AlignGraph: A Group of Generative Models for Graphs

Kimia Shayestehfard Dana Brooks Stratis Ioannidis*

Abstract

It is challenging for generative models to learn a distribution over graphs because of the lack of permutation invariance: nodes may be ordered arbitrarily across graphs, and standard graph alignment is combinatorial and notoriously expensive. We propose AlignGraph, a group of generative models that combine fast and efficient graph alignment methods with a family of deep generative models that are invariant to node permutations. Our experiments demonstrate that our framework successfully learns graph distributions, outperforming competitors by $25\% - 560\%$ in relevant performance scores.

1 Introduction

Graph generative models have applications across domains like chemistry, neuroscience and engineering. Generative models learn a distribution over graphs, and are used to subsequently sample from this distribution. They can, e.g., be used to predict interfaces between proteins during drug design and discovery [1, 2], or to perform hypothesis testing and simulation for social networks, when collecting real graphs is difficult [3, 4].

Traditional generative models for graphs such as the Barabási-Albert [5], Erdös-Rényi [6], and stochastic block models [7] generate graphs with provable formal properties but which often lack realism. For example, the Erdös-Rényi model produces graphs with a light-tailed degree distribution [5, 8], while the Barabási-Albert model fails to generate graphs with a high clustering coefficient [9]. Deep generative models such as variational autoencoders [10] and graph recurrent neural networks [11, 12] have shown great potential in learning distributions from graph datasets, at greater fidelity than traditional models. However, learning a distribution of graphs over a dataset poses a significant challenge because of the *lack of permutation invariance*, since graph nodes may be subject to arbitrary permutations across graphs: the correspondence between nodes in different graph samples may be a priori unknown.

This is a problem because state-of-the-art generative models, like the ones listed above, rely on latent node embeddings. Such embeddings vary drastically even under nearly isomorphic graphs [13]. In turn, this can hamper the fidelity of the graph generation process significantly. Note that this is a much harder setting than, e.g., images or text, where inputs have a canonical orientation. Finding the correspondence between graph nodes is a notoriously hard problem [14, 15, 11, 16], and it is exacerbated when the number of sampled graphs is large. To that end, we propose AlignGraph, a group of *permutation invariant* graph alignment methods combined with their application to a group of base generative models. Our main contributions are as follows:

1. AlignGraph incorporates convex graph multi-distance methods in training to achieve permutation invariance. We use these tools both as means to construct alignment in a tractable fashion as well as to create soft penalties in training.
2. AlignGraph is a general flexible framework. It can be applied to a broad class of base generative models, enhancing their permutation invariance. We demonstrate this here by applying it to graph recurrent neural networks [11], gated recurrent attention networks [12] and variational autoencoders [10] as our base generative models.
3. We propose three methods that can be parallelized to accelerate graph multi-distances. Leveraging parallelism, our methods speed up computation by a 40× factor, while maintaining the alignment accuracy and, in some cases, improving it.
4. We conduct experiments on both synthetic and real data, showing that AlignGraph outperforms both our base and other competitor models. We define two performance scores to measure accuracy. We then show that our model achieves $25\% - 250\%$ improvement in those scores over base models and $62.5\% - 4000\%$ improvement over other competitors.

2 Related Work

Graph Embeddings. Graph embeddings map nodes into a lower-dimensional space and have been ap-

*{kshayestehfard, brooks, ioannidis}@ece.neu.edu, Electrical and Computer Engineering Department, Northeastern University, Boston, MA, USA.

plied to link prediction [10, 17, 16], node classification [18, 19, 17, 20], and clustering [20]. Many of the state-of-the-art graph embedding algorithms capture the relative position of nodes on the embedding space [13]. Because of the non-convexity of training objectives and the existence of multiple local minima, even isomorphic graphs can map to completely different embeddings using the same embedding algorithm [13]. This is further exacerbated when graphs are near-isomorphic (i.e., differ in a few edges) as well as when the embeddings are randomized [13]. Embeddings play a central role in graph generative models (see Sec. 3.3), but lack of permutation invariance can introduce significant distortions.

Deep Generative Models. Deep generative models can be categorized into three groups: generative adversarial networks (GANs), variational autoencoders (VAEs), and auto-regressive models. NetGAN [16] learns the distribution of biased random walks over a single graph. GraphVAE [14] and NEVAE [15] use VAEs linking node embeddings to edges. GraphRNN [11] is an auto-regressive model that constructs a graph sequentially over nodes and edges. GRAN [12] is another auto-regressive model that uses graph neural networks (GNNs) with an attention mechanism to generate a block of nodes and edges sequentially.

Several of these methods contain techniques to partially deal with permutation invariance. For example, GraphVAE [14] uses an approximate graph matching to penalize misalignment between each input graph and its corresponding reconstructed graph. NEVAE [15] and GraphRNN [11] use a breadth-first-search node ordering scheme and GRAN [12] marginalizes over a family of canonical node orderings to handle permutation invariance. However, none of these methods address permutation invariance by finding a consistent node ordering across sampled graphs. In comparison, the graph alignment approach we introduce here does exactly this; in addition, it is generic and can be applied to the broad group of base generative models listed above to enhance their permutation invariance (see also Sec. 6).

Graph distances. Classic methods to compute the distance between graphs include the edit distance [21, 22] and the maximum common subgraph distance [23, 24]. Although they are metrics, they are hard to compute. Bento & Ioannidis [25] recently introduced a family of metrics for graph distances that is computationally tractable but limited to computing the distance between two graphs. To compute distances among a larger group of graphs, it is important that the distance function satisfies alignment consistency [26]. There are works on multi-distances that enforce this constraint [27, 28, 29]; however, none of these methods satisfy generalizations of the metric properties. Gromov-Wasserstein Learning (GWL), proposed by Xu et al. [30] satisfies both of these properties. However, GWL has cubic complexity and is not applicable to recurrent neural networks. Recently, two approaches were proposed by Kiss et al. [31] and Safavi & Bento [32] to measure the distance among a group of graphs; we describe both in detail in Sec 3.2. Both satisfy alignment consistency and a generalization of metric properties [31]. However, both are also slow when applied to a large number of graphs.

We propose a framework to accelerate these two graph multi-distance algorithms, by leveraging parallelization and graph coarsening [33]. Graph coarsening has been used in community detection [34, 35], graph embeddings [36] and alignment between two graphs [37]. We coarsen graphs using K-means clustering, and incorporate this accelerated graph alignment method into our framework to address permutation invariance. To the best of our knowledge, we are the first to accelerate graph multi-distances by graph coarsening.

3 Background

3.1 Minimum Distance between two Graphs.

Let $\mathcal{G} = (V, E)$ be an undirected graph with node set $V = [m] \equiv \{1, 2, \ldots, m\}$ and edge set $E \subseteq [m] \times [m]$, represented by adjacency matrix $A \in \{0, 1\}^{m\times m}$. The entries of this adjacency matrix are indexed by the nodes in V. We denote the set that contains all such matrices by $\Omega \subseteq \mathbb{R}^{m\times m}$. Consider two graphs $\mathcal{G}_A = (V, E_A)$, $\mathcal{G}_B = (V, E_B)$ with adjacency matrices $A, B \in \Omega$. One way to measure the distance between these two graphs is to find an alignment between nodes and compute an edge discrepancy (i.e., edit distance [38, 21]) between them. An alignment can be represented by a permutation matrix $P \in \mathcal{P}^m$, where:

$$\mathcal{P}^m \triangleq \{P \in \{0, 1\}^{m\times m};\ P\mathbb{1} = \mathbb{1},\ P^T\mathbb{1} = \mathbb{1}\}. \tag{3.1}$$

However, finding such an alignment is generally computationally intractable [25, 39]. Bento & Ioannidis [25] introduce a distance function $d_S : \Omega^2 \longmapsto \mathbb{R}$, defined as:

$$d_S(A, B) = \min_{P\in\mathcal{W}^m} \|AP - PB\| + \beta\mathrm{tr}(P^T D_{A,B}), \tag{3.2}$$

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 the dissimilarity between nodes across the two graphs, and matrix P is a doubly stochastic alignment matrix, that is, $P \in \mathcal{W}^m$, where

$$\mathcal{W}^m \triangleq \{P \in [0, 1]^{m\times m};\ P\mathbb{1} = \mathbb{1},\ P^T\mathbb{1} = \mathbb{1}\}. \tag{3.3}$$

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 of graph embeddings $Z_A \in \mathbb{R}^{m\times d}$ and $Z_B \in \mathbb{R}^{m\times d}$ that map nodes of a graph into a lower-dimensional space, i.e. $d < m$, $D_{A,B}$ is:

$$D_{A,B} = [D_{a,b}]_{a\in V, b\in V} \in \mathbb{R}^{m\times m}, \text{ and} \tag{3.4a}$$

$$D_{a,b} = ||z_a^A - z_b^B||_2, \qquad \forall\ a \in V, b \in V, \tag{3.4b}$$

where z_a^A indicates the a-th row of matrix Z_A, z_b^B indicates the b-th row of Z_B. Intuitively, the first term in Eq. (3.2) is a probabilistic mapping between nodes of two graphs and the second term penalizes the dissimilarity between the embeddings of the nodes that are mapped to each other. Eq. (3.2) is a pseudometric and a convex optimization problem [25], and thus can be computed efficiently via standard techniques.

