Fast and Efficient n-Metrics for Multiple Graphs

Kimia Shayestehfard
kshayestehfard@ece.neu.edu
Northeastern University
Boston, MA, USA

Dana Brooks
brooks@ece.neu.edu
Northeastern University
Boston, MA, USA

Stratis Ioannidis
ioannidis@ece.neu.edu
Northeastern University
Boston, MA, USA

ABSTRACT

Graphs are widely used in diverse areas such as chemistry, engineering and network science to express relations among a set of objects. In all these fields, measuring the similarity among a group of graphs or indicating the correspondence among their nodes is of great importance. Many of the algorithms introduced for this purpose are not metrics and the few graph distance functions that do satisfy the properties of a metric are extremely slow to compute. In this work, we propose a fast and efficient framework to compute the distance among a group of graphs. Our method can improve the computation time up to 20 times over existing methods.

CCS CONCEPTS

• **Computing methodologies: parallel computing methodologies**;

KEYWORDS

graph distances, parallelization, n-metric distances, graph mining

ACM Reference Format:
Kimia Shayestehfard, Dana Brooks, and Stratis Ioannidis. 2022. Fast and Efficient n-Metrics for Multiple Graphs. In *Proceedings of 17th International Workshop on Mining and Learning with Graphs (MLG'22).* ACM, Washington DC, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION.

Graph similarities and graph isomorphism have applications across domains such as machine learning, data mining, pattern recognition and transfer learning [9, 16, 20, 23]. Graph distances arise naturally in this literature: intuitively, given a group of (unlabeled) graphs, their pairwise distances are scores quanitifying their structural differences or correspondences among their nodes. One very desirable property of such distances is that they are indeed true distances, meaning that they satisfy the properties of a metric. The distance-based scores satisfying the metric properties have stronger performance compared to non-metrics [4]. Non-compliance with metric properties can result in poor performance in clustering tasks and increase the misclassification rate in classifiers [4]. A canonical way to compute a distance score between two graphs is to find a mapping between the two graphs that minimizes their edge discrepancies. The edit distance [13, 15] and the maximum common subgraph distance [5, 6] are two classic examples of metric distance scores between two graphs. Although these distance functions are metrics, they are hard to compute. Bento & Ioannidis [4] recently introduced a family of metrics that are tractable but are limited to comparing two graphs.

In this paper we study the problem of computing the distance between a group of more than two graphs via a distance-based score, known as multi-distances. We specifically study the multi-distances that satisfy the n-metrics properties (the generalization of the metric properties to multiple graphs). In computing multi-distances, a highly desirable property is consistency of alignment [24], i.e., a distance-based score that produces mapping between every two pairs of graphs should give joint mappings among all graphs. There has been some work on computing multi-distances that does not satisfy n-metrics property but enforces consistency of alignment constraint [18], [8], [34]. Fermat distance function introduced by Kiss et al. [19], measures the distance between a group of graphs that satisfies n-metrics properties and alignment consistency [19]. Safavi and Bento [28] introduced G-align distance that jointly computes the graph distance between a group of graphs incorporating n-metrics properties and satisfying consistency of alignment [28].

To compute multi-distances in a setting that we have large number of graphs in the graph set (e.g., 100 graphs), it would take weeks for Fermat distance function. [19] and G-align distance function [28] to compute a distance score, while in our framework it takes at most a few hours hours to come up with a distance score. In this work we introduce a fast and scalable framework to compute the distance among a group of graphs. We make the following contributions:

- Our framework can be applied to a group of n-metrics multi-distance functions.
- Our framework can be parallelized and significantly improves the speed of n-metrics multi-distances algorithms.
- We report on a set of experiments, demonstrating that our method is up to 20 times faster compared to baselines, while our best accuracy is ranging from 10% lower than the baselines to 60% better than the baselines.

The remainder of the paper is organized as follows. We review related work in Sec. 2. We review background in Sec. 3. We present our main algorithm including the problem formulation and solution in Sec. 4, our experiment setup in Sec. 5 and we conclude in Sec. 6.

2 RELATED WORK.

Graph distance (or similarity) scores have been popular in various fields such as in image processing [9], chemistry [2], social network analysis [20, 23] and transfer learning [16]. When graphs are labeled, graph distances are easy to define [20, 25, 30]. Examples of distances in this setting are the chemical distance [22] and the Chartrand-KubikiShultz (CKS) [7] distance. The edit distance [15, 29] and the maximum common subgraph distance [5, 6] are two classic

MLG'22, Aug 15, 2022, Washington DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

examples of graph distances that have versions for both labeled graphs and unlabeled graphs. All four of these distances are metrics and are hard to compute.

