_+} \|\boldsymbol{v}\|_p + \frac{1}{2}\|\boldsymbol{v} - \hat{\boldsymbol{w}}\|_2^2$,as $\|e \odot y\|_p = \|y\|_p$ for all $e \in \{-1, 1\}^d$, $y \in \mathbb{R}^d$, and $p \geq 1$. Hence, given an optimal solution $\hat{\boldsymbol{u}}$ to Prob. (17), the optimal solution to Prob. (15) is given by $\boldsymbol{u}^* = \frac{1}{\rho}(\text{sign}(\boldsymbol{w}) \odot \hat{\boldsymbol{u}})$. □

D. Proof of Lemma B.1

Suppose that $\hat{\boldsymbol{u}}$ is an optimal solution of (17), s.t., $\hat{u}_i < 0$ for some $i \in [d]$. Then, for the vector $\hat{\boldsymbol{u}}'$ with all elements equal to the elements $\hat{\boldsymbol{u}}$ except the i-th element with $\hat{u}'_i = 0$ we have the following $\|\hat{\boldsymbol{u}}'\|_p + \frac{1}{2}\|\hat{\boldsymbol{u}}' - \hat{\boldsymbol{w}}\|_2^2 < \|\hat{\boldsymbol{u}}\|_p + \frac{1}{2}\|\hat{\boldsymbol{u}} - \hat{\boldsymbol{w}}\|_2^2$, a contradiction. Similarly, if $\hat{u}_i > \hat{\boldsymbol{w}}_i$ for some $i \in [d]$, we can construct a vector $\hat{\boldsymbol{u}}'$, s.t., all of its elements are the same with the elements of $\hat{\boldsymbol{u}}$, except $\hat{u}'_i = \hat{\boldsymbol{w}}_i$, then again $\|\hat{\boldsymbol{u}}'\|_p + \frac{1}{2}\|\hat{\boldsymbol{u}}' - \hat{\boldsymbol{w}}\|_2^2 < \|\hat{\boldsymbol{u}}\|_p + \frac{1}{2}\|\hat{\boldsymbol{u}} - \hat{\boldsymbol{w}}\|_2^2$, a contradiction. □

E. Proof of Lemma B.2

We show that $\boldsymbol{u} = \boldsymbol{0}, \boldsymbol{\alpha} = \boldsymbol{0}$ satisfy the KKT conditions (27). For $\boldsymbol{u} = \boldsymbol{\alpha} = 0$ Eq. (27b) is obviously satisfied. Then we need to show that $\boldsymbol{0} \in \{\boldsymbol{g} - \hat{\boldsymbol{w}} \mid \boldsymbol{g} \in \partial f_p(\boldsymbol{0})\}$, or equivalently, $\hat{\boldsymbol{w}} \in \partial f_p(\boldsymbol{0})$. Formally, the subdifferential at zero is the set $\partial f_p(\boldsymbol{0}) = \{\boldsymbol{g} \in \mathbb{R}^d \mid \boldsymbol{g}^\top \boldsymbol{v} \leq \|\boldsymbol{v}\|_p \text{ for all } \boldsymbol{v} \in \mathbb{R}^d\}$. By Holder's inequality, for every $\boldsymbol{v} \in \mathbb{R}^d$ we have $\hat{\boldsymbol{w}}^\top \boldsymbol{v} \leq \sum_{i=1}^d |\hat{w}_i||v_i| \leq \|\hat{\boldsymbol{w}}\|_q \|\boldsymbol{v}\|_p \leq \|\boldsymbol{v}\|_p$, where the last inequality holds as $\|\hat{\boldsymbol{w}}\|_q \leq 1$. Hence, $\hat{\boldsymbol{w}} \in \partial f_p(\boldsymbol{0})$, and $\boldsymbol{u} = \boldsymbol{\alpha} = \boldsymbol{0}$ satisfy the KKT conditions, so $\hat{\boldsymbol{u}} = \boldsymbol{0}$ is optimal. □

F. Proof of Lemma B.3

The inverse g^{-1} of function g is given by

$$g^{-1}(x) = x(1 - x)^{-1/(p-1)}. \tag{31}$$

As g^{-1} is monotone and continuous in $[0, 1)$, we have that g is also monotone and continuous in $\mathbb{R}_+$. Hence, function h is continuous in the interval $[0, \|\hat{\boldsymbol{w}}\|_p]$. Given that, by the intermediate value theorem, $g(a) \in [0, 1]$, we have that $h(\|\hat{\boldsymbol{w}}\|_p) \leq 0$. Since, by definition, $g(0) = 0$, we also have that $h(0) = 0$. Moreover,

$$\frac{dh(0)}{ds} = \lim_{\delta\to 0} \frac{\left(\sum_{i=1}^d (g_i(\delta))^p\right)^{\frac{1}{p}} - \delta}{\delta} = \lim_{\delta\to 0} \left(\frac{\sum_{i=1}^d (g_i(\delta))^p}{\delta^p}\right)^{\frac{1}{p}} - 1.$$