3.2 Minimum Distance among n Graphs. Consider a distance function $d(\mathcal{G}_i, \mathcal{G}_j)$ like Eq. (3.2) that induces a (possibly stochastic) alignment matrix P_{ij} between two pairs of graphs. To compute the minimum distance between a group of $n > 2$ graphs, one could simply generalize the distance function $d(\mathcal{G}_i, \mathcal{G}_j)$ to multiple graphs, via $d(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n) = \sum_{i,j\in[n]} d(\mathcal{G}_i, \mathcal{G}_j)$. However, such a generalization does not guarantee the joint alignment between multiple graphs: that is, if P_{ij} aligns $\mathcal{G}_i$ with $\mathcal{G}_j$, and P_{jl} aligns $\mathcal{G}_j$ with $\mathcal{G}_l$, the alignment matrix P_{il} should keep the consistency of alignments under transitivity, i.e., $P_{il} = P_{ij}P_{jl}$ [32]. This property is known as *alignment consistency*. We describe next two distance functions that induce alignments that satisfy this property.

3.2.1 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 [31] associated with d is the map of $d_F : \Omega^n \longmapsto \mathbb{R}$ defined by: $d_F(A_1, A_2, \ldots, A_n) = \min\limits_{A_0\in\Omega} \sum_{i=1}^n d(A_i, A_0)$, capturing the distance among a set of graphs. If d is a metric then the Fermat distance function induced by d is a so-called n-metric [32]. The Fermat distance function induced by Eq. (3.2) is:

$$d_F(A_1, \ldots, A_n) = \min_{\substack{A_0\in\Omega \\ P_i\in\mathcal{W}^m, \forall i\in[n]}} \sum\nolimits_{i=1}^n G(P_i, A_0; A_i), \tag{3.5}$$

where $D = 0$ and $\mathcal{W}^m$ represents the set of doubly stochastic matrices and $G : \mathcal{W}^m \times \Omega \times \Omega \longmapsto \mathbb{R}$ is:

$$G(P_i, A_0; A_i) = \|A_iP_i - P_iA_0\|. \tag{3.6}$$

Graph $\mathcal{G}_0$, corresponding to A_0, represents the center of set $\boldsymbol{\mathcal{G}}$. The Fermat distance function in Eq. (3.5) is a pseudo n-metric [32]. This optimization problem is non-convex; nonetheless, it can be solved approximately via alternating minimization (AM): Eq. (3.5) then reduces to solving two alternating convex optimization problems. The first one has nm^2 parameters and nm^2 constraints. The second problem reduces to n optimization problems with m^2 parameters and m^2 constraints. More details regarding AM iterations can be found in App. A.

3.2.2 G-align distance. Safavi & Bento [32] introduce the G-align distance function, a convex function that satisfies metric properties and alignment consistency. Consider the map $d_G : \Omega^n \longmapsto \mathbb{R}$ defined by:

$$d_G(A_1, \ldots, A_n) = \min_{P_{ij}\in S} \tfrac{1}{2} \sum\nolimits_{i,j\in[n]} G(P_{ij}; A_i, A_j), \tag{3.7}$$

where $G(P_{ij}; A_i, A_j)$ is given by Eq. (3.6) (with $D = 0$),

$$\begin{gathered} S = \{\{P_{ij}\}_{i,j\in[n]} : P_{ij} \in \mathcal{P}^m, \forall i, j \in [n], \\ P_{il}P_{lj} = P_{ij}, \forall i, j, l \in [n], P_{ii} = I, \forall i \in [n]\}, \end{gathered} \tag{3.8}$$

where $\mathcal{P}^m$ is the set of permutation matrices and $P_{il}P_{lj} = P_{ij}$ captures alignment consistency.

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.:

$$\boldsymbol{P} = \begin{bmatrix} I & P_{12} & P_{13} & \ldots & P_{1n} \\ P_{21} & I & P_{23} & \ldots & P_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ P_{n1} & P_{n2} & P_{n3} & \ldots & I \end{bmatrix}. \tag{3.9}$$

Safavi & Bento [32] prove that the alignment consistency is equivalent to $\boldsymbol{P} \succeq 0$ (see Lemma 4 in Safavi & Bento [32]). By relaxing the permutation matrices constraint in (3.7) to a doubly stochastic constraint, the G-align distance function is as follows:

$$d_G(A_1, \ldots, A_n) = \min_{\substack{P_{ij}\in\mathcal{W}^m, \\ P_{ii}=I, \boldsymbol{P}\succeq 0}} \frac{1}{2} \sum_{i,j\in[n]} G(P_{ij}; A_i, A_j). \tag{3.10}$$

This is a pseudo n-metric (see Theorem 5 and Remark 4 in Safavi and Bento [32]) and a convex optimization problem with $O(n^2m^2)$ variables and n constraints. In practice, this problem can be solved via optimization toolboxes such as CVXPY [40] as well as the Frank-Wolfe algorithm (FW) [41]; the latter is outlined in App. B.

Despite convexity, the quadratic nature of $\boldsymbol{P}$ (in terms of n and m) in G-align distance and $P_i \in \mathcal{W}^m$, $i \in [n]$ (in terms of m) in Fermat distance makes these computations expensive. We address this in Section 5.

3.3 Graph Generative Models. Given a set of undirected graphs $\boldsymbol{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_n\}$ sampled from $p(G)$, for each graph $\mathcal{G}_i(V_i, E_i)$, $\forall i \in [n]$, we denote the adjacency matrices by $A_i \in \mathbb{R}^{m\times m}$ and feature matrices by $X_i \in \mathbb{R}^{m\times f}$. Features could be either one-hot node indicator vectors or consist of graph characteristics, such as, e.g., node degrees. The nodes of each graph are mapped to a latent embedding space via a deep neural network, parameterized by ϕ [20]:

$$Z_i = f_\phi(A_i, X_i), \tag{3.11}$$

where $Z_i = \{z_{i_1}, z_{i_2}, \cdots, z_{i_m}\}$ denotes the hidden node representations and ϕ represents parameters of the deep neural network encoder. The decoder parameterized by θ takes the hidden representations and reconstructs the adjacency matrix, i.e.:

$$\hat{A}_i = g_\theta(Z_i), \tag{3.12}$$

where $\hat{A}_i$ is the estimated adjacency matrix. Both the encoder and decoder can be randomized, and induce a distribution $p_{\phi,\theta}$ over graphs. A loss often used to train parameters over graphs is the negative log likelihood:

$$L(\phi, \theta; \boldsymbol{A}, \boldsymbol{X}) = -\textstyle\sum_{i=1}^{n} \log p_{\phi,\theta}(A_i), \tag{3.13}$$

where $\boldsymbol{A} = \{A_1, A_2, \ldots, A_n\}$ denotes the set of adjacency matrices and $\boldsymbol{X} = \{X_1, X_2, \ldots, X_n\}$ represents the set of feature matrices.