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 the embeddings. Several approaches follow such an approach via embedding graphs into real vectors by computing their edit distances to a set of prototype graphs [26, 27] or through mapping graphs to spaces determined by their spectral decomposition [11, 32, 35]. In general, such approaches are not as discriminative as the metrics considered here [4], because embeddings only summarize the graph structure. Bento & Ioannidis [4], in contrast, introduced a family of metrics for graph distances that are computationally tractable but limited to computing the distance between two graphs.

To compute the distances among a group of more than two graphs, it is important that the distance function satisfies consistency of alignment [24]. There has been some work on computing multi-distances that enforce this constraint [8, 18, 34]; however none of these methods satisfy n-metrics properties. Fermat distance, proposed by Kiss et al., 2016 [19] and G-align distance proposed by Safavi & Bento, 2018 [28] are two approaches to measure the distances among a group of graphs in a way that satisfies (pseudo) n-metrics properties and alignment consistency [19]. We apply our framework to these two graph distance approaches to enhance their scalability and computational speed.

3 BACKGROUND.

Let $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_n\}$ be a group of n undirected graphs. For a given graph $\mathcal{G}_i = (\mathcal{V}, \mathcal{E}_i) \in \mathcal{G}$, $\mathcal{V} = [m] \equiv \{1, 2, \cdots, m\}$ indicates the node set and the edge set is denoted by $\mathcal{E}_i \subseteq [m] \times [m]$. This graph is represented by adjacency matrix $A_i \in \{0,1\}^{m\times m}$. The entries of this adjacency matrix are indexed by the nodes in $\mathcal{V}$. We denote the set that contains all such matrices by $\Omega \subseteq \mathbb{R}^{m\times m}$.

Consider two graphs $\mathcal{G}_i = (\mathcal{V}, \mathcal{E}_i)$ and $\mathcal{G}_j = (\mathcal{V}, \mathcal{E}_j)$ sampled from $\mathcal{G}$, with adjacency matrices $A_i, A_j \in \Omega$. To compute the distance between these two graphs, one possible approach is to find a mapping between nodes and compute an edge discrepancy between them. A mapping between $\mathcal{G}_i$ and $\mathcal{G}_j$ can be represented by a permutation matrix $P \in \mathcal{P}^m$, i.e., $\mathcal{P}^m \triangleq \{P \in \{0,1\}^{m\times m};\ P1 = 1,\ P^T1 = 1\}$.

However, this mapping is computationally intractable [3, 4]. To address this issue, Bento and Ioannidis [4] introduced a family of tractable distance metrics for comparing two graphs. They introduced a distance function $d_S : \Omega^2 \longmapsto \mathbb{R}$, defined as:

$$d_S(A, B) = \min_{P\in\mathcal{W}^m} \|AP - PB\| + \beta \mathrm{tr}(P^T D_{A,B}), \tag{1}$$

where $\beta > 0$ is a positive regularization parameter, $\|\cdot\|$ is a matrix norm, tr is the trace operator, matrix $D_{A,B} \in \mathbb{R}^{m\times m}$ represents a dissimilarity between nodes across the two graphs, and matrix P is a doubly stochastic alignment matrix, that is, $P \in \mathcal{W}^m$ where

$$\mathcal{W}^m \triangleq \{P \in [0,1]^{m\times m};\ P1 = 1,\ P^T1 = 1\}. \tag{2}$$

Matrix $D_{A,B}$ is generally a distance matrix, where each element represents the pairwise distances between the embeddings or features of nodes across two graphs. For example, for two matrices with graph features $X_A \in \mathbb{R}^{m\times d}$ and $X_B \in \mathbb{R}^{m\times d}$ that map nodes of a graph into a lower-dimensional space, i.e. $d < m$, $D_{A,B}$ is:

$$D_{A,B} = [D_{a,b}]_{a\in\mathcal{V}, b\in\mathcal{V}} \in \mathbb{R}^{m\times m}, \text{ where} \tag{3a}$$

$$D_{a,b} = \|x_a^A - x_b^B\|_2 \qquad \forall\, a \in \mathcal{V}, b \in \mathcal{V}, \tag{3b}$$

where x_a^A indicates the a-th row of matrix X_A and x_b^B indicates the b-th row of X_B.

A naïve way to compute the distance among a group of $n > 2$ graphs is to generalize a distance function $d(A_i, A_j)$ between two graphs to n graphs, i.e., if $d(A_i, A_j)$ is a distance function between two graphs that satisfies metric property like Eq. (1), $d(A_1, A_2, \ldots, A_n) = \sum_{i,j\in[n]} d(A_i, A_j)$. The problem with this generalization is that it does not guarantee the alignment consistency between multiple graphs. Alignment consistency states that if P_{ij} aligns $\mathcal{G}_i$ with $\mathcal{G}_j$, and P_{jl} aligns $\mathcal{G}_j$ with $\mathcal{G}_l$, the mapping matrix P_{il} should keep the consistency of mappings under transitivity, i.e., $P_{il} = P_{ij}P_{jl}$ [28]. We describe next two functions, Fermat distance and G-align distance, for measuring the distance among a group of graphs that satisfy consistency of alignment [28] and are (pseudo) n-metrics [28].