By Taylor's theorem, the first-degree Taylor approximation of g_i at 0 is $g_i(\delta) = g'_i(0)\delta + o(\delta^2)$. The partial derivative

$g_i'(0)$ is given by: $g_i'(0) = \hat{w}_i^{\frac{1}{p-1}} g'(0)$. Note that $g'(0) = \left(\frac{dg^{-1}(0)}{dx}\right)^{-1} = 1$, and, as a result,

$$g_i'(0) = \hat{w}_i^{\frac{1}{p-1}}. \tag{32}$$

Therefore, we have:

$$\frac{\partial h(0)}{\partial s} = \lim_{\delta \to 0} \left(\sum_{i=1}^{d} (\hat{w}_i^{\frac{1}{p-1}} \delta + O(\delta^2))^p / \delta^p\right)^{\frac{1}{p}} - 1 = \|\hat{\boldsymbol{w}}\|_q^{\frac{1}{p-1}} - 1 > 0,$$

as $\|\hat{\boldsymbol{w}}\|_q > 1$ for $q = \frac{p}{p-1}$ by the hypothesis of the theorem. Hence, $h(s)$ is positive in a neighborhood of 0. As h is continuous, and $h(\|\boldsymbol{w}\|_p) \leq 0$, by the intermediate value theorem h must contain an $s^* \in (0, \|\hat{\boldsymbol{w}}\|_p]$ s.t, $h(s^*) = 0$. Such a solution satisfies Eq. (29) for all $i \in [d]$ and, as such, it satisfies the KKT conditions of Prob. (17); therefore, it is an optimal solution. Strong convexity implies its uniqueness. □

G. Proof of Lemma B.5.

By Lemma B.3, there must exist a unique root of $h(s) = 0$ in $(0, \|\hat{\boldsymbol{w}}\|_p]$. This, along with the strict positivity of h in the vicinity of 0 when $\|\hat{\boldsymbol{w}}\|_q > 1$, implies that if $h(\|\hat{\boldsymbol{w}}\|_p) = 0$, all values $h(s)$ for $s \in (0, \|\hat{\boldsymbol{w}}\|_p)$ are strictly positive, and the bisection will repeatedly update the lower bound but never the upper bound. The lemma therefore holds. If, on the other hand $h(\|\hat{\boldsymbol{w}}\|_p]) < 0$, s^* must be in $(0, \|\hat{\boldsymbol{w}}\|_p)$ by Lemma B.3, so the lemma holds for the first iteration. We can show, by induction on iterations, that $h(s_L) \geq 0$, with equality holding only if $s_L = 0$, and $h(s_U) < 0$. If $h(s_L) > 0$, the lemma follows from the intermediate value theorem. If $h(s_L) = 0$, the lemma again follows from the intermediate value theorem, and the fact that h is strictly positive in the vicinity of 0. □

H. Proof of Lemma B.6

The inverse of g is given by Eq. (31), which is strictly increasing, differentiable, and convex in $[0, 1)$; the latter follows from the fact that the second derivative is non-negative for $p > 1$, $x \in [0, 1)$. Hence, g is strictly increasing, differentiable, and concave. Each function g_i, $i \in [d]$, consists of a composition of g with an affine function, and a multiplication with a non-negative scalar, so it is also concave. Hence, it is Lipschitz continuous with parameter given by $g_i'(0)$; the latter is characterized by Eq. (32). □

APPENDIX C
PROOFS OF PARALLEL COMPLEXITY RESULTS

A. Proof of Lemma V.1

The number of non-zero elements in the matrix $\boldsymbol{AP} - \boldsymbol{PB}$ is at most the number of the non-zero elements in $\boldsymbol{AP} + \boldsymbol{PB}$, where $\boldsymbol{P}$ is a matrix has the full support under constraints $\mathcal{G}$, e.g., $\boldsymbol{P} = \boldsymbol{E}$. Each set $\mathcal{S}_{ij}$ defines the support of the (i, j)-th element of $\boldsymbol{AP} - \boldsymbol{PB}$; therefore, we conclude that the total number of non-empty sets $\mathcal{S}_{ij}$ is upper-bounded by m_0. For the terms (8c) and (8d), it is easy to see that we have $|\mathcal{V}_A| = n$ sets $\mathcal{S}_i$ and $|\mathcal{V}_B| = n$ sets $\mathcal{S}_j$, each corresponding to the nodes $i \in \mathcal{V}_A$ and $j \in \mathcal{V}_B$, respectively. □