Several existing generative models can be described using the general framework described by Eq. (3.11)-(3.13). GraphRNN [11] is an auto-regressive model that generates node embeddings sequentially: f_ϕ (3.11) is an RNN that encodes the states of graph generated so far, and g_θ (3.12) is a Gated Recurrent Unit (GRU) model that outputs the distribution of the next node's adjacency vector. GRAN [12] is also an auto-regressive model that generates the graph in a block by block basis. In this model, f_ϕ (3.11) is a GRU that uses an attention-weighted sum over the neighborhood of each node to produce the corresponding node embedding and g_θ (3.12) models the probability of generating edges in a block comprising multiple rows of a graph adjacency matrix via a mixture of Bernoulli distributions. In VAE [10], f_ϕ (3.11) is a probabilistic encoding denoted by $q_\phi(Z_i|A_i, X_i)$. There is a prior over the latent variables $p_z(Z_i) \sim N(0, I)$ and g_θ (3.12) is defined as the inner product between latent variables. The loss function (3.13) is further approximated by a variational lower bound of the log-likelihood [42]. All these models can be trained by minimizing the loss (3.13) via standard gradient methods.

4 AlignGraph

We present AlignGraph, our framework for enhancing the permutation invariance of base generative models. AlignGraph can be applied to any base generative model of the form given by Eq. (3.11)- (3.13): we indeed apply it to GraphRNN [11], GRAN [12] and VAE [10] in Sec. 6. We consider three AlignGraph variants, described next.

4.1 G-align-Single. We begin by aligning sampled graphs. To do this, we first compute $\boldsymbol{P}$ by solving the problem in Eq. (3.10) [32]. We take the first block column of $\boldsymbol{P}$, i.e., $\{P_{i1}\}_{i=1}^{n}$, and project each $P_{i1} \in \mathcal{W}^m$ onto the set of permutation matrices, i.e.:

$$\tilde{P_{i1}} = \Pi_{\mathcal{P}^m}(P_{i1}), \tag{4.14}$$

where $\Pi_{\mathcal{P}^m}$ is the orthogonal projection to $\mathcal{P}^m$. This can be done in polynomial time with the Hungarian algorithm [43]. Given these permutation matrices, we align all graphs and features with the first graph, via:

$$\tilde{A}_i = \tilde{P_{i1}}^T A_i \tilde{P_{i1}}, \quad \tilde{X}_i = \tilde{P_{i1}}^T X_i \quad \forall i \in [n]. \tag{4.15}$$

Note that, by alignment consistency (3.8), this could be done on any block column; our selection of $\{P_{i1}\}_{i=1}^{n}$ is arbitrary. Given the alignment matrices $\{\tilde{P_{i1}}\}_{i\in[n]}$, the adjacency matrices $\mathbf{A}$ and feature matrices $\mathbf{X}$, we aim to solve the following optimization problem:

$$\min_{\phi,\theta} \tfrac{1}{n} \textstyle\sum_{i=1}^{n} L(\phi, \theta; \tilde{A}_i, \tilde{X}_i), \tag{4.16}$$

where ϕ and θ are the DNN parameters and $L(\cdot)$ is the loss defined in Eq. (3.11)-(3.13). Eq. (4.16) can be solved via stochastic gradient descent (SGD).

4.2 G-align-Double. In our second approach, we (a) compute a central graph across the graph set and (b) enforce that this graph and aligned graphs are jointly embeddable in the same space. To that end, we use two base generative models combined with the G-align distance function. We again compute $\boldsymbol{P}$ from the G-align distance [32] by solving Eq. (3.10) and project each $P_{i1} \in \mathcal{W}^m$ onto the set of permutation matrices $\{\tilde{P_{i1}}\}_{i\in n}$ via (4.14). We then align the adjacency matrices and feature matrices as in Eq. (4.15). We then estimate the center graph $\mathcal{G}_0$ of set $\boldsymbol{\mathcal{G}}$, i.e. the graph which has the minimum distance from all the graphs in the graph set via:

$$\min_{\hat{A}_{0_{j,k}}, \in [0,1], \forall j,k \in [m]} \textstyle\sum_{i=1}^{n} \|\tilde{A}_i - \hat{A}_0\|. \tag{4.17}$$

Prob. (4.17) is convex and can be solved via standard methods: the objective has n terms, and the problem has m^2 parameters and $O(m^2)$ constraints. Since $\hat{A}_{0_{j,k}} \in [0,1]$, $\forall j,k \in [m]$, we binarize the elements of the adjacency matrix for $\mathcal{G}_0$ by using a threshold. Once we estimate $\mathcal{G}_0$, given the permutation matrices $\{\tilde{P_{i1}}\}_{i\in n}$ and the graph set $\boldsymbol{\mathcal{G}}$ with one-hot encoding feature matrices $\mathbf{X}$, we train two generative models of the chosen type. We train the first with $\mathcal{G}_0$ and the second with all aligned $\{\mathcal{G}_i\}_{i=1}^{n}$. We train the generative models jointly, by penalizing the distance between the embeddings of $\mathcal{G}_i$ and $\mathcal{G}_0$, i.e.:

$$\begin{aligned}(4.18)\quad \min_{\Phi,\Theta}\ & \tfrac{1}{n}\textstyle\sum_{i=1}^{n}[L(\phi,\theta;\tilde{A}_i,\tilde{X}_i)+\beta\,\mathrm{tr}(D(\tilde{Z}_i,Z_0))]\\ & + L(\phi_0,\theta_0,A_0,X_0),\end{aligned}$$

where $\beta > 0$ is a positive regularization parameter, $L(\cdot)$ is a loss function of a base generative model defined in Eq. (3.11)-(3.13), $\Phi = \{\phi_0, \phi\}$ and $\Theta = \{\theta_0, \theta\}$ are the generative models parameters, $Z_0, \{Z_i\}_{i\in[n]} \in \mathbb{R}^{m\times d}$ are the hidden representation of nodes and D is given by Eq. (3.4b). Note that the trace enforces the joint embeddability of all graphs with the central graph. The objective in Eq. (4.18) again can be minimized via SGD. After training we take *only* the generative model parameterized by ϕ, θ to generate new graphs.

4.3 Fermat-Double. In this model, we combine the Fermat distance function with two similar-structure generative models. We first use the Fermat distance function defined in Eq. (3.5) to estimate graph alignment matrices $\{P_i\}_{i\in[n]}$ and $\mathcal{G}_0$ via alternating minimization. Then, we project each $P_i \in \mathcal{W}^m$ onto the set of permutation matrices $\{\tilde{P}_i\}_{i\in n}$ via Eq. (4.14). Given the graph set $\boldsymbol{\mathcal{G}}$, the center graph $\mathcal{G}_0$ and alignment matrices $\{\tilde{P}_i\}_{i\in[n]}$, we train the two generative models jointly. We train the first with $\mathcal{G}_0$ and the second with the aligned $\{\mathcal{G}_i\}_{i\in[n]}$. We minimize the distance between the embeddings of these two generative models by solving the following optimization problem:

$$\begin{aligned}(4.19)\quad \min_{\Phi,\Theta}\ & \tfrac{1}{n}\textstyle\sum_{i=1}^{n}[L(\phi,\theta;\tilde{A}_i,\tilde{X}_i)+\beta\,\mathrm{tr}(D(\tilde{Z}_i,Z_0))]\\ & + L(\phi_0,\theta_0,A_0,X_0),\end{aligned}$$

where $\beta > 0$ is a positive regularization parameter, $L(\cdot)$ is again a loss function of a base generative model defined in Eq. (3.11)-(3.13), $\Phi = \{\phi_0, \phi\}$ and $\Theta = \{\theta_0, \theta\}$ are the generative models parameters, $Z_0, \{Z_i\}_{i\in[n]} \in \mathbb{R}^{m\times d}$ are graph embeddings and D is given in Eq. (3.4b). We again solve Eq. (4.19) w.r.t. Φ and Θ via SGD. After training, we again use only the generative model parameterized by ϕ and θ to generate graphs.

4.4 Extensions. Our proposed graph alignment methods are not limited to graphs with equal numbers of nodes; they can be readily extended to collections of graphs with a variable number of nodes by employing one of several ways to add "dummy" nodes such that all graphs have equal number of nodes [25]. A simple solution is to first find the maximum number of nodes $\mathrm{m}_{\max}$ in the graph set and then expand all graphs with $|V_i| < \mathrm{m}_{\max}$, $i \in [n]$ by adding "dummy" nodes such that all graphs have $\mathrm{m}_{\max}$ nodes. In the expanded graphs "dummy" nodes are connected to each other as well the actual nodes by edges with a small weight (e.g., 0.01) to differentiate these edges from the edges connecting the actual nodes.