3.1 Fermat Distance.

Let $d(A, B)$ be a metric for two graphs such that $d : \Omega^2 \longmapsto \mathbb{R}$. Then the Fermat distance function [19] associated with d is the map $d_F : \Omega^n \longmapsto \mathbb{R}$ defined by:

$$d_F(A_1, A_2, \ldots, A_n) = \min_{A_0\in\Omega} \textstyle\sum_{i=1}^n d(A_i, A_0), \tag{4}$$

where $d_F(A_1, A_2, \ldots, A_n)$ is the alignment score for a set of graphs. If d is a metric like the d_s function given in Eq. 1, then the Fermat distance function induced by a d function is n-metric [28]. The Fermat distance function induced by Eq. (1) and matrix $D = 0$ is:

$$d_F(A_1, A_2, \ldots, A_n) = \min_{P_i\in\mathcal{W}^m, \forall i\in[n] A_0\in\Omega} \sum_{i=1}^n G(P_i, A_0; A_i), \tag{5}$$

where $\mathcal{W}^m$ represents the set of doubly stochastic matrices and $G : \mathcal{W}^m \times \Omega \times \Omega \longmapsto \mathbb{R}$ is formulated as follows:

$$G(P_i, A_0; A_i) = \|A_iP_i - P_iA_0\|. \tag{6}$$

Graph $\mathcal{G}_0$, corresponding to A_0, represents the center of set $\mathcal{G}$. Fig. 1 illustrates how the center graph in Fermat distance function has the minimum distance from all graphs in the graph set.

The Fermat distance function in Eq. (5) satisfies consistency of alignment, and is a pseudo n-metric [28]. It satisfies consistency of alignment, since $P_{ij} = P_i({P_l}^T P_l){P_j}^T = P_i{P_j}^T = P_{il}P_{lj}$. This optimization problem is not convex; nevertheless, it can be solved approximately via alternating minimization [16]. In alternating minimization, at each iteration $t \in \mathbb{N}$, we update A_0 and $\{P_i\}_{i\in[n]}$ as follows:

Updating A_0**.** Given that $\{P_i\}_{i\in[n]}$ is fixed and $D = 0$, minimizing Eq. (5) w.r.t A_0 leads to the following problem:

$$\min_{A_0\in\mathbb{R}^{m\times m}} \sum_{i=1}^n \|A_iP_i^{(t-1)} - P_i^{(t-1)}A_0^{(t)}\| \tag{7}$$

This problem is convex and at step $t \in \mathbb{N}$ can be solved via convex optimization.

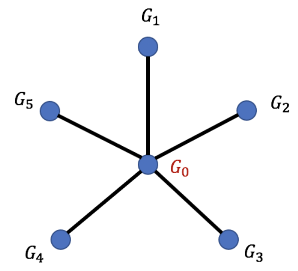


Figure 1: Illustration of Fermat distance. $\mathcal{G}_0$ is the center graph and all graphs in the graph set are getting aligned w.r.t this graph.

Updating $\{P_i\}_{i\in[n]}$. Given that $A0$ is fixed and $D = 0$, minimizing Eq. (5) w.r.t $\{P_i\}_{i\in[n]}$ leads to the following problem.

$$\min_{P_i \in \mathcal{W}^m} \sum_{i=1}^{n} \|A_i P_i^{(t)} - P_i^{(t)} A_0^{(t)}\| \quad (8)$$

This step is convex and can be solved via optimization toolboxes such as CVXPY [1, 10] or efficient algorithms such as Frank-Wolfe algorithm [14]. Eq. (7) has nm^2 parameters and nm^2 constraints and Eq. 8 contains n optimization problems with m^2 parameters and m^2 constraints. In practice, we need between 80 to 100 iterations to compute the Fermat distance among 100 graphs.

3.2 G-align Distance

Safavi and Bento [28] introduced the G-align distance function, that is a map $d_G : \Omega^n \longmapsto \mathbb{R}$ defined by:

$$d_G(A_1, \ldots, A_n) = \min_{P_{ij}\in S} \frac{1}{2} \sum_{i,j\in[n]} G(P_{ij}; A_i, A_j), \quad (9)$$

where $G(P; \mathbf{A})$ is given in Eq. (6) and matrix $D = 0$.

$$\begin{aligned} S = \{\{P_{ij}\}_{i,j\in[n]} : P_{ij} \in \mathcal{W}^m, \forall i, j \in [n], \\ P_{il}P_{lj} = P_{ij}, \forall i, j, l \in [n], P_{ii} = I, \forall i \in [n]\} \end{aligned} \quad (10)$$