B. Proof of Lemma V.2

Let $\mathcal{S}_L^{(i,j)} = \{(k, j) \in \mathcal{Q}\} \cap \{(i, k) \in \mathcal{E}_A\}$ and $\mathcal{S}_R^{(i,j)} = \{(i, k) \in \mathcal{Q}\} \cap \{(k, j) \in \mathcal{E}_B\}$. Then $\sum_{i,j \in [n]} |\mathcal{S}_L^{(i,j)}| = \sum_{j \in [n]} \sum_{i \in [n]} |\mathcal{S}_L^{(i,j)}| \leq \sum_{j \in [n]} \min(|\mathcal{E}_A|, nd_j) \leq \min(n|\mathcal{E}_A|, \sum_{j \in [n]} nd_j) = \min(n|\mathcal{E}_A|, n|\mathcal{Q}|)$, where d_j is the node degree for $j \in \mathcal{V}_B$.

Similarly, we can show that $\sum_{i,j \in [n]} |\mathcal{S}_R^{(i,j)}| \leq \min(n|\mathcal{E}_B|, n|\mathcal{Q}|)$. As a result, $\sum_{i,j \in [n]} |\mathcal{S}_{ij}| \leq \sum_{i,j \in [n]} |\mathcal{S}_L^{(i,j)}| + |\mathcal{S}_R^{(i,j)}| \leq \min(n|\mathcal{E}_A|, n|\mathcal{Q}|) + \min(n|\mathcal{E}_B|, n|\mathcal{Q}|)$.

Now we prove (25c). For each set $\mathcal{S}_{ij}$ we have that: $|\mathcal{S}_{ij}| \leq |\mathcal{S}_L^{(i,j)}| + |\mathcal{S}_R^{(i,j)}| \leq \max(d_i, d_{\mathcal{Q}}) + \max(d_j, d_{\mathcal{Q}}) \leq \max(d_A, d_{\mathcal{Q}}) + \max(d_B, d_{\mathcal{Q}})$. From this and Lemma V.1, which shows that $\mathcal{I} \leq m$, we get $\sum_{i,j \in [n]} |\mathcal{S}_{ij}| \leq m \max_{i,j}(|\mathcal{S}_{ij}|) \leq m(\max(d_A, d_{\mathcal{Q}}) + \max(d_B, d_{\mathcal{Q}}))$. For $\mathcal{S}_i, i \in [n]$ we have that: $\bigcap_i \mathcal{S}_i = \emptyset$, and $\bigcup_i \mathcal{S}_i = \{(k, j) \in \mathcal{Q} | j \in \mathcal{V}_B\} \subseteq \mathcal{Q}$. Therefore, for the total size we have: $\sum_i |\mathcal{S}_i| = |\cup_i \mathcal{S}_i| \leq |\mathcal{Q}|$. □

REFERENCES

[1] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 18, no. 03, pp. 265–298, 2004.

[2] F. H. Allen, "The Cambridge Structural Database: a quarter of a million crystal structures and rising," *Acta Crystallographica Section B: Structural Science*, vol. 58, no. 3, pp. 380–388, 2002.

[3] V. Kvasnička, J. Pospíchal, and V. Baláž, "Reaction and chemical distances and reaction graphs," *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta)*, vol. 79, no. 1, pp. 65–79, 1991.

[4] O. Macindoe and W. Richards, "Graph comparison using fine structure analysis," in *SocialCom*, 2010.

[5] D. Koutra, J. T. Vogelstein, and C. Faloutsos, "Deltacon: A principled massive-graph similarity function," in *SDM*, 2013.

[6] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Network similarity via multiple social theories," in *ASONAM*, 2013.

[7] D. Koutra, N. Shah, J. T. Vogelstein, B. Gallagher, and C. Faloutsos, "Deltacon: Principled massive-graph similarity function with attribution," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 3, pp. 28:1–28:43, 2016.

[8] C. Chen, X. Yan, F. Zhu, and J. Han, "gApprox: Mining frequent approximate patterns from a massive network," in *ICDM*, 2007.