5 Accelerated Multi-Distances.

In both Fermat distance and G-align distance, as the number n of graphs grows, alignment becomes more computationally expensive. We propose three methods to accelerate multi-distance algorithms. All methods produce a final center graph, $\mathcal{G}_{0_{\mathrm{out}}}$; once this is computed, all the graphs in $\boldsymbol{\mathcal{G}}$ can be aligned with $\mathcal{G}_{0_{\mathrm{out}}}$ (and each other) via Eq. (3.2). We describe these methods assuming alignment happens via the G-align distance, but the methods extend, mutatis mutandis, to Fermat distance as well, by replacing Eq. 3.10 with Eq. 3.5. We provide pseudocode for all three methods in App. E.

G-Parallel: Grouping and Parallelizing Graphs. This method has a recursive structure, comprising $O(\log_K n)$ stages, where $K \in \mathbb{N}$. In each stage, we apply the same three-step procedure on a smaller set of graphs, starting from the full set of graphs in the training set. In the first step, we divide the set of graphs into a collection of smaller groupings of size $K \ll n$. In the second step, we compute the alignment via Eq. (3.10) *within each group.* In the third step, we output a center graph, computed via Eq. (4.17), for each group. Note that the operations in the second and third steps can happen in parallel. The procedure then executes recursively on the (smaller) set of center graphs. The output of the final stage is a single center graph, $\mathcal{G}_{0_{\mathrm{out}}}$. We note that, for Eq. (3.5), Eq. (3.10), and Eq. (4.17), computing alignments over $K \ll n$ rather than n graphs yields significant performance dividends even serially, because the execution cost is super-quadratic in the number of graphs. The total number of such K-graph problems we compute is $O(\frac{n}{K})$.

C-Serial: Coarsening Graphs. In this method, we create coarsened graphs [33] by partitioning each graph into $c \in \mathbb{N}$ clusters via clustering algorithm such as K-means. In short, the nodes in a coarsened graph are super-nodes representing all nodes in the original graphs' clusters. The weighted edges are the unions of edges connecting two clusters in the original graph. We next compute the graph alignment across the *coarsened graphs*, via Eq. (3.10). Having mapped clusters to each other across graphs, we refine alignments: we align the nodes within the clusters via Eq. (3.10) on a per-cluster basis. This yields a global alignment; finally, we construct a center graph by computing the center for the clusters and the edges connecting the clusters via Eq. (4.17). In this method, we need to compute

	$\lvert V\rvert_{\mathrm{ave}}$	$\lvert E\rvert_{\mathrm{ave}}$	n	Alignment alg.
Community (small)	45	98	100	G-Parallel
Community (large)	150	2727	100	CG-Parallel
Grid	36	265	100	G-Parallel
Ego-Citeseer	35	65	100	G-Parallel
Ego-B-A (small)	118	298	100	CG-Parallel
Ego-B-A (large)	1028	1471	68	CG-Parallel
Protein	117	280	100	CG-Parallel

Table 1: Dataset summary including average number of nodes and edges and number of graphs in the graph set, along with the algorithm used to compute graph alignment. For smaller graphs (with $|V|_{\mathrm{ave}} < 50$) we use the G-Parallel method. For larger graphs, to further accelerate computing the graph alignment, we use CG-Parallel. For all parallel alignment algorithms we use a single machine with 40 CPUs.

distances over $O(n)$ graphs again but of size $O(c)$, with the refinement involving nc pairwise alignments of size, approximately, m/c, assuming clusters of equal size.

CG-Parallel: Coarsening, Grouping and Parallelizing. Similar to G-Parallel, this method is recursive and in each stage we apply the same procedure on a smaller set of graph. We just change what happens in each stage compared to G-Parallel. Again, similar to G-Parallel, in each stage we first divide graphs into smaller groupings. In each of these smaller groups, we compute the center graphs exactly the same way we did in C-serial, i.e., by coarsening graphs, computing the alignments via Eq. (3.10), computing the center graph by computing the center of clusters and edges connecting clusters via Eq. (4.17). The procedure then executes recursively on the (smaller) set of center graphs. The output of the final stage is a center graph, $\mathcal{G}_{0_{\mathrm{out}}}$, for the whole set. The total number of stages in this method is $O(\log_K n)$. The total number of such K-graph problems we compute is $O(\frac{n}{K})$ with the refinement involving Kc pairwise alignments of size, approximately, m/c, assuming clusters of equal size.

6 Experimental Setup

6.1 Datasets. We perform experiments on both synthetic and real datasets with varying numbers of nodes and edges, using the code in [11].

Community. We generate two community graphs, with three-communities from the stochastic block model [11]. The first graph has $|V| = 45$ total nodes and $[5, 15, 17]$ nodes in the communities. The second has $|V| = 150$ total nodes and $[40, 50, 60]$ nodes in the communities. In both graphs, each community is generated by the Erdős-Rényi model (E-R) [6]. The probability for edge creation in each community is $p = 0.7$. For the smaller graph $0.05|V|$ inter-community edges were added and for the large community graph $0.005|V|$ inter-community edges were added u.a.r. In order to build the graph set, we generate 100 random graphs by randomly permuting the graph and then add noise by randomly removing and re-adding 10% of edges, selected u.a.r.

Grid. We construct a 2-D grid graph with $|V| = 36$ nodes. As above, we generate 100 graphs by randomly permuting the graph and again add noise by randomly removing and re-adding 10% of edges, u.a.r.

Ego-B-A (small). We generate 100 graphs with $|V| = 950$ nodes using the Barabási-Albert model. During the generation of each graph, each node in a graph is connected to 5 existing nodes. We then construct $1-$hop ego graphs with $|V| \in [100 - 130]$ nodes.

Ego-B-A (large). We generate 68 graphs using the Barabási-Albert model. Each graph has $|V| = 75500$ nodes such that each node is connected to 5 existing nodes during generation. In the next step, we construct $1-$hop ego graphs with $|V| \in [1000 - 1050]$ nodes.

Ego-Citeseer. Similar to [11, 44], we construct 100 3-hop ego graphs from the Citeseer network [45], with $|V| \in [30 - 40]$ nodes.

Protein. Similar to [11, 12], we select 100 protein graphs from a protein dataset [46] with $|V| \in [100, 130]$ nodes. The nodes in these graphs represent amino acids and the edges are placed between all pairs of nodes that are less than 6 Angstroms apart.

Table 1 summarizes each dataset as well as graph set size and the methods used to compute graph alignments. In all datasets, we use CVXPY [40] as our solver; additional implementation details can be found in App. E.

6.2 Algorithms. We compare our methods against three base generative models, GraphRNN [11], GRAN [12], and VAE [10] and two competitors, GraphVAE [14] and DeepGMG [47]. Additional details on baseline algorithms are in App. F. We compare these baselines to all three versions of AlignGraph described in Sec. 4, where for each of our algorithms we test with three base generative models (GraphRNN [11], GRAN [12], VAE [10]). Our code is publicly available.[1]

6.3 Performance Metrics . In all experiments we take 80% of the full set of graphs for training and use the rest for testing. We train our generative models on the training set, and use them to generate a set of synthetic graphs, whose properties we then compare to graphs in the test set to evaluate whether the generated graphs are likely to have come from the same distribution as the test set. We use two performance metrics to assess the quality of the generated graphs. In both metrics, we first calculate a set of summary statistics from each individual graph (e.g., degree distribution, clustering co-

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

efficient, etc.); we summarise these statistics in App. C. Then we compare the distributions of these statistics between the generated and test graphs w.r.t. two metrics. The first is the $\mathrm{s}_{\mathrm{mmd}}$ score: this score, proposed by You et al. [11], measures the maximum mean discrepency (MMD) between two distributions of graph statistics. The s_{mmd} takes values in $[0, 1]$ (the smaller the better). We calculate an average MMD across all the statistics; a formal definition can be found in App. D.

The second performance score is the $\mathrm{s}_{\mathrm{mvr}}$ score: this measures the squared difference between the mean values of the two distributions, rescaled by the variance of the value over the ground truth graphs. This score takes values in $[0, \infty]$ (the smaller the better). Again, we average this across all statistics (see also App. D).