Let $\boldsymbol{P} \in R^{nm\times nm}$ be a matrix with n^2 blocks such that the (i, j)th block is P_{ij}, i.e.:

$$\boldsymbol{P} = \begin{bmatrix} I & P_{12} & P_{13} & \ldots & P_{1n} \\ P_{21} & I & P_{23} & \ldots & P_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & P_{n3} & \ldots & I \end{bmatrix}. \quad (11)$$

Then the constraints can be written in the G-align distance function:

$$d_G(A_1, \ldots, A_n) = \min_{\substack{P_{ij}\in\mathcal{W}^m, \\ P_{ii}=I, \boldsymbol{P}\geq 0}} \frac{1}{2} \sum_{i,j\in[n]} G(P_{ij}; A_i, A_j), \quad (12)$$

where $\boldsymbol{P} \geq 0$ indicates that $\boldsymbol{P}$ is positive semidefinite. Eq. (12) satisfies consistency of alignment and is a pseudo n-metric [28]. Given the constraints in Eq. (12), this problem is convex and can be solved via optimization toolboxes such as CVXPY [1, 10] or more efficient algorithms such as the Frank-Wolfe algorithm [14].

Eq. (12) contains n^2m^2 variables. The constraints $P_{ij} \in \mathcal{W}^m$, $P_{ii} = I$ and $\boldsymbol{P} \geq 0$ have $O(n^2m^2)$, $O(nm)$ and polynomial worst-case complexity [31], respectively. The quadratic nature of matrix $\boldsymbol{P}$ makes it very expensive to solve.

3.3 Extensions.

The Fermat and G-align distance functions are not limited to graphs with equal numbers of nodes. They can be extended to graph sets with a variable number of nodes by adding "dummy" nodes such that all graphs then have an equal number of nodes [4]. One of several ways to do so is to first find the maximum number of nodes $\mathrm{m}_{\max}$ in the graph set and then expand all graphs with $|\mathcal{V}_i| < \mathrm{m}_{\max}$, $i \in [n]$ by adding "dummy" nodes such that all graphs have $\mathrm{m}_{\max}$ nodes. In the expanded graphs, in order to differentiate "dummy" nodes from actual nodes, "dummy" nodes are connected to each other as well the actual nodes by edges with a small weight (e.g., 0.01).

4 ACCELERATED METRIC MULTI-DISTANCES.

4.1 Problem Formulation.

Given a graph set $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_n\}$ with n undirected graphs, our goal is to measure distance (similarity) among graphs via a fast n-metric multi-distance function that satisfies consistency of alignment. For this purpose we present our framework that accelerate both the Fermat distance function [19] and the G-align distance function [28].

4.2 Solution.

There are 3 paths that we can take to accelerate the Fermat [19] and G-align distances [28]. Below we describe these paths in details.

4.2.1 G-Parallel: Grouping and Parallelizing Graphs. This method has a recursive structure. We start by dividing the full group of graphs into a collection of smaller groupings of graphs. In each group, we first compute the mappings among graphs and the center graph for each of these groupings, which can be done in parallel. Once we find the center graph for each group, we compute the mappings among them to find $\mathcal{G}_0$. We keep grouping graphs and computing their centers until we find the final center graph for the whole set. In G-Parallel, every operation that happens in the groups can happen in parallel. Algorithm 1 illustrates how our framework works, where K represents the number of graphs in each group and the "for loops" are processed in parallel. In the final stage, we stop when the number of graphs is less than K. Once we have the final center graph fixed, we can compute the graph distances via the Fermat distance given in Eq. (5). Note that however, in all the intermediate stages we can use both Fermat or a variant of G-align distance which we will describe below. For both Eq. (5) and Eq. (12), rather than having inputs of size n, we have inputs of size K and because the cost is polynomial in the number of graphs (super quadratic), this is a significant benefit.

The total number of stages in this recursive process is $O(\log_K n)$ and the total number of problems with K graphs that we need to compute is $O(n)$.

Algorithm 1: Our Framework

```
Input: $\mathcal{G} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$, $K$ : number of graphs in each group.
Output: $\mathcal{G}_{0_{\text{out}}}$ : the center graph.
for $k = \{0, 1, 2, \cdots, [\frac{n}{K}]\}$ do
    $\tilde{\mathcal{G}}_{\boldsymbol{k}} = \{\mathcal{G}_{1+k\times K}, \mathcal{G}_{2+k\times K}, \cdots, \mathcal{G}_{K+k\times K}\}$
    $\tilde{\mathcal{G}_0}^k = \text{center}(\tilde{\mathcal{G}}_{\boldsymbol{k}})$
end for
$\tilde{\mathcal{G}} = \{\tilde{\mathcal{G}_0}^k$, for $k = \{0, 1, 2, \cdots, [\frac{N}{K}]\}\}$
if $[\frac{n}{K}] > K$ then
    while $[\frac{n}{K}] > K$ do
        $n = [\frac{n}{K}]$
        $\mathcal{G} = \tilde{\mathcal{G}}$
        for $k = \{0, 1, 2, \cdots, [\frac{n}{K}]\}$ do
            $\tilde{\mathcal{G}}_{\boldsymbol{k}} = \{\mathcal{G}_{1+k\times K}, \mathcal{G}_{2+k\times K}, \cdots, \mathcal{G}_{K+k\times K}\}$
            $\tilde{\mathcal{G}_0}^k = \text{center}(\tilde{\mathcal{G}}_{\boldsymbol{k}})$
        end for
        $\tilde{\mathcal{G}} = \{\tilde{\mathcal{G}_0}^k$ for $k = \{0, 1, 2, \cdots, [\frac{n}{K}]\}\}$
    end while
end if
$\mathcal{G}_{\text{out}} = \tilde{\mathcal{G}}$
$\mathcal{G}_{0_{\text{out}}} = \text{center}(\mathcal{G}_{\text{out}})$
```