[9] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, "Fast best-effort pattern matching in large attributed graphs," in *KDD*, 2007.

[10] F. Falchi, C. Gennaro, and P. Zezula, "A content–addressable network for similarity search in metric spaces," in *DBISP2P*, 2006.

[11] S. B. Roy, T. Eliassi-Rad, and S. Papadimitriou, "Fast best-effort search on graphs with multiple attributes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 755–768, 2015.

[12] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li, "Rolx: structural role extraction & mining in large graphs," in *KDD*, 2012.

[13] J. Bento and S. Ioannidis, "A family of tractable graph metrics," *Applied Network Science*, vol. 4, no. 1, p. 107, 2019.

[14] K. L. Clarkson, "Nearest-neighbor searching and metric space dimensions," *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pp. 15–59, 2006.

[15] ——, "Nearest neighbor queries in metric spaces," *Discrete & Computational Geometry*, vol. 22, no. 1, pp. 63–93, 1999.

[16] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *ICML*, 2006.

[17] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *ECML PKDD*, 2002.

[18] M. R. Ackermann, J. Blömer, and C. Sohler, "Clustering for metric and nonmetric distance measures," *ACM Transactions on Algorithms (TALG)*, vol. 6, no. 4, p. 59, 2010.

[19] R. R. Mettu and C. G. Plaxton, "Optimal time bounds for approximate clustering," *Machine Learning*, vol. 56, no. 1-3, pp. 35–60, 2004.

[20] Y. Bartal, M. Charikar, and D. Raz, "Approximating min-sum k-clustering in metric spaces," in *STOC*, 2001.

[21] P. Indyk, "Sublinear time algorithms for metric space problems," in *STOC*, 1999.

[22] G. Chartrand, G. Kubicki, and M. Schultz, "Graph similarity and distance in graphs," *Aequationes Mathematicae*, vol. 55, no. 1-2, pp. 129–145, 1998.

[23] L. Babai, "Graph isomorphism in quasipolynomial time," in *STOC*, 2016.

[24] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[25] L. Dagum and R. Menon, "OpenMP: an Industry-Standard API for Shared-Memory Programming," *Computing in Science & Engineering*, no. 1, pp. 46–55, 1998.

[26] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets." in *HotCloud*, 2010.

[27] P. Papadimitriou, A. Dasdan, and H. Garcia-Molina, "Web Graph Similarity for Anomaly Detection," *Internet Services and Applications*, vol. 1, no. 1, pp. 19–30, 2010.

[28] S. Soundarajan, T. Eliassi-Rad, and B. Gallagher, "A guide to selecting a network similarity method," in *SDM*, 2014.

[29] M. R. Garey and D. S. Johnson, *Computers and Intractability*. WH Freeman New York, 2002.

[30] A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, and H. Bunke, "Approximation of Graph Edit Distance Based on Hausdorff Matching," *Pattern Recognition*, vol. 48, no. 2, pp. 331–343, 2015.

[31] H. Bunke and K. Shearer, "A Graph Distance Metric Based on the Maximal Common Subgraph," *Pattern Recognition Letters*, vol. 19, no. 3, pp. 255–259, 1998.

[32] H. Bunke, "On a Relation Between Graph Edit Distance and Maximum Common Subgraph," *Pattern Recognition Letters*, vol. 18, no. 8, pp. 689–694, 1997.

[33] J. Koca, M. Kratochvil, V. Kvasnicka, L. Matyska, and J. Pospichal, *Synthon Model of Organic Chemistry and Synthesis Design*. Springer Science & Business Media, 2012, vol. 51.

[34] B. J. Jain, "On the Geometry of Graph Spaces," *Discrete Applied Mathematics*, vol. 214, pp. 126–144, 2016.

[35] K. Riesen and H. Bunke, "Approximate Graph Edit Distance Computation by Means of Bipartite Graph Matching," *Image and Vision Computing*, vol. 27, no. 7, pp. 950–959, 2009.

[36] S. Fankhauser, K. Riesen, and H. Bunke, "Speeding Up Graph Edit Distance Computation Through Fast Bipartite Matching," in *GBR*, 2011.

[37] K. Riesen, M. Neuhaus, and H. Bunke, "Graph embedding in vector spaces by means of prototype selection," in *GBR*, 2007.

[38] K. Riesen and H. Bunke, *Graph Classification and Clustering Based on Vector Space Embedding*. World Scientific, 2010, vol. 77.