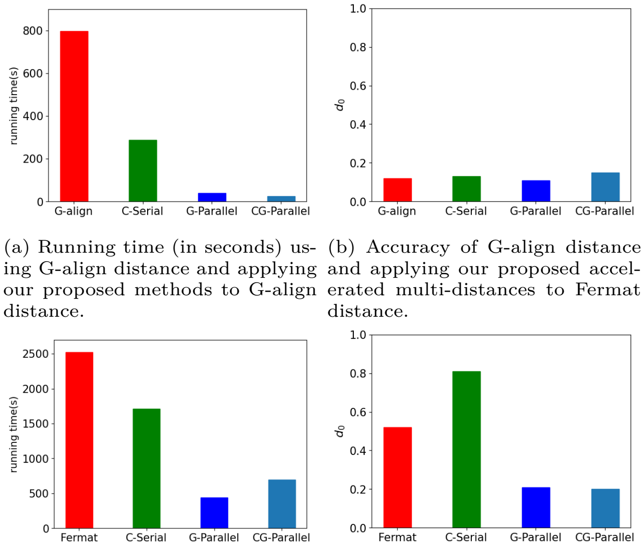
We also report the time it took to compute graph alignments, t_a, and the total training time of generative models, t_{tr}. We compute graph alignment only once and pre-align graphs before training our AlignGraph models. We measure t_a and t_{tr} to have a fair comparison between the improvement we might get in $\mathrm{s}_{\mathrm{mmd}}$ and $\mathrm{s}_{\mathrm{mvr}}$ and the cost of this improvement in terms of the total time consumed by each model.

To evaluate the performance of our accelerated multi-distances, we measure the accuracy of alignments. For this purpose, we first compute graph alignment and the center graph via Eq. (3.5) for Fermat distance and Eq. (3.10) and Eq. (4.17) for G-align distance. We then align the graphs in the graph set w.r.t $\mathcal{G}_0$ and evaluate the distance of $\mathcal{G}_0$ from the graph set via $d_0 = \frac{1}{n}\sum_{i=1}^{n} \frac{\|P_i^T A_i P_i - A_0\|}{\|A_1\|}$ (smaller is better).

6.4 Results

Accelerated Multi-distances Speed and Accuracy. We investigate the impact of our methods on running time and on the accuracy of graph alignment computation on a graph set with 12 3−community graphs of $|V| = 45$ nodes. In Fig. 1a and Fig. 1c we report the total time to compute the alignment using G-align distance and Fermat distance, respectively. These figures demonstrate that our proposed methods reduces the computation time by 40 times. In Fig. 1b and Fig. 1d we compute d_0 via Eq. 6.3. Our results illustrate that our acceleration methods improve the accuracy of estimated center graphs. Since G-Parallel and CG-Parallel have the best trade offs for the running time and accuracy, we use these two methods to compute graph alignment in our next experiments.

Evaluating the Generated Graphs. Table 2 summarizes the performance scores s_{mmd} and s_{mvr} on all 7 datasets. Our experiments show that our model achieves $25\% - 250\%$ accuracy improvement over base models and $62.5\%-4000\%$ improvement over other competitors. In some datasets, such as Community graphs and Protein graphs, G-align-Double and Fermat-Double that jointly train two similar structure generative models produce the best performance scores. In the majority of the experiments, applying our frameworks to either base GraphRNN or base GRAN leads to the best performance scores. However, there is no clear winner between these two base generative models. Our results in Table 2 illustrate that our accelerated multi-distances methods scale well to larger graphs and are compatible with large datasets with $|V| > 1000$. Moreover, comparing the t_{tr} of G-align-Single (GraphRNN) and G-align-Single (GRAN) models with their baselines demonstrate that our models are $4.21\% - 44\%$ faster. This happens due to the pre-alignment of graphs in our models. On the other hand, the t_a/t_{rm} ratio for G-align-Single (GraphRNN) and G-align-Single (GRAN) models ranges from 0.89% to 150%, where 150% belongs to the alignment of our largest dataset, Ego-B-A (large). While this pre-alignment took 70 minutes, it led to at least 83% improvement in the performance scores.



(a) Running time (in seconds) using G-align distance and applying our proposed methods to G-align distance.

(b) Accuracy of G-align distance and applying our proposed accelerated multi-distances to Fermat distance.

(c) Same as part (a) but for Fermat distance.

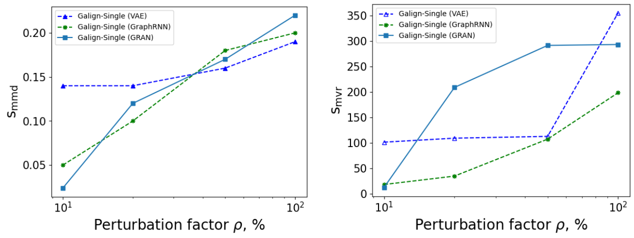
(d) Same as part (b) but for Fermat distance.

Figure 1: Computation time and accuracy of computing the graph alignment in community graphs given the baselines and our three accelerated multi-distances, G-Parallel (40 CPUs), CG-Parallel (40 CPUs), and C-Serial. G-align distance has better performance compared to Fermat distance. Moreover, due to the clustered structure of community graphs clustering and grouping graphs in CG-Parallel also improves the accuracy.

Impact of Graph Perturbation. We investigate the impact of graph perturbation on the performance of our models by perturbing edges in the 3-community graphs dataset with $|V| = 45$. The perturbation

	Community Graphs								Grid Graphs				Ego – Citeseer Graphs				Ego – B – A Graphs								Protein Graphs			
	$(\lvert V\rvert_{\text{ave}},\ \lvert E\rvert_{\text{ave}})$ $(45, 98)$				$(\lvert V\rvert_{\text{ave}},\ \lvert E\rvert_{\text{ave}})$ $(150, 2727)$				$(\lvert V\rvert_{\text{ave}},\ \lvert E\rvert_{\text{ave}})$ $(36, 265)$				$(\lvert V\rvert_{\text{ave}},\ \lvert E\rvert_{\text{ave}})$ $(35, 65)$				$(\lvert V\rvert_{\text{ave}},\ \lvert E\rvert_{\text{ave}})$ $(118, 298)$				$(\lvert V\rvert_{\text{ave}},\ \lvert E\rvert_{\text{ave}})$ $(1028, 1471)$				$(\lvert V\rvert_{\text{ave}},\ \lvert E\rvert_{\text{ave}})$ $(117, 280)$			
	$\mathrm{s_{mmd}}$	$\mathrm{s_{mvr}}$	$\mathrm{t_{tr}}$(min)	$\mathrm{t_a}$(min)	$\mathrm{s_{mmd}}$	$\mathrm{s_{mvr}}$	$\mathrm{t_{tr}}$(min)	$\mathrm{t_a}$(min)	$\mathrm{s_{mmd}}$	$\mathrm{s_{mvr}}$	$\mathrm{t_{tr}}$(min)	$\mathrm{t_a}$(min)	$\mathrm{s_{mmd}}$	$\mathrm{s_{mvr}}$	$\mathrm{t_{tr}}$(min)	$\mathrm{t_a}$(min)	$\mathrm{s_{mmd}}$	$\mathrm{s_{mvr}}$	$\mathrm{t_{tr}}$(min)	$\mathrm{t_a}$(min)	$\mathrm{s_{mmd}}$	$\mathrm{s_{mvr}}$	$\mathrm{t_{tr}}$(min)	$\mathrm{t_a}$(min)	$\mathrm{s_{mmd}}$	$\mathrm{s_{mvr}}$	$\mathrm{t_{tr}}$(min)	$\mathrm{t_a}$(min)
GraphVAE	0.20	612.17	5229.60	0	–	–	–	–	0.13	13.39	3827.79	0	0.04	0.66	4084.49	0	–	–	–	–	–	–	–	–	–	–	–	–
DeepGMG	0.15	1180.66	2771.26	0	–	–	–	–	0.18	7.16	2771.41	0	0.01	0.92	2771.47	0	–	–	–	–	–	–	–	–	–	–	–	–
VAE	0.18	895.42	0.13	0	0.22	6475.40	0.86	0	0.24	66.09	0.06	0	0.06	24.22	0.07	0	0.16	375.55	0.18	0	0.25	22110.65	26.64	0	0.12	8.05	19.40	0
GraphRNN	0.08	50.11	45.30	0	0.14	1270.27	213.16	0	0.10	11.21	45.73	0	0.005	0.19	52.45	0	0.018	2.78	75.86	0	–	–	–	–	0.06	1.28	75.19	0
GRAN	0.017	26.98	77.33	0	0.16	4611223.41	525.76	0	0.12	27.64	76.69	0	0.009	0.70	76.17	0	0.011	0.60	19.54	0	0.011	31.26	50.44	0	0.07	1.61	22.03	0
G-align-Single (VAE)	0.16	357.20	9.72	25.5	0.19	4983.63	72.70	50.41	0.20	12.52	5.04	5.08	0.02	5.37	8.32	8.86	0.10	63.56	16.76	2.99	0.24	22131.28	70.45	70.36	0.15	4.47	22.31	4.7
G-align-Double (VAE)	0.09	**8.76**	9.12	25.5	0.16	488.93	126.35	50.41	0.17	39.72	6.97	5.08	0.04	2.81	8.41	8.86	0.03	38.08	108.29	2.99	–	–	–	–	**0.03**	**0.90**	137.26	4.7
Fermat-Double (VAE)	0.09	1543.08	8.84	6.33	0.18	138.39	77.90	77.90	0.18	45.19	5.70	3.18	0.02	13.37	7.21	5.69	0.07	380.99	95.44	125.4	–	–	–	–	0.04	0.93	85.30	121.81
G-align-Single (GraphRNN)	0.04	92.49	41.42	25.5	0.12	1268.49	211.58	50.41	0.09	**6.72**	46.86	5.08	**0.002**	0.12	41.10	8.86	**0.007**	1.05	76.50	2.99	–	–	–	–	0.07	1.74	67.10	4.7
G-align-Double (GraphRNN)	0.06	116.28	190.87	25.5	0.13	796.20	1919.36	50.41	0.12	8.04	140.63	5.08	**0.002**	0.08	204.85	8.86	0.009	0.97	1266.28	2.99	–	–	–	–	0.05	1.26	1251.21	4.7
Fermat-Double (GraphRNN)	0.05	40.41	152.45	6.33	0.13	764.67	1816.18	77.90	0.19	12.55	105.59	3.18	0.004	**0.05**	133.41	5.69	**0.007**	0.85	1250.64	125.4	–	–	–	–	0.04	1.55	1277.53	121.81
G-align-Single (GRAN)	**0.012**	24.75	73.57	25.5	0.12	27974.71	473.26	50.41	**0.08**	12.81	53.0	5.08	0.005	0.20	47.74	8.86	0.04	6.60	18.75	2.99	**0.006**	**6.86**	45.56	70.36	0.13	3.68	19.92	4.7
G-align-Double (GRAN)	0.14	1171.44	175.35	25.5	0.19	15481.15	962.48	50.41	0.13	45.77	191.80	5.08	0.008	6.90	223.56	8.86	0.008	**0.56**	846.33	2.99	–	–	–	–	0.07	2.88	843.62	4.7
Fermat-Double (GRAN)	0.13	928.87	166.14	6.33	**0.04**	**40.6**	1039.26	77.90	0.20	46.03	198.68	3.18	0.03	24.55	206.43	5.69	0.11	9.32	785.09	125.4	–	–	–	–	0.13	178.12	835.23	121.81

Table 2: Comparison of two performance scores for synthetic and real graphs graphs. $|V|_{\text{ave}}$ is the average number nodes and $|E|_{\text{ave}}$ is the average number of edges in the graph set. $\mathrm{t_{tr}}$ indicates the total time to train generative models and $\mathrm{t_a}$ is the total time to compute graph alignment using either G-align distance or Fermat distance. $(-)$ indicates an out of memory failure. Overall, applying our frameworks to base RNN or base GRAN leads to better performance scores compared to baselines.



(a)

(b) .

Figure 2: Sensitivity of 3 base generative models combined with G-align-Single to noise for community graphs with $|V| = 45$. x-axis: the percentage of edges perturbed, y-axis : $\mathrm{s_{mmd}}$ (left), $\mathrm{s_{mvr}}$ (right). G-align-Single (GraphRNN) shows better overall performance, however, in all models there is a direct relation between the perturbation factor and the performance drop.

factor ρ is defined as the percentage of edges that we randomly remove and re-add u.a.r. The ρ values are set to $[10, 20, 50, 100]$ in our experiments. We note that with $\rho < 20\%$, graphs still have community structures. In the extreme however, with a $\rho = 100\%$, graphs are effectively Erdös-Rényi and, thus, their statistics differ significantly from those of the test set. We compute $\mathrm{s_{mmd}}$ and $\mathrm{s_{mvr}}$ for graphs generated with these perturbation factors. Fig. 2 illustrates the performance of the G-align-Single model using GraphRNN, GRAN and VAE as base generative models. G-align-Single (GraphRNN) and G-align-Single (GRAN) models have relatively good $\mathrm{s_{mmd}}$ compared to G-align-Single (VAE) when $\rho < 50\%$. At $\rho = 10\%$, G-align-Single (GRAN) has the best performance which is exactly inline with the results we have in Table 2. As the noise increases, G-align-Single (GraphRNN) shows more robustness to noise compared to the other two models. As expected, all models are adversely affected when $\rho > 50\%$.

7 Conclusion

We present a group of models that learn distributions of graphs. Our method is generic with respect to the generative model employed, performs better than the competitors, and enhances permutation invariant and robustness to noise.

Acknowledgments

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

References

[1] K. Do, T. Tran, and S. Venkatesh, "Graph transformation policy network for chemical reaction prediction," in *KDD*, 2019.

[2] Y. Li, L. Zhang, and Z. Liu, "Multi-objective de novo drug design with conditional graph generative model," *Journal of cheminformatics*, 2018.

[3] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: an approach to modeling networks." *Journal of Machine Learning Research*, 2010.

[4] M. Kim and J. Leskovec, "Modeling Social Networks with Node Attributes using the Multiplicative Attribute Graph Model," in *UAI*, 2011.

[5] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, 1999.

[6] P. Erdös and A. Rényi, "On random graphs I," *Publ. Math. Debrecen*, 1959.

[7] T. A. Snijders and K. Nowicki, "Estimation and prediction for stochastic blockmodels for graphs with latent block structure," *Journal of Classification*, 1997.

[8] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, 1998.

[9] A. Fronczak, J. A. Hołyst, M. Jedynak, and J. Sienkiewicz, "Higher order clustering coefficients in

Barabási–Albert networks," *Physica A: Statistical Mechanics and its Applications*, 2002.

[10] T. N. Kipf and M. Welling, "Variational graph autoencoders," *CoRR*, vol. abs/1611.07308, 2016.

[11] J. You, R. Ying, X. Ren, W. Hamilton, and J. Leskovec, "GraphRNN: Generating realistic graphs with deep auto-regressive models," in *ICML*, 2018.

[12] R. Liao, Y. Li, Y. Song, S. Wang, W. Hamilton, D. K. Duvenaud, R. Urtasun, and R. Zemel, "Efficient Graph Generation with Graph Recurrent Attention Networks," *NeurIPS*, 2019.

[13] A. Gritsenko, Y. Guo, K. Shayestehfard, A. Moharrer, J. Dy, and S. Ioannidis, "Graph Transfer Learning," in *ICDM*, 2021.

[14] M. Simonovsky and N. Komodakis, "GraphVAE: Towards generation of small graphs using variational autoencoders," in *ICANN*, 2018.

[15] B. Samanta, A. De, G. Jana, V. Gómez, P. Chattaraj, N. Ganguly, and M. Gomez-Rodriguez, "NEVAE: A deep generative model for molecular graphs," *JMLR*, 2020.

[16] A. Bojchevski, O. Shchur, D. Zügner, and S. Günnemann, "NetGAN: Generating graphs via random walks," in *ICML*, 2018.

[17] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *KDD*, 2016.

[18] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[19] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *KDD*, 2014.

[20] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.

[21] M. R. Garey and D. S. Johnson, "Computers and intractability, vol. 29," 2002.

[22] A. Fischer, C. Y. Suen, V. Frinken, K. Riesen, and H. Bunke, "Approximation of graph edit distance based on hausdorff matching," *Pattern Recognition*, 2015.

[23] H. Bunke and K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognition Letters*, 1998.

[24] H. Bunke, "On a relation between graph edit distance and maximum common subgraph," *Pattern Recognition Letters*, 1997.

[25] J. Bento and S. Ioannidis, "A family of tractable graph distances," in *SDM*, 2018.

[26] A. Nguyen, M. Ben-Chen, K. Welnicka, Y. Ye, and L. Guibas, "An optimization approach to improving collections of shape maps," in *Computer Graphics Forum*, 2011.