Computing The Center Graph for Fermat Distance. For the Fermat distance, we compute the mappings and the center graph simultaneously via Eq. (5)- (8).

Computing The Center Graph for G-align Distance. For the G-align distance, in contrast, in order to compute the center graph we first compute the mapping matrix $\boldsymbol{P}$ by solving the problem in Eq. (12) [28] and then take the first block column of $\boldsymbol{P}$, i.e., $\{P_{i1}\}_{i=1}^{K}$, and project each $P_{i1} \in \mathcal{W}^m$ onto the set of permutation matrices, i.e.:

$$\tilde{P_{i1}} = \Pi_{\mathcal{P}^m}(P_{i1}), \tag{13}$$

where $\Pi_{\mathcal{P}^m}$ is the orthogonal projection to $\mathcal{P}^m$. This problem can be solved in polynomial time with the Hungarian algorithm [21]. (Note that our selection of $\{P_{i1}\}_{i=1}^{n}$ is arbitrary and we could choose any block column.) We align all graphs with the first graph as follows:

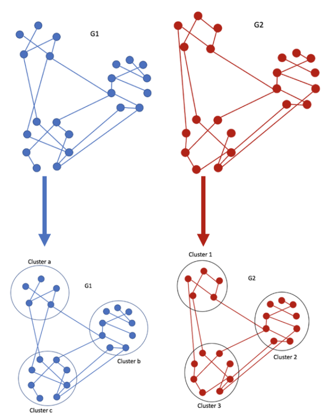
$$\tilde{A_i} = \tilde{P_{i1}}^T A_i \tilde{P_{i1}} \quad \forall i \in [K] \tag{14}$$

Finally we compute the adjacency matrix of the center graph:

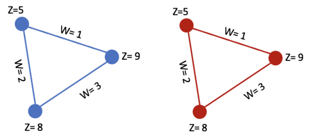
$$\min_{\hat{A}_{0_{j,k}} \in [0,1], \forall j,k \in [m_0]} \quad \textstyle\sum_{i=1}^{n} \|\tilde{A}_i - \hat{A}_0\| \tag{15}$$

Eq. (15) is convex and can be solved via CVXPY toolbox [1, 10]. Since the elements of $\hat{A}$ are in the range $[0, 1]$, we binarize the elements of the adjacency matrix for the center graph $\mathcal{G}_0$ by using a threshold. The time complexity of these equations is dominated by the one for Eq. (12).

4.2.2 CG-Parallel: Clustering and parallelization of stochastic block models. CG-Parallel has a recursive structure. In stochastic block model graphs such as community graphs, to further speed up the computation of n-metric multi-distances, within each smaller groups of graphs we first cluster nodes in each graph via a clustering algorithm such as k-means. After this step we will have multiple clusters and a sparse matrix representing the edges connecting those clusters (see Fig. 2a). We then create reduced size weighted graphs in which each node represents a cluster and weighted edges represent the number of edges connecting two clusters in the original graphs. Ideally we want clusters with similar numbers of nodes to get mapped to each other. For this reason in matrix D (3b) we set the node features or embeddings (Xs) to the number of nodes in the corresponding cluster. We compute the graph distances across the clusters via G-align or Fermat distance (see Fig. 2b). Once we compute the mappings among the nodes in the small graphs, we know which clusters should be aligned with each other across graphs. After aligning clusters and the sparse matrix connecting the clusters, we solve the optimization problem given in Eq. (15) to find the center of clusters and the edges connecting clusters. Finally given the cluster centers and the edges connecting them, we recover the original large center graphs.



(a) Breaking graphs into small clusters



(b) Small weighted graphs representing the original graphs.

Figure 2: The procedure to break graphs into multiple clusters and find the mappings across clusters rather than the original larger graphs

This is a recursive process. We repeat these steps until we find a center for the whole graph set. Here again many of the steps can be parallelized.