[39] M. Ferrer, E. Valveny, F. Serratosa, K. Riesen, and H. Bunke, "Generalized Median Graph Computation by Means of Graph Embedding in Vector Spaces," *Pattern Recognition*, vol. 43, no. 4, pp. 1642–1655, 2010.

[40] P. Zhu and R. C. Wilson, "A Study of Graph Spectra for Comparing Graphs," in *BMVC*, 2005.

[41] R. C. Wilson and P. Zhu, "A Study of Graph Spectra for Comparing Graphs and Trees," *Pattern Recognition*, vol. 41, no. 9, pp. 2833–2841, 2008.

[42] H. Elghawalby and E. R. Hancock, "Measuring Graph Similarity Using Spectral Geometry," in *ICIAR*, 2008.

[43] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance Metric Learning with Application to Clustering with Side-Information," in *NIPS*, 2002.

[44] J. A. Hartigan and J. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.

[45] S. Gold and A. Rangarajan, "A Graduated Assignment Algorithm for Graph Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 4, pp. 377–388, 1996.

[46] L. Wiskott, J.-M. Fellous, N. Krüger, and C. Von Der Malsburg, "Face Recognition by Elastic Bunch Graph Matching," in *CAIP*. Springer, 1997.

[47] G. W. Klau, "A New Graph-Based Method for Pairwise Global Network Alignment," *BMC Bioinformatics*, vol. 10, no. 1, p. S59, 2009.

[48] T. Goldstein, G. Taylor, K. Barabin, and K. Sayre, "Unwrapping ADMM: Efficient Distributed Computing via Transpose Reduction," in *AISTATS*, 2016.

[49] M. El-Kebir, J. Heringa, and G. W. Klau, "Natalie 2.0: Sparse global network alignment as a special case of quadratic assignment," *Algorithms*, 2015.

[50] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang, "Algorithms for Large, Sparse Network Alignment Problems," in *ICDM*, 2009.

[51] R. Singh, J. Xu, and B. Berger, "Pairwise Global Alignment of Protein Interaction Networks by Matching Neighborhood Topology," in *RECOMB*, 2007.

[52] V. Lyzinski, D. E. Fishkind, M. Fiori, J. T. Vogelstein, C. E. Priebe, and G. Sapiro, "Graph Matching: Relax at Your Own Risk," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 60–73, 2016.

[53] C. Schellewald and C. Schnörr, "Probabilistic Subgraph Matching Based on Convex Relaxation," in *CVPR*, 2005.

[54] J. Liu and J. Ye, "Efficient l1/lq NormRregularization," *arXiv preprint arXiv:1009.4766*, 2010.

[55] S. Sra, "Fast Projections Onto l1, q-norm Balls for Grouped Feature Selection," in *ECML PKDD*, 2011.

[56] ——, "Fast projections onto mixed-norm balls with applications," *Data Mining and Knowledge Discovery*, vol. 25, no. 2, pp. 358–377, 2012.

[57] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in Optimization*, 2014.

[58] J. J. Moreau, "Décomposition orthogonale d'un espace hilbertien selon deux cônes mutuellement polaires," 1962.

[59] S. R. Becker, E. J. Candès, and M. C. Grant, "Templates for Convex Cone Problems with Applications to Sparse Signal Recovery," *Mathematical Programming Computation*, vol. 3, no. 3, p. 165, 2011.

[60] D. Gabay and B. Mercier, "A Dual Algorithm for the Ssolution of Non Linear Variational Problems via Finite Element Approximation," *Computers & Mathematics with Applications*, vol. 2, no. 1, pp. 17–40, 1976.

[61] C. Song, S. Yoon, and V. Pavlovic, "Fast admm Algorithm for Distributed Optimization with Adaptive Penalty," in *AAAI*, 2016.

[62] T.-H. Chang, M. Hong, W.-C. Liao, and X. Wang, "Asynchronous Distributed Alternating Direction Method of Multipliers: lgorithm and Convergence Analysis," in *ICASSP*, 2016.

[63] E. Wei and A. Ozdaglar, "Distributed Alternating Direction Method of Multipliers," in *CDC*, 2012.

[64] R. Zhang and J. Kwok, "Asynchronous Distributed ADMM for Consensus Optimization," in *ICML*, 2014.

[65] Z. Xu, G. Taylor, H. Li, M. A. Figueiredo, X. Yuan, and T. Goldstein, "Adaptive Consensus ADMM for Distributed Optimization," in *ICML*, 2017.