[27] Q.-X. Huang and L. Guibas, "Consistent shape maps via semidefinite programming," in *Computer Graphics Forum*, 2013.

[28] Y. Chen, L. Guibas, and Q. Huang, "Near-optimal joint object matching via convex relaxation," in *ICML*, 2014.

[29] X. Zhou, M. Zhu, and K. Daniilidis, "Multi-image matching via fast alternating minimization," in *ICCV*, 2015.

[30] H. Xu, D. Luo, H. Zha, and L. C. Duke, "Gromov-wasserstein learning for graph matching and node embedding," in *ICML*. PMLR, 2019, pp. 6932–6941.

[31] G. Kiss, J.-L. Marichal, and B. Teheux, "A generalization of the concept of distance based on the simplex inequality," *Contributions to Algebra and Geometry*, 2018.

[32] S. Safavi and J. Bento, "Tractable n-Metrics for Multiple Graphs," in *ICML*, 2019.

[33] G. Karypis, "Metis: Unstructured graph partitioning and sparse matrix ordering system," *Technical report*, 1997.

[34] V. Satuluri and S. Parthasarathy, "Scalable graph clustering using stochastic flows: applications to community discovery," in *ACM SIGKDD*, 2009.

[35] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE TPAMI / PAMI*, 2007.

[36] J. Liang, S. Gurukar, and S. Parthasarathy, "Mile: A multi-level framework for scalable graph embedding," in *ICWSM*, 2021.

[37] J. Zhu, D. Koutra, and M. Heimann, "Caper: Coarsen, align, project, refine-a general multilevel framework for network alignment," in *CIKM*, 2022, pp. 4747–4751.

[38] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE SMC*, 1983.

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

[40] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *JMLR*, 2016.

[41] M. Frank, P. Wolfe *et al.*, "An algorithm for quadratic programming," *Naval Research Logistics Quarterly*, 1956.

[42] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *ICLR*, 2014.

[43] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, 1955.

[44] C. Tran, W.-Y. Shin, A. Spitz, and M. Gertz, "Deepnc: Deep generative network completion," *IEEE TPAMI / PAMI*, 2020.

[45] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI magazine*, 2008.

[46] P. D. Dobson and A. J. Doig, "Distinguishing enzyme structures from non-enzymes without alignments," *Journal of molecular biology*, 2003.

[47] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. W. Battaglia, "Learning deep generative models of graphs," *CoRR*, vol. abs/1803.03324, 2018.

[48] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization.* Cambridge University Press, 2004.

[49] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *JMLR*, 2011.

A Alternating Minimization.

At each iteration $t \in \mathbb{N}$, we update A_0 and $\{P_i\}_{i\in[n]}$ as follows:

A.1 Updating A_0.

Given that $\{P_i\}_{i\in[n]}$ is fixed and $D = 0$, minimizing Eq. (3.5) w.r.t A_0 leads to the following problem:

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

This problem is convex and at step $t \in \mathbb{N}$ can be solved via convex optimization. Once we solve this optimization problem, we set a threshold to binarize the elements of A_0.

A.2 Updating $\{P_i\}_{i\in[n]}$.

Given that $A0$ is fixed and $D = 0$, let $L_P(\{P_i\}^{(t)}_{i\in[n]})$ be the loss function at step $t \in \mathbb{N}$.

$$L_P(\{P_i\}^{(t)}_{i\in[n]})) = \textstyle\sum_{i=1}^{n} \|A_i P_i^{(t)} - P_i^{(t)} A_0^{(t)}\| \tag{A.2}$$

Minimizing Eq. (3.5) w.r.t $\{P_i\}_{i\in[n]}$ leads to the following problem.

$$\min_{P_i\in\mathcal{W}^m} L_P(\{P_i\}^{(t)}_{i\in[n]}) \tag{A.3}$$

This step is convex. It can be solved via optimization toolboxes such as CVXPY [40] or efficient algorithms such as Frank-Wolfe algorithm [41]. Frank-Wolfe algorithm is explained in details in the Section B.

B Frank Wolfe.

The objective function in Eq. (A.2) can be solved via Frank-Wolfe algorithm. Frank-Wolfe is an iterative algorithm that solves the problem through a sequence of linear programs (LPs). This algorithm starts from a feasible $P^0 \in \mathcal{W}^m$, e.g. , the identity matrix I and in each iteration $t \in \mathbb{N}$ proceeds as follows:

$$S^{(t)} = \arg\min_{S_{ij}\in\mathcal{W}^m, S_{ii}=I, S\succeq 0} \operatorname{tr}(S^T, \nabla_P L_P(P^{(t)})) \tag{B.4a}$$

$$P^{(t+1)} = (1-\gamma_t)P^{(t)} + \gamma_t S^{(t)}, \tag{B.4b}$$

where γ_t is the step size and can be set to e.g. $\frac{2}{t+2}$ or determined by line search [48] as follows:

$$\gamma_t = \arg\min_{\gamma_t\in[0,1]} L_P(((1-\gamma_t)P^{(t)} + \gamma_t S^{(t)}) \tag{B.5}$$

C Table of metrics.

In the Table 3 we provide the lists of metrics we measured in the experiments and their description.

Notation	Description
D.D	Graphs degree distribution
C.C	distribution of clustering coefficient of nodes for each graph in the graph set
ASRT:	assortativity, Pearson correlation coefficient of degree between pairs of linked nodes
TRI:	number of triangles for each graph in the graph set
WG.C:	wedge count, number of wedges for each graph in the graph set
CL.C:	claw count, number of claws for each graph in the graph set

Table 3: Summary of metrics.

D Performance scores.

In order to calculate MMD^2 , let a function f belong to a unit ball in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, $f \in \mathcal{H}$, and k be the kernel. The MMD^2 between two sets of samples $\{x_i\}_{i=1}^{N} \sim^{\text{iid}} p$ and $\{y_i\}_{i=1}^{N} \sim^{\text{iid}} q$ from distributions p and q is computed as follows:

(D.6)

$$\begin{aligned}\text{MMD}^2 = \\ &\frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j\neq i}^{N} (k(x_i, x_j) + k(y_i, y_j)) \\ &- \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (k(x_i, y_j) + k(x_j, y_i))\end{aligned}$$

The performance of MMD^2 depends on choice of the kernel. Here we use Gaussian-Wasserstein RBF kernel $k(x, y) = e^{-\frac{W(p,q)^2}{2\sigma^2}}$, where $W(p, q)$ is the first Wasserstein distance. The $k(x, y)$ function is bounded, $k(x, y) \in [0, 1]$ and therefore MMD$^2 \in [0, 2]$.

s_{mmd} **score.** Combining MMD^2 of all metrics we measured, we present s_{mmd} score to assess the overall quality of generated graphs.

(D.7)

$$\begin{aligned}\text{s}_{\text{mmd}} = \\ &\frac{1}{12}(\text{MMD}^2(\text{D.D}) + \text{MMD}^2(\text{C.C}) + \text{MMD}^2(\text{ASRT}) \\ &+ \text{MMD}^2(\text{TRI}) + \text{MMD}^2(\text{WG.C}) + \text{MMD}^2(\text{CL.C}))\end{aligned}$$

Note that $\text{s}_{\text{mmd}} \in [0, 1]$. The smaller this score, the smaller the distance between the generated graphs and test set.

s_{mvr} **score.** Our second performance score, s_{mvr} is formulated as follows:

Algorithm 1: G-Parallel: Grouping and Parallelizing Graphs

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

$$
\begin{aligned}
\mathrm{s}_{\text{mvr}} = \frac{1}{6}(& \\
& \frac{(\mu_{\text{r}}(D.D) - \mu_{\text{g}}(D.D))^2}{\sigma_{\text{r}}^2(D.D)} \\
& + \frac{(\mu_{\text{r}}(ASRT) - \mu_{\text{g}}(ASRT))^2}{\sigma_{\text{r}}^2(ASRT)} \\
& + \frac{(\mu_{\text{r}}(C.C) - \mu_{\text{g}}(C.C))^2}{\sigma_{\text{r}}^2(C.C)} \\
& + \frac{(\mu_{\text{r}}(TRI) - \mu_{\text{g}}(TRI))^2}{\sigma_{\text{r}}^2(TRI)} \\
& + \frac{(\mu_{\text{r}}(WG.C) - \mu_{\text{g}}(WG.C))^2}{\sigma_{\text{t}}^2(WG.C)} + \\
& \frac{(\mu_{\text{r}}(CL.C) - \mu_{\text{g}}(CL.C))^2}{\sigma_{\text{r}}^2(CL.C)}),
\end{aligned}
\tag{D.8}
$$

where μ_{r}, μ_{g} and σ_{t}^2 represent mean value for the reference set, mean value for the generated set and variance of the reference set, respectively.

E Accelerating Multi-distances

In Alg. 1 we explain how to compute the final center graph by G-Parallel. We describe C-Serial algorithm is in details in Alg. 2 and the detail of CG-Parallel are in Alg. 3. Table 4 shows summary of datasets, acceleration methods and solvers used to compute graph alignment.

Algorithm 2: C-Serial: Coarsening Graphs

```
Input: $\boldsymbol{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$, $c$ : number of clusters in
    graphs.
Output: $\mathcal{G}_{0_{\text{out}}}$ : the center graph.
for $l = \{0, 1, 2, \cdots, n\}$ do
    $\hat{G}_l = \text{coarsen}(G_l)$
end
$\hat{\boldsymbol{\mathcal{G}}} = \text{align}(\{\hat{\mathcal{G}}_1, \hat{\mathcal{G}}_2, \cdots, \hat{\mathcal{G}}_n\})$
  for $l = \{0, 1, 2, \cdots, n\}$ do
    align clusters given their alignment in the coarsened
      graphs.
end
compute center of clusters.
compute center of edges connecting clusters.

$\tilde{\mathcal{G}_0}^k$: build given the center of clusters and center of edges
  connecting clusters.
```

Algorithm 3: CG-Parallel: Coarsening, Grouping and Parallelizing Graphs

```
Input: $\boldsymbol{\mathcal{G}} = \{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_n\}$, $K$ : number of graphs in each
    group, $c$ : number of clusters in graphs.
Output: $\mathcal{G}_{0_{\text{out}}}$ : the center graph.
if $[\frac{n}{K}] > K$ then
    while $[\frac{n}{K}] > K$ do
        for $k = \{0, 1, 2, \cdots, [\frac{n}{K}]\}$ do
            $\tilde{\boldsymbol{\mathcal{G}}}_{\boldsymbol{k}} = \{\mathcal{G}_{1+k\times K}, \mathcal{G}_{2+k\times K}, \cdots, \mathcal{G}_{K+k\times K}\}$
              for $l = \{0, 1, 2, \cdots, K\}$ do
                $\hat{G}_{l+k\times K} = \text{coarsen}(G_{l+k\times K})$
            end
            $\hat{\boldsymbol{\mathcal{G}}}_{\boldsymbol{k}} = \text{align}(\{\hat{\mathcal{G}}_{1+k\times K}, \hat{\mathcal{G}}_{2+k\times K}, \cdots, \hat{\mathcal{G}}_{K+k\times K}\})$
              for $l = \{0, 1, 2, \cdots, K\}$ do
                align clusters given their alignment in the
                  coarsened graphs.
            end
            compute center of clusters.
            compute center of edges connecting clusters.

            $\tilde{\mathcal{G}_0}^k$: build given the center of clusters and center
              of edges connecting clusters.
        end
        $\tilde{\boldsymbol{\mathcal{G}}} = \{\tilde{\mathcal{G}_0}^k,$ for $k = \{0, 1, 2, \cdots, [\frac{N}{K}]\}\}$
    end
end
$\boldsymbol{\mathcal{G}}_{\text{out}} = \tilde{\boldsymbol{\mathcal{G}}}$
$\mathcal{G}_{0_{\text{out}}} = \text{center}(\boldsymbol{\mathcal{G}}_{\text{out}})$
```

	$\|V\|_{\text{ave}}$	$\|E\|_{\text{ave}}$	n	Alignment alg.	Solver (Fermat)	Solver (G-align)
Community (small)	45	98	100	G-Parallel	CVXPY + AM	CVXPY
Community (large)	150	2727	100	CG-Parallel	CVXPY + AM	CVXPY
Grid	36	265	100	G-Parallel	CVXPY + AM	CVXPY
Ego-Citeseer	35	65	100	G-Parallel	CVXPY + AM	CVXPY
Ego-B-A (small)	118	298	100	CG-Parallel	CVXPY + AM	CVXPY
Ego-B-A (large)	1028	1471	68	CG-Parallel	CVXPY + AM	CVXPY

Table 4: Dataset summary including average number of nodes and edges and number of graphs in the graph set, along with the algorithm and solvers used to compute graph alignment in Fermat and G-align distance. For smaller graphs (with $|V|_{\text{ave}} < 50$) we use the G-Parallel method. For larger graphs, to further accelerate computing the graph alignment we use CG-Parallel. By using these acceleration techniques, all alignment problems can be solved by CVXPY.

F Implementation Details

We compared the performance our models against five different deep baseline described below.

GraphRNN. You et al. [11] proposes a framework based on graph neural networks. This model uses a graph-level RNN to add a new node to a node sequence each time step and an edge-level RNN to model the generation process of nodes and edges. The reference code for this model is provided by the authors and we followed their recommendation for setting the hyperparameters.

GRAN. Liao et al. [12] proposes a graph recurrent attention framework. This model uses an attention-based GNN and generates a block of graphs that consists of multiple rows of graph adjacency matrices conditioned on the previously generated blocks of the graph and uses a group canonical node ordering, e.g., DFS and BFS to address node ordering problem.

VAE. Kipf & Welling [10] propose a variational autoencoder that is characterized by a probabilistic inference model that maps observed data to a latent representation, a prior distribution over the latent variables and a probabilistic generative model. We randomly pick a graph with a random node ordering from graph set $\boldsymbol{\mathcal{G}}$ and train a VAE to generate graphs.

GraphVAE. Simonovsky & Komodakis [14] propose a variational autoencoder that outputs a probabilistic fully-connected graph and uses a graph matching algorithm to align graph to the ground truth. GraphVAE outputs a graph with adjacency matrix, node attributes and edge attributes. We adapt it to our problem by using one-hot representations of the features. The encoder is a graph convolutional network and the decoder is a multi-layer perception. We used code for this model from [11] and set the hyperparameters based on recommendations made in [14].

DeepGMG. Li et al. [47] introduce a generative model for graphs that generates graphs in a sequential manner. It generates one node at a time and connects each node to the partial graph already generated by creating edges one by one. We used the implementation in [11] and the hyperparameters were set based on the recommendations made in [47].

We take 80% of graphs for training and the rest for the test sets. During testing, GraphRNN model and GRAN model generate graphs directly. However, the output of the VAE decoder is an adjacency matrix with elements in the range of $[0, 1]$. We binarize the adjacency matrix by applying a threshold, τ. We find τ by comparing two sets of graphs, the ones generated from the VAE and 20% of the graphs in the training set, and computing two scores, which we denote s_{mmd} and s_{mvr} (see Sec. 6.3) to measure the distance between these two sets for a range of values of τ. We chose the value of τ that returns the smallest s_{mmd} as our optimal threshold in testing. In all our AlignGraph models, we pre-compute the graph alignment for all datasets and use the aligned graphs for training the generative models. We use GraphRNN [11], GRAN [12] and VAE [10] as our base generative models. We followed the instructions given in [11] and [12] to set the hyperparameters in GraphRNN [11] and GRAN [12] and for the VAE [10] we used the hyperparameters given in [10] and set the learning rate to 0.001. (We note that VAE[18] here refers to the model proposed by Kipf & Welling and is different from GraphVAE[14] by Simonovsky & Komodakis. In all models, the hidden dimensions $\{Z_i\}_{i\in[n]}$ of small graphs are set to 16. For medium graphs ($|V| \in [100 - 500]$) the hidden dimensions are 64 and the hidden dimensions of the large graphs ($|V| > 500$) are set to 256. In all experiments, The node features are one-hot indicator vectors. The AlignGraph architectures are implemented in Python3 using Tensorflow and Pytorch. We implemented the solution of the constrained optimization problems in Section 4 via CVXPY. We implemented all solvers in Python3. For clustering graphs we use Scikit-learn [49]. All experiments are carried out on a Tesla V100 GPU with 32 GB memory and 5120 cores. G-Parallel and CG-Parallel methods parallelizes the computations using python multiprocessing package. For both of these parallel graph alignment algorithms we use a single machine with 40 CPUs.