Note that this method can be applied to graphs that do not follow stochastic models, as well. In this case, we can break graphs in equal sized clusters or set a minimum and maximum cluster size in k-means to control the clustering procedure.

The total number of stages in this recursive process is $O(\log_K^n)$ and the total number of problems with K graphs that we need to compute is $O(cn)$ and the time complexity for k-means is $O(m^2)$.

4.2.3 C-NonParallel: Clustering Graphs. This method is similar to Sec. 4.2.2. The only difference is that the structure is non-recursive, i.e., we do not divide graphs into smaller groups. We only break each graph into c clusters using k-means algorithm and compute the center graph by computing the centers for the clusters. The total number of problems that we need to compute is $O(cn)$ and the time complexity for k-means is $O(m^2)$.

	Community	Grid
$\lvert\mathcal{V}\rvert$	45	36
$\lvert\mathcal{E}\rvert_{\text{ave}}$	98	265
Fermat Solver	CVXPY + AM	CVXPY+ AM
G-align Solver	CVXPY	CVXPY

Table 1: Dataset summary, along with the algorithm and solver used to compute graph mappings. CVXPY is the toolbox used to solve the G-align distance problem. In Fermat distance in addition to CVXPY solver we use alternating minimization (AM) to compute the graph distance.

5 EXPERIMENTAL SETUP

5.1 Datasets

We performed experiments on two synthetic datasets.

Community. We generated a community graph with 3-communities, from the stochastic block model [33]. The graph has $|\mathcal{V}| = 45$ total nodes and $[5, 15, 17]$ nodes in the communities. Each community is generated by the Erdős-Rényi model (E-R) [12]. The probability for edge creation in each community is $p = 0.7$ and $0.05|\mathcal{V}|$ inter-community edges were added u.a.r. We obtain the code for graph construction from [33].

Grid. We constructed a $2-$D grid graph with $|\mathcal{V}| = 36$ nodes using NetworkX [17].

In both community and grid graphs, in order to build the graph set we started with a single graph. Then we generated 100 random graphs by randomly permuting the graph and then adding noise by randomly removing 10% of edges and add back the same amount of new edges added.

Table 1 gives details about the graphs as well as the algorithms we used.

5.2 Comparison Algorithms

We compare our methods against two baseline computation of the Fermat [19] and G-align [28] distances. We compare these baselines to three of our frameworks.

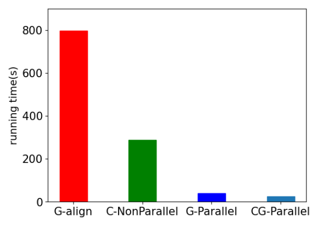
G-Parallel: Compute the graph distance diving graphs into small groupings and parallelization as defined in Sec. 4.2.1.

CG-Parallel: Compute the graph distance by diving graphs into small groups and breaking graphs into clusters as defined in Sec. 4.2.2.

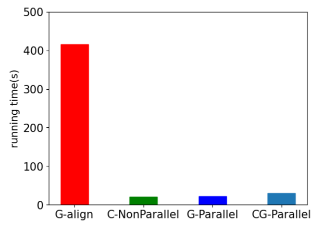
C-NonParallel: Compute the graph distance by breaking each graph into clusters and finding the mapping between clusters rather than the whole graph as defined in Sec. 4.2.3.

5.3 Performance Evaluation.

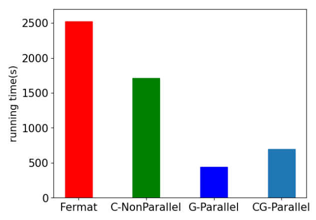
To assess the performance our framework we designed two experiments. In the first experiment we measured the algorithm speed improvement for Fermat distance [19] and G-align distance [28] before and after applying our framework. In the second experiment our goal is to study the impact of of parallelization on the accuracy of mappings. For this purpose, we computed the center graph in the baselines and in our framework. We then computed the distance of



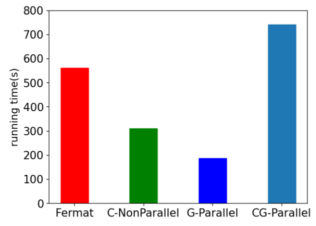
(a) Running time with G-align distance baseline and applying our frameworks to G-align distance using community graphs.



(b) Running time with G-align distance baseline and applying our frameworks to G-align distance using grid graphs.



(c) Running time with Fermat distance baseline and applying our frameworks to Fermat distance using community graphs.



(d) Running time with Fermat distance baseline and applying our frameworks to Fermat distance using grid graphs.

Figure 3: Running time for computing the graph distance in community graphs (left) and grid graphs (right) given the baselines and three different scenarios, G-Parallel, CG-Parallel and C-NonParallel.

final A_0 from the graph set via

$$\text{ACC} = \frac{1}{n}\sum_{i=1}^{n}\frac{\|P_i^T A_i P_i - A_0\|}{\|A_1\|}. \tag{16}$$

5.4 Parallelization Impact on Scalability and Accuracy.

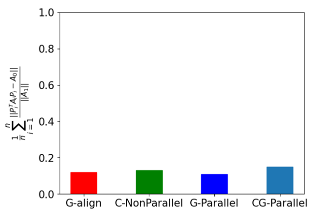
Due to scalability issues with the baselines we computed the distance among only 12 graphs. Fig. 3 represents the time consumed for the baselines and our 3 methods to compute the graph distance. Our methods make the n-metric multi-distance functions significantly faster. Among our methods, G-Parallel is having the best performance.

For both the baselines if we increase the number of graphs in the graph set to 100 graphs, it will take weeks to compute the graph distances while with our framework while dividing graphs into groups we can compute the graph distance in at most in 4.3 hours. In some scenarios this happens in less than half an hour. Table 2 summarizes the running time applying the G-Parallel scenario to both Fermat and G-align distances.

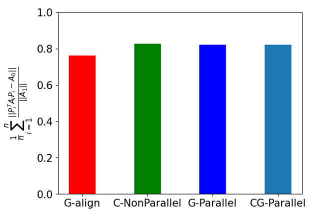
In general all our 3 methods outperforms the baselines. The only exception is CG-NonParallel applied to Fermat distance for grid graphs. The running time for applying Fermat distance is longer compared to G-align, since the Fermat distance function is not convex and we need to use at least 80 iterations in alternating minimization to solve the problem.

	Community	Grid
Group-Parallel (Fermat)	15784.82 (s)	3584.53 (s)
Group-Parallel (G-align)	1973.82 (s)	1252.65 (s)

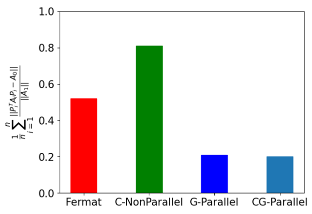
Table 2: Running time in seconds (s) using G-Parallel scenario for 100 graphs.



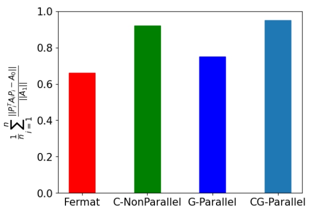
(a) Accuracy of using G-align distance base line and applying our frameworks to G-align distance using community graphs.



(b) Accuracy of using G-align distance base line and applying our frameworks to G-align distance using grid graphs.



(c) Accuracy of using Fermat distance base line and applying our frameworks to Fermat distance using community graphs.



(d) Accuracy of using Fermat distance base line and applying our frameworks to Fermat distance using grid graphs.

Figure 4: Accuracy of computing the graph distance in community graphs (left) and grid graphs (right) given the baselines and three different scenarios, G-Parallel, CG-Parallel and C-NonParallel. G-align distance has better performance compared to Fermat distance. Moreover, due to clustered structure of community graphs clustering clustering and grouping graphs in CG-Parallel even improves the accuracy.

We also investigated the impact of our framework on the accuracy of the graph distances via Eq. (16). Fig. 4 shows the comparison of the baselines and our methods. Our results demonstrate that the G-align distance is more accurate compared to Fermat distance in computing the graph distance. Due to the clustered structure of community graphs clustering and grouping graphs in CG-Parallel improves accuracy even more. Based on the results in Figs. 3 and 4 we conclude that G-Parallel is the best method to improve computational speed while preserving accuracy.

6 CONCLUSION.

In this work we presented a framework that improves the scalability and speed of two n-metric multi-distances. Our method can compute the distance among a large group of graphs with while maintaining accuracy.

ACKNOWLEDGMENTS

The authors gratefully acknowledge support by the National Science Foundation (grants IIS-1741197, CCF-1750539) and Google via GCP credit support.

REFERENCES

[1] Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. 2018. A rewriting system for convex optimization problems. *Journal of Control and Decision* 5 (2018), 42–60.
[2] Frank H Allen. 2002. The Cambridge Structural Database: a quarter of a million crystal structures and rising. *Acta Crystallographica Section B: Structural Science* 58, 3 (2002), 380–388.
[3] László Babai. 2016. Graph isomorphism in quasipolynomial time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, Cambridge, MA, USA, 684–697.
[4] Jose Bento and Stratis Ioannidis. 2018. A family of tractable graph distances, In Proceedings of the 2018 SIAM International Conference on Data Mining. *Appl. Netw. Sci.*, 333–341.
[5] Horst Bunke. 1997. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters* 18, 8 (1997), 689–694.
[6] Horst Bunke and Kim Shearer. 1998. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters* 19, 3-4 (1998), 255–259.
[7] Gary Chartrand, Grzegorz Kubicki, and Michelle Schultz. 1998. Graph similarity and distance in graphs. *Aequationes Mathematicae* 55, 1 (1998), 129–145.
[8] Yuxin Chen, Leonidas J. Guibas, and Qi-Xing Huang. 2014. Near-Optimal Joint Object Matching via Convex Relaxation. In *ICML 2014 (JMLR Workshop and Conference Proceedings, Vol. 32)*. JMLR.org, Beijing, China, 100–108.
[9] Donatello Conte, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence* 18, 03 (2004), 265–298.
[10] Steven Diamond and Stephen Boyd. 2016. CVXPY: A Python-embedded modeling language for convex optimization. *JMLR* 17, 1 (2016), 2909–2913.
[11] Hewayda Elghawalby and Edwin R Hancock. 2008. Measuring graph similarity using spectral geometry. In *International Conference Image Analysis and Recognition*. Springer, 517–526.
[12] Paul Erdös and Alfréd Rényi. 1959. On random graphs I. *Publ. Math. Debrecen* 6 (1959), 290–297.
[13] Andreas Fischer, Ching Y Suen, Volkmar Frinken, Kaspar Riesen, and Horst Bunke. 2015. Approximation of graph edit distance based on Hausdorff matching. *Pattern Recognition* 48, 2 (2015), 331–343.
[14] Marguerite Frank, Philip Wolfe, et al. 1956. An algorithm for quadratic programming. *Naval Research Logistics Quarterly* 3, 1-2 (1956), 95–110.
[15] Michael R Garey and David S Johnson. 2002. Computers and Intractability, vol. 29.
[16] Andrey Gritsenko, Yuan Guo, Kimia Shayestehfard, Armin Moharrer, Jennifer Dy, and Stratis Ioannidis. 2021. Graph Transfer Learning. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, IEEE, Auckland, New Zealand, 141–150.
[17] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX*. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).
[18] Qi-Xing Huang and Leonidas Guibas. 2013. Consistent shape maps via semidefinite programming, In Computer Graphics Forum. *Computer Graphics Forum* 32, 5, 177–186.
[19] Gergely Kiss, Jean-Luc Marichal, and Bruno Teheux. 2018. A generalization of the concept of distance based on the simplex inequality. *Contributions to Algebra and Geometry* 59, 2 (2018), 247–266.
[20] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. 2013. Deltacon: A principled massive-graph similarity function. In *Proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, SIAM, Austin, Texas, USA, 162–170.
[21] Harold W Kuhn. 1955. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly* 2, 1-2 (1955), 83–97.
[22] Vladimír Kvasnička, Jiří Pospíchal, and Vladimír Baláž. 1991. Reaction and chemical distances and reaction graphs. *Theoretica chimica acta* 79, 1 (1991), 65–79.
[23] Owen Macindoe and Whitman Richards. 2010. Graph comparison using fine structure analysis. In *2010 IEEE Second International Conference on Social Computing*. IEEE, IEEE Computer Society, Minneapolis, Minnesota, USA, 193–200.
[24] Andy Nguyen, Mirela Ben-Chen, Katarzyna Welnicka, Yinyu Ye, and Leonidas Guibas. 2011. An optimization approach to improving collections of shape maps, In Computer Graphics Forum. *Comput. Graph. Forum* 30, 1481–1491.
[25] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. 2010. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications* 1, 1 (2010), 19–30.

[26] Kaspar Riesen and Horst Bunke. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing* 27, 7 (2009), 950–959.

[27] Kaspar Riesen and Horst Bunke. 2010. *Graph classification and clustering based on vector space embedding*. Vol. 77. World Scientific.

[28] Sam Safavi and José Bento. 2019. Tractable n-Metrics for Multiple Graphs. In *ICML*. PMLR, Long Beach, California, USA, 5568–5578.

[29] Alberto Sanfeliu and King-Sun Fu. 1983. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics* 3 (1983), 353–362.

[30] Sucheta Soundarajan, Tina Eliassi-Rad, and Brian Gallagher. 2014. A guide to selecting a network similarity method. In *Proceedings of the 2014 Siam international conference on data mining*. SIAM, 1037–1045.

[31] Lieven Vandenberghe and Stephen Boyd. 1996. Semidefinite programming. *SIAM review* 38, 1 (1996), 49–95.

[32] Richard C Wilson and Ping Zhu. 2008. A study of graph spectra for comparing graphs and trees. *Pattern Recognition* 41, 9 (2008), 2833–2841.

[33] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *ICML*. PMLR, PMLR, Stockholm, Sweden, 5708–5717.

[34] Xiaowei Zhou, Menglong Zhu, and Kostas Daniilidis. 2015. Multi-image matching via fast alternating minimization. In *Proceedings of the IEEE International Conference on Computer Vision*. IEEE Computer Society, Santiago, Chile, 4032–4040.

[35] Ping Zhu and Richard C Wilson. 2005. A study of graph spectra for comparing graphs.. In *BMVC*.