[66] L. Majzoobi and F. Lahouti, "Analysis of distributed ADMM Algorithm for Consensus Optimization in Presence of Error," in *ICASSP*, 2016.

[67] G. França and J. Bento, "An Explicit Rate Bound for Over-Relaxed ADMM," in *ISIT*, 2016.

[68] T. Chang, M. Hong, and X. Wang, "Multi-Agent Distributed Optimization via Inexact Consensus ADMM," *IEEE Transactions on Signal Processing*, vol. 63, no. 2, pp. 482–497, 2015.

[69] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge university press, 2004.

[70] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999, previous number = SIDL-WP-1999-0120.

[71] A. Grover and J. Leskovec, "node2vec: Scalable Feature Learning for Networks," in *KDD*, 2016.

[72] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *NIPS*, 2017.

[73] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[74] B. Weisfeiler and A. Lehman, "A Reduction of a Graph to a Canonical form and an Algebra Arising During this Reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[75] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Trees Hypercubes*. Elsevier, 2014.

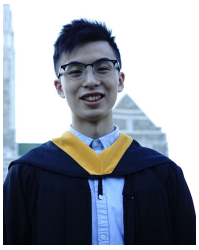
[76] M. Hong and Z.-Q. Luo, "On the Linear Convergence of the Alternating Direction Method of Multipliers," *Mathematical Programming*, vol. 162, no. 1-2, pp. 165–199, 2017.

[77] R. J. Tibshirani, J. Taylor *et al.*, "The Solution Path of the Generalized LASSO," *The Annals of Statistics*, vol. 39, no. 3, pp. 1335–1371, 2011.

[78] C. Michelot, "A Finite Algorithm for Finding the Projection of a Point Onto the Canonical Simplex of n," *Journal of Optimization Theory and Applications*, vol. 50, no. 1, pp. 195–200, 1986.
[79] J. L. Gustafson, "Reevaluating Amdahl's Law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

Armin Moharrer received a B.Sc. degree in Electrical Engineering from Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran, in 2015 and his M.Sc. degree in Electrical and Computer Engineering from Northeastern University, Boston, MA, in 2018. Since Jan 2016 he has been a Ph.D. student in the Electrical and Computer Engineering at Northeastern University under supervision of Prof. Stratis Ioannidis. His research focuses on distributed algorithms and data mining.

Jasmin Gao is finishing her Bachelor of Science and Engineering degree in the Operations Research and Financial Engineering Department at Princeton University. She is obtaining minors in Computer Science, Finance, and Cognitive Science. Jasmin holds interests in Graph Matching, Machine Learning, and Optimization, currently combining those interests in a thesis research role on optimization of financial asset allocation using machine learning.



Shikun (Jason) Wang received a Bachelor of Science in Computer Science from Boston College. During his senior year at Boston College, he worked with Professor Bento on topics related to distributed computing and graph distance. After graduation in 2019, he works as a Software Engineer at Audible on integrating Audible books and podcasts experience with Alexa smart home system.

José Bento completed his Ph.D. in Electrical Engineering at Stanford University where he worked with Professor Andrea Montanari on statistical inference and structural learning of graphical models. After his Ph.D., he moved to Disney Research, Boston lab, where he worked with Dr. Jonathan Yedidia on algorithms for distributed optimization, robotics, and computer vision. He is now with the Computer Science department at Boston College. His current research lies at the intersection of distributed algorithms and machine learning. In 2014 he received a Disney Inventor Award for his work on distributed optimization, which lead three granted patents. In 2016 he was awarded a 10M$ NIH joint grant to study the emergence of antibiotic resistance and in 2017 a 2M$ NSF joint grant to study measures of distance between large graphs. He is currently an Amazon Scholar, working with Amazon Robotics.

Stratis Ioannidis is an Associate Professor in the Electrical and Computer Engineering department at Northeastern University, in Boston, MA, where he also holds a courtesy appointment with the Khoury College of Computer Sciences. He received his B.Sc. (2002) in Electrical and Computer Engineering from the National Technical University of Athens, Greece, and his M.Sc. (2004) and Ph.D. (2009) in Computer Science from the University of Toronto, Canada. Prior to joining Northeastern, he was a research scientist at the Technicolor research centers in Paris, France, and Palo Alto, CA, as well as at Yahoo Labs in Sunnyvale, CA. He is the recipient of an NSF CAREER award, a Google Faculty Research Award, a Facebook Research Award, and Best Paper Awards at the 2017 ACM Conference on Information-centric Networking (ICN) and the 2019 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN).