Scalable Gromov-Wasserstein Learning for Graph Partitioning and Matching

Hongteng Xu[1,2] **Dixin Luo**[2] **Lawrence Carin**[2]
[1]Infinia ML Inc. [2]Duke University
{hongteng.xu, dixin.luo, lcarin}@duke.edu

Abstract

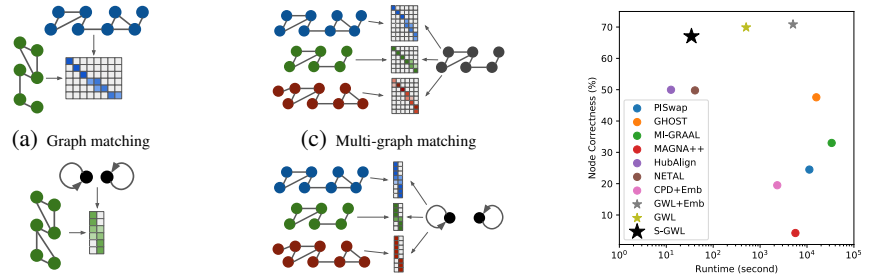
We propose a scalable Gromov-Wasserstein learning (S-GWL) method and establish a novel and theoretically-supported paradigm for large-scale graph analysis. The proposed method is based on the fact that Gromov-Wasserstein discrepancy is a pseudometric on graphs. Given two graphs, the optimal transport associated with their Gromov-Wasserstein discrepancy provides the correspondence between their nodes and achieves graph matching. When one of the graphs has isolated but self-connected nodes ($i.e.$, a disconnected graph), the optimal transport indicates the clustering structure of the other graph and achieves graph partitioning. Using this concept, we extend our method to multi-graph partitioning and matching by learning a Gromov-Wasserstein barycenter graph for multiple observed graphs; the barycenter graph plays the role of the disconnected graph, and since it is learned, so is the clustering. Our method combines a recursive K-partition mechanism with a regularized proximal gradient algorithm, whose time complexity is $\mathcal{O}(K(E+V)\log_K V)$ for graphs with V nodes and E edges. To our knowledge, our method is the first attempt to make Gromov-Wasserstein discrepancy applicable to large-scale graph analysis and unify graph partitioning and matching into the same framework. It outperforms state-of-the-art graph partitioning and matching methods, achieving a trade-off between accuracy and efficiency.

1 Introduction

Gromov-Wasserstein distance [42, 29] was originally designed for metric-measure spaces, which can measure distances between distributions in a relational way, deriving an optimal transport between the samples in distinct spaces. Recently, the work in [11] proved that this distance can be extended to *Gromov-Wasserstein discrepancy* (GW discrepancy) [37], which defines a pseudometric for graphs. Accordingly, the optimal transport between two graphs indicates the correspondence between their nodes. This work theoretically supports the applications of GW discrepancy to structural data analysis, $e.g.$, 2D/3D object matching [30, 28, 8], molecule analysis [43, 44], network alignment [49], etc. Unfortunately, although GW discrepancy-based methods are attractive theoretically, they are often inapplicable to large-scale graphs, because of high computational complexity. Additionally, these methods are designed for two-graph matching, ignoring the potential of GW discrepancy to other tasks, like graph partitioning and multi-graph matching. As a result, the partitioning and the matching of large-scale graphs still typically rely on heuristic methods [16, 12, 45, 27], whose performance is often sub-optimal, especially in noisy cases.

Focusing on the issues above, we design a scalable Gromov-Wasserstein learning (S-GWL) method and establish a new and unified paradigm for large-scale graph partitioning and matching. As illustrated in Figure 1(a), given two graphs, the optimal transport associated with their Gromov-Wasserstein discrepancy provides the correspondence between their nodes. Similarly, graph partitioning corresponds to calculating the Gromov-Wasserstein discrepancy between an observed graph and a disconnected graph, as shown in Figure 1(b). The optimal transport connects each node of the ob-

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.



(a) Graph matching (c) Multi-graph matching

(b) Graph partitioning (d) Multi-graph partitioning (e) Comparisons on accuracy and efficiency

Figure 1: (a)-(d) Illustrations of graph partitioning and matching in the GWL framework. (c, d) The barycenter graph in black and its optimal transports to observed graphs are learned jointly. (d) When the barycenter graph is initialized as a graph with few isolated nodes, the optimal transports indicate aligned partitions of observed graph. (e) We test various graph matching methods in 10 trials on an Intel i7 CPU. In each trial, the source graph has 2,000 nodes and the target graph has 100 more noisy nodes and corresponding edges. The graphs yield either Gaussian partition model [7] or Barabási-Albert model [4]. The GWL-based methods ('$\star$') obtains higher node correctness than other baselines ('$\bullet$'), and our S-GWL (big '$\star$') achieves a trade-off on accuracy and efficiency.

served graph with an isolated node of the disconnected graph, yielding a partitioning. In Figures 1(c) and 1(d), taking advantage of the Gromov-Wasserstein barycenter in [37], we achieve multi-graph matching and partitioning by learning a "barycenter graph". For arbitrary two or more graphs, the correspondence (or the clustering structure) among their nodes can be established indirectly through their optimal transports to the barycenter graph.

The four tasks in Figures 1(a)-1(d) are explicitly unified in our Gromov-Wasserstein learning (GWL) framework, which corresponds to the same GW discrepancy-based optimization problem. To improve its scalability, we introduce a recursive mechanism to the GWL framework, which recursively applies K-way partitioning to decompose large graphs into a set of aligned sub-graph pairs, and then matches each pair of sub-graphs. When calculating GW discrepancy, we design a regularized proximal gradient method, that considers the prior information of nodes and performs updates by solving a series of convex sub-problems. The sparsity of edges further helps us reduce computations. These acceleration strategies yield our S-GWL method: for graphs with V nodes and E edges, its time complexity is $\mathcal{O}(K(E+V)\log_K V)$ and memory complexity is $\mathcal{O}(E+VK)$. To our knowledge, our S-GWL is the first to make GW discrepancy applicable to large-scale graph analysis. Figure 1(e) illustrates the effectiveness of S-GWL on graph matching, with more results presented in Section 5.

2 Graph Analysis Based on Gromov-Wasserstein Learning

Denote a *measure graph* as $G(\mathcal{V}, \boldsymbol{C}, \boldsymbol{\mu})$, where $\mathcal{V} = \{v_i\}_{i=1}^{|\mathcal{V}|}$ is the set of nodes, $\boldsymbol{C} = [c_{ij}] \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$ is the adjacency matrix, and $\boldsymbol{\mu} = [\mu_i] \in \Sigma^{|\mathcal{V}|}$ is a Borel probability measure defined on $\mathcal{V}$. The adjacency matrix is continuous for weighted graph while binary for unweighted graph. In practice, $\boldsymbol{\mu}$ is an empirical distribution of nodes, which can be estimated by a function of node degree. A K-way graph partitioning aims to decompose a graph G into K sub-graphs by clustering its nodes, *i.e.*, $\{G_k = G(\mathcal{V}_k, \boldsymbol{C}_k, \boldsymbol{\mu}_k)\}_{k=1}^K$, where $\cup_k \mathcal{V}_k = \mathcal{V}$ and $\mathcal{V}_k \cap \mathcal{V}_{k'} = \emptyset$ for $k \neq k'$. Given two graphs G_s and G_t, graph matching aims to find a correspondence between their nodes, *i.e.*, $\pi : \mathcal{V}_s \mapsto \mathcal{V}_t$. Many real-world networks are modeled using graph theory, and graph partitioning and matching are important for community detection [21, 16] and network alignment [39, 40, 54], respectively. In this section, we propose a Gromov-Wasserstein learning framework to unify these two problems.

2.1 Gromov-Wasserstein discrepancy between graphs

Our GWL framework is based on a pseudometric on graphs called Gromov-Wasserstein discrepancy:

Definition 2.1 ([11]). *Denote the collection of measure graphs as $\mathcal{G}$. For each $p \in [1, \infty]$ and each $G_s, G_t \in \mathcal{G}$, the Gromov-Wasserstein discrepancy between G_s and G_t is*

$$d_{gw}(G_s, G_t) := \min_{\boldsymbol{T}\in\Pi(\boldsymbol{\mu}_s,\boldsymbol{\mu}_t)} \Big(\sum\nolimits_{i,j\in\mathcal{V}_s} \sum\nolimits_{i',j'\in\mathcal{V}_t} |c^s_{ij} - c^t_{i'j'}|^p T_{ii'} T_{jj'} \Big)^{\frac{1}{p}}, \tag{1}$$

where $\Pi(\boldsymbol{\mu}_s, \boldsymbol{\mu}_t) = \{\boldsymbol{T} \geq \boldsymbol{0} | \boldsymbol{T}\boldsymbol{1}_{|\mathcal{V}_t|} = \boldsymbol{\mu}_s, \boldsymbol{T}^\top \boldsymbol{1}_{|\mathcal{V}_s|} = \boldsymbol{\mu}_t\}$.

GW discrepancy compares graphs in a relational way, measuring how the edges in a graph compare to those in the other graph. It is a natural extension of the Gromov-Wasserstein distance defined for metric-measure spaces [29]. We refer the reader to [29, 11, 36] for mathematical foundations.

Graph matching According to the definition, GW discrepancy measures the distance between two graphs, and the optimal transport $\boldsymbol{T} = [T_{ij}] \in \Pi(\boldsymbol{\mu}_s, \boldsymbol{\mu}_t)$ is a joint distribution of the graphs' nodes: T_{ij} indicates the probability that the node $v_i^s \in \mathcal{V}_s$ corresponds to the node $v_j^t \in \mathcal{V}_t$. As shown in Figure 1(a), the optimal transport achieves an assignment of the source nodes to the target ones.

Graph partitioning Besides graph matching, this paradigm is also suitable for graph partitioning. Recall that most existing graph partitioning methods obey the modularity maximization principle [16, 12]: for each partitioned sub-graph, its internal edges should be dense, while its external edges connecting with other sub-graphs should be sparse. This principle implies that if we treat each sub-graph as a "super node" [21, 47, 34], an ideal partitioning should correspond to a disconnected graph with K isolated, but self-connected super nodes. Therefore, we achieve K-way partitioning by calculating the GW discrepancy between the observed graph G and a disconnected graph, $i.e.$, $d_{gw}(G, G_{\text{dc}})$, where $G_{\text{dc}} = G(\mathcal{V}_{\text{dc}}, \text{diag}(\boldsymbol{\mu}_{\text{dc}}), \boldsymbol{\mu}_{\text{dc}})$. $|\mathcal{V}_{\text{dc}}| = K$. $\boldsymbol{\mu}_{\text{dc}} \in \Sigma^K$ is a node distribution, whose derivation is in Appendix A.1. $\text{diag}(\boldsymbol{\mu}_{\text{dc}})$ is the adjacency matrix of G_{dc}. As shown in Figure 1(b), the optimal transport is a $|\mathcal{V}| \times K$ matrix. The maximum in each row of the matrix indicates the cluster of a node.

2.2 Gromov-Wasserstein barycenter graph for analysis of multiple graphs

Multi-graph matching Distinct from most graph matching methods [17, 13, 39, 14], which mainly focus on two-graph matching, our GWL framework can be readily extended to multi-graph cases, by introducing the Gromov-Wasserstein barycenter (GWB) proposed in [37]. Given a set of graphs $\{G_m\}_{m=1}^M$, their p-order Gromov-Wasserstein barycenter is a *barycenter graph* defined as

$$G(\bar{\mathcal{V}}, \bar{\boldsymbol{C}}, \bar{\boldsymbol{\mu}}) := \arg\min_{\bar{G}} \sum_{m=1}^{M} \omega_m d_{gw}^p(G_m, \bar{G}), \tag{2}$$

where $\boldsymbol{\omega} = [\omega_m] \in \Sigma^M$ contains predefined weights, and $\bar{G} = G(\bar{\mathcal{V}}, \bar{\boldsymbol{C}} \in \mathbb{R}^{|\bar{\mathcal{V}}| \times |\bar{\mathcal{V}}|}, \bar{\boldsymbol{\mu}} \in \Sigma^{|\bar{\mathcal{V}}|})$ is the barycenter graph with a predefined number of nodes. The barycenter graph minimizes the weighted average of its GW discrepancy to observed graphs. It is an average of the observed graphs aligned by their optimal transports. The matrix $\bar{\boldsymbol{C}}$ is a "soft" adjacency matrix of the barycenter. Its elements reflect the confidence of the edges between the corresponding nodes in $\bar{\mathcal{V}}$. As shown in Figure 1(c), the barycenter graph works as a "reference" connecting with the observed graphs. For each node in the barycenter graph, we can find its matched nodes in different graphs with the help of the corresponding optimal transport. These matched nodes construct a node set, and two arbitrary nodes in the set are a correspondence. The collection of all the node sets achieves multi-graph matching.

Multi-graph partitioning We can also use the barycenter graph to achieve multi-graph partitioning, with the *learned* barycenter graph playing the role of the aforementioned disconnected graph. Given two or more graphs, whose nodes may have unobserved correspondences, existing partitioning methods [21, 16, 12, 6, 34] only partition them independently because they are designed for clustering nodes in a single graph. As a result, the first cluster of a graph may correspond to the second cluster of another graph. Without the correspondence between clusters, we cannot reduce the search space in matching tasks. Although this correspondence can be estimated by matching two coarse graphs that treat the clusters as their nodes, this strategy not only introduces additional computations but also leads to more uncertainty on matching, because different graphs are partitioned independently without leveraging structural information from each other. By learning a barycenter graph for multiple graphs, we can partition them and align their clusters simultaneously. As shown in Figure 1(d), when applying K-way multi-graph partitioning, we initialize a disconnected graph with K isolated nodes as the barycenter graph, and then learn it by $\min_{\bar{G}} \sum_{m=1}^{M} \omega_m d_{gw}^p(G_m, \bar{G})$. For each node of the barycenter graph, its matched nodes in each observed graph belong to the same cluster.

3 Scalable Gromov-Wasserstein Learning

Based on Gromov-Wasserstein discrepancy and the barycenter graph, we have established a GWL framework for graph partitioning and matching. To make this framework scalable to large graphs, we propose a regularized proximal gradient method to calculate GW discrepancy and integrate multiple acceleration strategies to greatly reduce the computational complexity of GWL.

3.1 Regularized proximal gradient method

Inspired by the work in [48, 49], we calculate the GW discrepancy in (1) based on a proximal gradient method, which decomposes a complicated non-convex optimization problem into a series of convex sub-problems. For simplicity, we set $p = 2$ in (1, 2). Given two graphs $G_s = G(\mathcal{V}_s, \boldsymbol{C}_s, \boldsymbol{\mu}_s)$ and

$G_t = G(\mathcal{V}_t, \boldsymbol{C}_t, \boldsymbol{\mu}_t)$, in the n-th iteration, we update the current optimal transport $\boldsymbol{T}^{(n)}$ by calculating $d_{gw}^2(G_s, G_t)$:

$$\begin{aligned}\boldsymbol{T}^{(n+1)} &= \arg\min_{\boldsymbol{T}\in\Pi(\boldsymbol{\mu}_s,\boldsymbol{\mu}_t)} \sum\nolimits_{i,j\in\mathcal{V}_s} \sum\nolimits_{i',j'\in\mathcal{V}_t} |c_{ij}^s - c_{i'j'}^t|^2 T_{ii'}^{(n)} T_{jj'} + \gamma \mathrm{KL}(\boldsymbol{T}\|\boldsymbol{T}^{(n)}) \\ &= \arg\min_{\boldsymbol{T}\in\Pi(\boldsymbol{\mu}_s,\boldsymbol{\mu}_t)} \langle \boldsymbol{L}(\boldsymbol{C}_s, \boldsymbol{C}_t, \boldsymbol{T}^{(n)}), \boldsymbol{T}\rangle + \gamma \mathrm{KL}(\boldsymbol{T}\|\boldsymbol{T}^{(n)}).\end{aligned} \tag{3}$$

Here, $\boldsymbol{L}(\boldsymbol{C}_s, \boldsymbol{C}_t, \boldsymbol{T}) = \boldsymbol{C}_s \boldsymbol{\mu}_s \mathbf{1}_{|\mathcal{V}_t|}^\top + \mathbf{1}_{|\mathcal{V}_s|} \boldsymbol{\mu}_t^\top \boldsymbol{C}_t^\top - 2\boldsymbol{C}_s \boldsymbol{T} \boldsymbol{C}_t^\top$, derived based on [37], and $\langle \cdot, \cdot \rangle$ represents the inner product of two matrices. The Kullback-Leibler (KL) divergence, $i.e.$, $\mathrm{KL}(\boldsymbol{T}\|\boldsymbol{T}^{(n)}) = \sum_{ij} T_{ij} \log(T_{ij}/T_{ij}^{(n)}) - T_{ij} + T_{ij}^{(n)}$, is added as the proximal term. We can solve (3) via the Sinkhorn-Knopp algorithm [41, 15] with nearly-linear convergence [1]. As demonstrated in [49], the global convergence of this proximal gradient method is guaranteed, so repeating (3) leads to a stable optimal transport, denoted as $\widehat{\boldsymbol{T}}$. Additionally, this method is robust to hyperparameter γ, achieving better convergence and numerical stability than the entropy-based method in [37].

Learning the barycenter graph is also based on the proximal gradient method. Given M graphs, we estimate their barycenter graph via alternating optimization. In the n-th iteration, given the previous barycenter graph $\bar{G}^{(n)} = G(\bar{\mathcal{V}}, \bar{\boldsymbol{C}}^{(n)}, \bar{\boldsymbol{\mu}})$, we update M optimal transports via solving (3). Given the updated optimal transports $\{\boldsymbol{T}_m^{(n+1)}\}_{m=1}^M$, we update the adjacency matrix of the barycenter graph by

$$\bar{\boldsymbol{C}}^{(n+1)} = \frac{1}{\bar{\boldsymbol{\mu}}\bar{\boldsymbol{\mu}}^\top} \sum\nolimits_m \omega_m (\boldsymbol{T}_m^{(n+1)})^\top \boldsymbol{C}_m \boldsymbol{T}_m^{(n+1)}. \tag{4}$$

The weights $\boldsymbol{\omega}$, the number of the nodes $|\bar{\mathcal{V}}|$ and the node distribution $\bar{\boldsymbol{\mu}}$ are predefined.

Different from the work in [49, 37], we use the following initialization strategies to achieve a regularized proximal gradient method and estimate optimal transports with few iterations.

Node distributions We estimate the node distribution $\boldsymbol{\mu}$ of a graph empirically by a function of node degree, which reflects the local topology of nodes, $e.g.$, the density of neighbors. In particular, for a graph with $|\mathcal{V}|$ nodes, we first calculate a vector of node degree, $i.e.$, $\boldsymbol{n} = [n_i] \in \mathbb{Z}^{|\mathcal{V}|}$, where n_i is the number of neighbors of the i-th node. Then, we estimate the node distribution $\boldsymbol{\mu}$ as

$$\boldsymbol{\mu} = \tilde{\boldsymbol{\mu}}/\|\tilde{\boldsymbol{\mu}}\|_1, \quad \tilde{\boldsymbol{\mu}} = (\boldsymbol{n} + a)^b. \tag{5}$$

where $a \geq 0$ and $b \geq 0$ are the hyperparameters controlling the shape of the distribution. For the graphs with isolated nodes, whose n_i's are zeros, we set $a > 0$ to avoid numerical issues when solving (3). For the graphs whose nodes obey to power-law distributions, $i.e.$, Barabási-Albert graphs, we set $b \in [0, 1)$ to balance the probabilities of different nodes. This function generalizes the empirical settings used in other methods: when $a = 0$ and $b = 1$, we derive the distribution based on the normalized node degree used in [49]; when $b = 0$, we assume the distribution is uniform as the work in [37, 44] does. We find that the node distributions have a huge influence on the stability and the performance of our learning algorithms, which will be discussed in the following sections.

Optimal transports For graph analysis, we can leverage prior knowledge to get a better regularization of optimal transport. Generally, the nodes with similar local topology should be matched with a high probability. Therefore, given two node distributions $\boldsymbol{\mu}_s$ and $\boldsymbol{\mu}_t$, we construct a node-based cost matrix $\boldsymbol{C}_{\text{node}} \in \mathbb{R}^{|\mathcal{V}_s|\times|\mathcal{V}_t|}$, whose element is $c_{ij} = |\mu_i^s - \mu_j^t|$, and add a regularization term $\langle \boldsymbol{C}_{\text{node}}, \boldsymbol{T}^{(n)} \rangle$ to (3). As a result, in the learning phase, we replace the $\boldsymbol{L}(\boldsymbol{C}_s, \boldsymbol{C}_t, \boldsymbol{T}^{(n)})$ in (3) with $\boldsymbol{L}(\boldsymbol{C}_s, \boldsymbol{C}_t, \boldsymbol{T}^{(n)}) + \tau \boldsymbol{C}_{\text{node}}$, where τ controls the significance of $\boldsymbol{C}_{\text{node}}$. Introducing the proposed regularizer helps us measure the similarity between nodes directly, which extends our GW discrepancy to the fused GW discrepancy in [44, 43]. In such a situation, the main difference here is that we use the proximal gradient method to calculate the discrepancy, rather than the conditional gradient method in [43].

Barycenter graphs When learning GWB, the work in [37] fixed the node distribution to be uniform In practice, however, both the node distribution of the barycenter graph and its optimal transports to observed graphs are unknown. In such a situation, we need to first estimate the node distribution $\bar{\boldsymbol{\mu}} = [\bar{\mu}_1, ..., \bar{\mu}_{|\bar{\mathcal{V}}|}]$. Without loss of generality, we assume that the node distribution of the barycenter graph is sorted, $i.e.$, $\bar{\mu}_1 \geq ... \geq \bar{\mu}_{|\bar{\mathcal{V}}|}$. We estimate the node distribution via the weighted average of the sorted and re-sampled node distributions of observed graphs:

$$\bar{\boldsymbol{\mu}} = \sum\nolimits_{m=1}^M \omega_m \text{interpolate}_{|\bar{\mathcal{V}}|}(\text{sort}(\boldsymbol{\mu}_m)), \tag{6}$$

Algorithm 1 ProxGrad(G_s, G_t, γ)

Set $n = 0$, $\boldsymbol{a} = \boldsymbol{\mu}_s$.
Calculate $\boldsymbol{C}_{\text{node}}$ with $c_{ij} = |\mu_i^s - \mu_j^t|$.
Initialize $\boldsymbol{T}^{(n)} = \boldsymbol{\mu}_s \boldsymbol{\mu}_t^\top$.
While not converge
  $\boldsymbol{G} = e^{-(\boldsymbol{C}_{\text{node}} + \boldsymbol{L}(\boldsymbol{C}_s, \boldsymbol{C}_t, \boldsymbol{T}^{(n)}))/\gamma} \odot \boldsymbol{T}^{(n)}$.
  $\boldsymbol{b} = \boldsymbol{\mu}_t / (\boldsymbol{G}^\top \boldsymbol{a})$, and $\boldsymbol{a} = \boldsymbol{\mu}_s / (\boldsymbol{G}\boldsymbol{b})$.
  $\boldsymbol{T}^{(n+1)} = \text{diag}(\boldsymbol{a}) \boldsymbol{G} \text{diag}(\boldsymbol{b})$, then $n = n + 1$.
Output: $\widehat{\boldsymbol{T}} = \boldsymbol{T}^{(n)}$.

Algorithm 2 GWB$(\{G_m\}_{m=1}^M, \gamma, |\bar{\mathcal{V}}|, \boldsymbol{\omega})$

Set $n = 0$.
Initialize $\bar{\boldsymbol{\mu}}$ via (6). $\bar{\boldsymbol{C}}^{(n)} = \text{diag}(\bar{\boldsymbol{\mu}})$.
While not converge
  For $m = 1, ..., M$
    $\boldsymbol{T}_m^{(n+1)} = \text{ProxGrad}(G_m, \bar{G}^{(n)}, \gamma)$.
  Calculate $\bar{\boldsymbol{C}}^{(n+1)}$ via (4).
  $n = n + 1$.
Output: $\widehat{\boldsymbol{T}}_m = \boldsymbol{T}_m^{(n)}$ for $m = 1, .., M$.

where sort$(\cdot)$ sorts the elements of the input vector in descending order, and interpolate$_{|\bar{\mathcal{V}}|}(\cdot)$ samples $|\bar{\mathcal{V}}|$ values from the input vector via bilinear interpolation. Given the node distribution, we initialize the optimal transports via the method mentioned above.

Algorithms 1 and 2 show the details of our method, where "$\odot$" and "$\cdot/\cdot$" represent elementwise multiplication and division, respectively. The GWL framework for the tasks in Figures 1(a)-1(d) are implemented based on these two algorithms, with details in Appendix A.1.

3.2 A recursive K-partition mechanism for large-scale graph matching

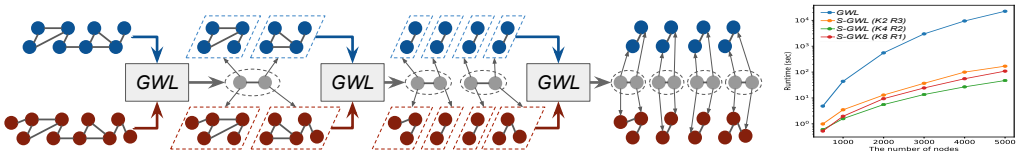
Assume that the observed graphs have comparable size, whose number of nodes and edges are denoted as V and E, respectively. When using the proximal gradient method directly to calculate the GW discrepancy between two graphs, the time complexity, in the worst case, is $\mathcal{O}(V^3)$ because the $\boldsymbol{L}(\boldsymbol{C}_s, \boldsymbol{C}_t, \boldsymbol{T}^{(n)})$ in (3) involves $\boldsymbol{C}_s \boldsymbol{T} \boldsymbol{C}_t^\top$. Even if we consider the sparsity of edges and implement sparse matrix multiplications, the time complexity is still as high as $\mathcal{O}(EV)$.

To improve the scalability of our GWL framework, we introduce a recursive K-partition mechanism, recursively decomposing observed large graphs to a set of aligned small graphs. As shown in Figure 2(a), given two graphs, we first calculate their barycenter graph (with K nodes) and achieve their joint K-way partitioning. For each node of the barycenter graph, the corresponding sub-graphs extracted from the observed two graphs construct an aligned sub-graph pair, shown as the dotted frames connected with grey circles in Figure 2(a). For each aligned sub-graph pair, we further calculate its barycenter graph and decompose the pair into more and smaller sub-graph pairs. Repeating the above step, we finally calculate the GW discrepancy between the sub-graphs in each pair, and find the correspondence between their nodes. Note that this recursive mechanism is also applicable to multi-graph matching: for multiple graphs, in the final step we calculate the GWB among the sub-graphs in each set. The details of our S-GWL method are provided in Appendix A.2.

Complexity analysis In Table 1, we compare the time and memory complexity of our S-GWL method with other matching methods. The Hungarian algorithm [24] has time complexity $\mathcal{O}(V^3)$ [17, 33, 50]. Denoting the largest node degree in a graph as d, the time complexity of GHOST [35] is $\mathcal{O}(d^4)$. The methods above take the graph affinity matrix as input, so their memory complexity in the worst case is $\mathcal{O}(V^4)$. MI-GRAAL [23], HubAlign [19] and NETAL [32] are relatively efficient, with time complexity $\mathcal{O}(VE + V^2 \log V)$, $\mathcal{O}(V^2 \log V)$ and $\mathcal{O}(E^2 + EV \log V)$, respectively. CPD+Emb first learns D-dimensional node embeddings [18], and then registers the embeddings by the CPD method [31], whose time complexity is $\mathcal{O}(DV^2)$. The memory complexity of these four methods is $\mathcal{O}(V^2)$. For GW discrepancy-based methods, the GWL+Emb in [49] achieves graph matching and node embedding jointly. It uses the distance matrix of node embeddings and breaks the sparsity of edges, so its time complexity is $\mathcal{O}(V^3)$ and memory complexity is $\mathcal{O}(V^2)$. The time complexity of GWL is $\mathcal{O}(VE)$, but its memory complexity is still $\mathcal{O}(V^2)$ because the $\boldsymbol{L}(\boldsymbol{C}_s, \boldsymbol{C}_t, \boldsymbol{T}^{(n)})$ in (3) is a dense matrix. Our S-GWL combines the recursive mechanism with the regularized proximal gradient method and implements the $\boldsymbol{C}_s \boldsymbol{T}^{(n)} \boldsymbol{C}_t^\top$ in (3) by sparse matrix multiplications. Ideally, we can apply $R = \lfloor \log_K V \rfloor$ recursions. In the r-th recursion we calculate K^r barycenter graphs for K^r sub-graph pairs. The sub-graphs in each pair have $\mathcal{O}(\frac{V}{K^r})$ nodes. As a result, we have

Proposition 3.1. *Suppose that we have M graphs, each of which has V nodes and E edges. With the help of the recursive K-partition mechanism, the time complexity of our S-GWL method is $\mathcal{O}(MK(E + V) \log_K V)$, and its memory complexity is $\mathcal{O}(M(E + VK))$.*

Choosing $K = 2$ and ignoring the number of graphs, we obtain the complexity shown in Table 1. Our S-GWL has lower computational time complexity and memory requirements than many existing



(a) Scheme of our S-GWL method (b) Runtime

Figure 2: (a) An illustration of S-GWL. (b) Comparisons on runtime.

Table 1: Comparisons for graph matching methods on time and memory complexity.

	Hungarian	GHOST*	MI-GRAAL	HubAlign	NETAL	CPD+Emb	GWL+Emb	GWL	S-GWL
Time $\mathcal{O}(\cdot)$	V^3	d^4	VE+$V^2 \log V$	$V^2 \log V$	E^2+$EV \log V$	DV^2	V^3	VE	$2(E$+$V) \log V$
Memory $\mathcal{O}(\cdot)$	V^4	V^4	V^2	V^2	V^2	V^2	V^2	V^2	$E+2V$

* d is the largest node degree in a graph.

methods. Figure 2(b) visualizes the runtime of GWL and S-GWL on matching synthetic graphs. The S-GWL methods with different configurations ($i.e.$, the number of partitions K and that of recursions R) are consistently faster than GWL. More detailed analysis is provided in Appendix A.3.

4 Related Work

Gromov-Wasserstein learning GW discrepancy has been applied in many matching problems, $e.g.$, registering 3D objects [28, 29] and matching vocabulary sets between different languages [2]. Focusing on graphs, a fused Gromov-Wasserstein distance is proposed in [44, 43], combining GW discrepancy with Wasserstein discrepancy [46]. The work in [49] further takes node embedding into account, learning the GW discrepancy between two graphs and their node embeddings jointly. The appropriateness of these methods is supported by [11], which proves that GW discrepancy is a pseudometric on measure graphs. Recently, an adversarial learning method based on GW discrepancy is proposed in [9], which jointly trains two generative models in incomparable spaces. The work in [37] further proposes Gromov-Wasserstein barycenters for clustering distributions and interpolating shapes. Currently, GW discrepancy is mainly calculated based on Sinkhorn iterations [41, 15, 5, 37], whose applications to large-scale graphs are challenging because of its high complexity. Our S-GWL method is the first attempt to make GW discrepancy applicable to large-scale graph analysis.

Graph partitioning and graph matching Graph partitioning is important for community detection in networks. Many graph partitioning methods have been proposed, such as Metis [21], EdgeBetweenness [16], FastGreedy [12], Label Propagation [38], Louvain [6] and Fluid Community [34]. All of these methods explore the clustering structure of nodes heuristically based on the modularity-maximization principle [16, 12]. Graph matching is important for network alignment [39, 40, 54] and 2D/3D object registration [31, 51, 20, 53]. Traditional methods formulate graph matching as a quadratic assignment problem (QAP) and solve it based on the Hungarian algorithm [17, 33, 51, 50], which are only applicable to small graphs. For large graphs like protein networks, many heuristic methods have been proposed, such as GRAAL [22], IsoRank [40], PISwap [10], MAGNA++ [45], NETAL [32], HubAlign [19], and GHOST [35], which mainly focus on two-graph matching and are sensitive to the noise in graphs. With the help of GW discrepancy, our work establishes a unified framework for graph partitioning and matching, that can be readily extended to multi-graph cases.

5 Experiments

The implementation of our S-GWL method can be found at https://github.com/HongtengXu/s-gwl. We compare it with state-of-the-art methods for graph partitioning and matching. All the methods are run on an Intel i7 CPU with 4GB memory. Implementation details and a further set of experimental results are provided in Appendix B.

5.1 Graph partitioning

We first verify the performance of the **GWL** framework on graph partitioning, comparing it with the following four baselines: **Metis** [21], **FastGreedy** [12], **Louvain** [6], and **Fluid** Community [34]. We consider synthetic and real-world data. Similar to [52], we compare these methods in terms of adjusted mutual information (AMI) and runtime. Each synthetic graph is a Gaussian random partition graph with N nodes and K clusters. The size of each cluster is drawn from a normal distribution $\mathcal{N}(200, 10)$. The nodes are connected within clusters with probability p_{in} and between clusters with probability p_{out}. The ratio $\frac{p_{\text{out}}}{p_{\text{in}}}$ indicates the clearness of the clustering structure, and accordingly

Table 2: Comparisons for graph partitioning methods on AMI, time complexity and runtime (second).

Method	Metis		FastGreedy		Louvain		Fluid		GWL	
Time complexity	$\mathcal{O}(V+E+K\log K)$		$\mathcal{O}(VE\log V)$		$\mathcal{O}(V\log V)$		$\mathcal{O}(E)$		$\mathcal{O}((E+V)K)$	
$(N, p_{\text{in}}, p_{\text{out}})$	AMI	Time	AMI	Time	AMI	Time	AMI	Time	AMI	Time
$(4000, 0.2, 0.05)$	0.413	1.744	0.247	55.435	0.747	22.889	0.776	21.580	**0.812**	13.033
$(4000, 0.2, 0.1)$	0.009	2.340	0.064	65.441	0.574	95.114	0.577	111.043	**0.590**	12.740
$(4000, 0.2, 0.15)$	0.002	3.592	0.002	80.322	0.005	290.846	0.005	203.225	**0.012**	12.901

Table 3: Comparisons for graph partitioning methods on AMI.

Method	Metis		FastGreedy		Louvain		Fluid		GWL	
Dataset	Raw	Noisy	Raw	Noisy	Raw	Noisy	Raw	Noisy	Raw	Noisy
EU-Email	0.421	0.246	0.312	0.118	0.434	0.272	—	0.338	**0.459**	**0.349**
Indian-Village	0.834	0.513	**0.882**	0.275	0.880	0.633	—	0.401	0.857	**0.664**

"—": Fluid is inapplicable when the networks have disconnected nodes or sub-graphs.

the difficulty of partitioning. We set $N = 4000$, $p_{\text{in}} = 0.2$, and $p_{\text{out}} \in \{0.05, 0.1, 0.15\}$. Under each configuration $(N, p_{\text{in}}, p_{\text{out}})$, we simulate 10 graphs. For each method, its average performance on these 10 graphs is listed in Table 2. GWL outperforms the alternatives consistently on AMI. Additionally, as shown in Table 2, GWL has time complexity comparable to other methods, especially when the graph is sparse, *e.g.*, $E = \mathcal{O}(V \log V)$. According to the runtime in practice, GWL is faster than most baselines except Metis, likely because Metis is implemented in the C language while GWL and other methods are based on Python.

Table 3 lists the performance of different methods on two real-world datasets. The first dataset is the email network from a large European research institution [25]. The network contains 1,005 nodes and 25,571 edges. The edge (v_i, v_j) in the network mean that person v_i sent person v_j at least one email, and each node in the network belongs to exactly one of 42 departments at the research institute. The second dataset is the interactions among 1,991 villagers in 12 Indian villages [3]. Furthermore, to verify the robustness of GWL to noise, we not only consider the raw data of these two datasets but also create their noisy version by adding 10% more noisy edges between different communities (*i.e.*, departments and villages). Experimental results show that GWL is at least comparable to its competitors on raw data, and it is more robust to noise than other methods.

5.2 Graph matching

For two-graph matching, we compare our S-GWL method with the following baselines: **PISwap** [10], **GHOST** [35], **MI-GRAAL** [23], **MAGNA++** [45], **HubAlign** [19], **NETAL** [32], **CPD+Emb** [18, 31], the **GWL** framework based on Algorithm 1, and the **GWL+Emb** in [49]. We test all methods on both synthetic and real-world data. For each method, given the learned correspondence set $\mathcal{S}$ and the ground-truth correspondence set $\mathcal{S}_{real}$, we calculate node correctness as $\text{NC} = |\mathcal{S} \cap \mathcal{S}_{real}|/|\mathcal{S}| \times 100\%$. The runtime of each method is recorded as well.

In the synthetic dataset, each source graph $G(\mathcal{V}_s, \mathcal{E}_s)$ obeys a Gaussian random partition model [7] or Barabási-Albert model [4]. For each source graph, we generate a target graph by adding $|\mathcal{V}_s| \times q\%$ noisy nodes and $|\mathcal{E}_s| \times q\%$ noisy edges to the source graph. Figure 1(e) compares our S-GWL with the baselines when $|\mathcal{V}_s| = 2000$ and $q = 5$. For each method, its average node correctness and runtime on matching 10 synthetic graph pairs are plotted. Compared with existing heursitic methods, GW discrepancy-based methods (GWL+Emb, GWL and S-GWL) obtain much higher node correctness. GWL+Emb achieves the highest node correctness, with runtime comparable to many baselines. Our GWL framework does not learn node embeddings when matching graphs, so it is slightly worse than GWL+Emb on node correctness but achieves about 10 times acceleration. Our S-GWL method further accelerates GWL with the help of the recursive mechanism. It obtains high node correctness and makes its runtime comparable to the fastest methods (HubAlign and NETAL).

In addition to graph matching on synthetic data, we also consider two real-world matching tasks. The first task is matching the protein-protein interaction (PPI) network of yeast with its noisy version. The PPI network of yeast contains 1,004 proteins and their 4,920 high-confidence interactions. Its noisy version contains $q\%$ more low-confidence interactions, and $q \in \{5, 10, 15, 20, 25\}$. The dataset is available on https://www3.nd.edu/~cone/MAGNA++/. The second task is matching user accounts in different communication networks. The dataset is available on http://vacommunity.org/VAST+Challenge+2018+MC3, which records the communications among a company's employees. Following the work in [49], we extract 622 employees and their *call-network* and *email-network*.

Table 4: Comparisons for graph matching methods on node correctness (%) and runtime (second).

Dataset	Yeast 5% noise		Yeast 15% noise		Yeast 25% noise		MC3 sparse		MC3 dense	
Method	NC	Time	NC	Time	NC	Time	NC	Time	NC	Time
PISwap	0.10	15.80	0.10	18.31	0.00	22.09	6.32	10.27	0.00	11.81
GHOST	11.06	25.67	0.40	30.22	0.30	35.54	21.27	17.86	0.03	22.90
MI-GRAAL	18.03	189.21	6.87	202.77	5.18	240.03	35.53	72.89	0.64	197.65
MAGNA++	48.13	603.29	25.04	630.60	13.61	624.17	7.88	425.16	0.09	447.86
HubAlign	50.00	3.27	35.16	3.50	12.85	3.89	36.21	2.11	3.86	2.29
NETAL	6.87	1.91	0.90	2.06	1.00	2.09	36.87	1.23	1.77	1.30
CPD+Emb	3.59	103.22	2.09	110.19	2.00	108.62	4.35	87.54	0.48	95.68
GWL+Emb	83.66	1340.58	66.63	1499.20	57.97	1537.93	40.45	608.76	4.23	831.80
GWL	82.37	190.97	65.34	212.16	58.76	210.86	34.21	89.43	3.96	93.94
S-GWL	81.08	68.58	61.85	70.06	56.27	74.64	36.92	8.39	4.03	9.01

Table 5: Comparisons for multi-graph matching methods on yeast networks.

Method	3 graphs		4 graphs		5 graphs		6 graphs	
	NC@1	NC@all	NC@1	NC@all	NC@1	NC@all	NC@1	NC@all
MultiAlign	62.97	45.19	—	—	—	—	—	—
GWL	**63.84**	**46.22**	**68.73**	**39.14**	71.61	31.57	76.49	28.39
S-GWL	60.06	43.33	68.53	38.45	**73.21**	**33.27**	**76.99**	**29.68**

For each communication network, we construct a dense version and a sparse one: the dense version keeps all the communications (edges) among the employees, while the sparse version only preserves the communications happening more than 8 times. We test different methods on i) matching yeast's PPI network with its 5%, 15% and 25% noisy versions; and ii) matching the employee call-network with their email-network in both sparse and dense cases. Table 4 shows the performance of various methods in these two tasks. Similar to the experiments on synthetic data, the GW discrepancy-based methods outperform other methods on node correctness, especially for highly-noisy graphs, and our S-GWL method achieves a good trade-off between accuracy and efficiency.

Given the PPI network of yeast and its 5 noisy versions, we test GWL and S-GWL for multi-graph matching. We consider several existing multi-graph matching methods and find that the methods in [33, 51, 50] are not applicable for the graphs with hundreds of nodes because i) their time complexity is at least $\mathcal{O}(V^3)$, and ii) they suffer from inadequate memory on our machine (with 4GB memory) because their memory complexity in the worst case is $\mathcal{O}(V^4)$. The IsoRankN in [26] can align multiple PPI networks jointly, but it needs confidence scores of protein pairs as input, which are not available for our dataset. The only applicable baseline we are aware of is the **MultiAlign** in [54]. However, it can only achieve three-graph matching. Table 5 lists the performance of various methods. Given learned correspondence sets, each of which is a set of matched nodes from different graphs, NC@1 represents the percentage of the set containing at least a pair of correctly-matched nodes, and NC@all represents the percentage of the set in which arbitrary two nodes are matched correctly. Both GWL and S-GWL obtain comparable performance to MultiAlign on three-graph matching, and GWL is the best. When the number of graphs increases, NC@1 increases while NC@all decreases for all the methods, and S-GWL becomes even better than GWL.

6 Conclusion and Future Work

We have developed a scalable Gromov-Wasserstein learning method, achieving large-scale graph partitioning and matching in a unified framework, with theoretical support. Experiments show that our approach outperforms state-of-the-art methods in many situations. However, it should be noted that our S-GWL method is sensitive to its hyperparameters. Specifically, we observed in our experiments that the γ in (3) should be set carefully according to observed graphs. Generally, for large-scale graphs we have to use a large γ and solve (3) with many iterations. The a and b in (5) are also significant for the performance of our method. The settings of these hyperparameters and their influences are shown in Appendix B. In the future, we will further study the influence of hyperparameters on the rate of convergence and set the hyperparameters adaptively according to observed data. Additionally, our S-GWL method can decompose a large graph into many independent small graphs, so we plan to further accelerate it by parallel processing and/or distributed learning.

Acknowledgements This research was supported in part by DARPA, DOE, NIH, ONR and NSF. We thank Dr. Hongyuan Zha for helpful discussions.

References

[1] J. Altschuler, J. Weed, and P. Rigollet. Near-linear time approximation algorithms for optimal transport via sinkhorn iteration. In *Advances in Neural Information Processing Systems*, pages 1964–1974, 2017.

[2] D. Alvarez-Melis and T. Jaakkola. Gromov-wasserstein alignment of word embedding spaces. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1881–1890, 2018.

[3] A. Banerjee, A. G. Chandrasekhar, E. Duflo, and M. O. Jackson. The diffusion of microfinance. *Science*, 341(6144):1236498, 2013.

[4] A.-L. Barabási et al. *Network science*. Cambridge university press, 2016.

[5] J.-D. Benamou, G. Carlier, M. Cuturi, L. Nenna, and G. Peyré. Iterative Bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.

[6] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[7] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *European Symposium on Algorithms*, pages 568–579. Springer, 2003.

[8] A. M. Bronstein, M. M. Bronstein, R. Kimmel, M. Mahmoudi, and G. Sapiro. A Gromov-Hausdorff framework with diffusion geometry for topologically-robust non-rigid shape matching. *International Journal of Computer Vision*, 89(2-3):266–286, 2010.

[9] C. Bunne, D. Alvarez-Melis, A. Krause, and S. Jegelka. Learning generative models across incomparable spaces. *NeurIPS Workshop on Relational Representation Learning*, 2018.

[10] L. Chindelevitch, C.-Y. Ma, C.-S. Liao, and B. Berger. Optimizing a global alignment of protein interaction networks. *Bioinformatics*, 29(21):2765–2773, 2013.

[11] S. Chowdhury and F. Mémoli. The Gromov-Wasserstein distance between networks and stable network invariants. *arXiv preprint arXiv:1808.04337*, 2018.

[12] A. Clauset, M. E. Newman, and C. Moore. Finding community structure in very large networks. *Physical review E*, 70(6):066111, 2004.

[13] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004.

[14] T. Cour, P. Srinivasan, and J. Shi. Balanced graph matching. In *NIPS*, pages 313–320, 2007.

[15] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013.

[16] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[17] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

[18] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016.

[19] S. Hashemifar and J. Xu. Hubalign: An accurate and efficient method for global alignment of protein–protein interaction networks. *Bioinformatics*, 30(17):i438–i444, 2014.

[20] S.-H. Jun, S. W. Wong, J. Zidek, and A. Bouchard-Côté. Sequential graph matching with sequential monte carlo. In *AISTATS*, pages 1075–1084, 2017.

[21] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[22] O. Kuchaiev, T. Milenković, V. Memišević, W. Hayes, and N. Pržulj. Topological network alignment uncovers biological function and phylogeny. *Journal of the Royal Society Interface*, page rsif20100063, 2010.

[23] O. Kuchaiev and N. Pržulj. Integrative network alignment reveals large regions of global network similarity in yeast and human. *Bioinformatics*, 27(10):1390–1396, 2011.

[24] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[25] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[26] C.-S. Liao, K. Lu, M. Baym, R. Singh, and B. Berger. Isorankn: spectral methods for global alignment of multiple protein networks. *Bioinformatics*, 25(12):i253–i258, 2009.

[27] N. Malod-Dognin and N. Pržulj. L-GRAAL: Lagrangian graphlet-based network aligner. *Bioinformatics*, 31(13):2182–2189, 2015.

[28] F. Mémoli. Spectral Gromov-Wasserstein distances for shape matching. In *ICCV Workshops*, pages 256–263, 2009.

[29] F. Mémoli. Gromov-Wasserstein distances and the metric approach to object matching. *Foundations of computational mathematics*, 11(4):417–487, 2011.

[30] F. Mémoli and G. Sapiro. Comparing point clouds. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 32–40, 2004.

[31] A. Myronenko and X. Song. Point set registration: Coherent point drift. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(12):2262–2275, 2010.

[32] B. Neyshabur, A. Khadem, S. Hashemifar, and S. S. Arab. NETAL: A new graph-based method for global alignment of protein–protein interaction networks. *Bioinformatics*, 29(13):1654–1662, 2013.

[33] D. Pachauri, R. Kondor, and V. Singh. Solving the multi-way matching problem by permutation synchronization. In *Advances in neural information processing systems*, pages 1860–1868, 2013.

[34] F. Parés, D. Garcia-Gasulla, A. Vilalta, J. Moreno, E. Ayguadé, J. Labarta, U. Cortés, and T. Suzumura. Fluid communities: A competitive and highly scalable community detection algorithm. *Complex Networks & Their Applications VI*, pages 229–240, 2018.

[35] R. Patro and C. Kingsford. Global network alignment using multiscale spectral signatures. *Bioinformatics*, 28(23):3105–3114, 2012.

[36] G. Peyré, M. Cuturi, et al. Computational optimal transport. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[37] G. Peyré, M. Cuturi, and J. Solomon. Gromov-wasserstein averaging of kernel and distance matrices. In *International Conference on Machine Learning*, pages 2664–2672, 2016.

[38] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[39] R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature biotechnology*, 24(4):427, 2006.

[40] R. Singh, J. Xu, and B. Berger. Global alignment of multiple protein interaction networks with application to functional orthology detection. *Proceedings of the National Academy of Sciences*, 2008.

[41] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[42] K.-T. Sturm et al. On the geometry of metric measure spaces. *Acta mathematica*, 196(1):65–131, 2006.

[43] T. Vayer, L. Chapel, R. Flamary, R. Tavenard, and N. Courty. Fused Gromov-Wasserstein distance for structured objects: theoretical foundations and mathematical properties. *arXiv preprint arXiv:1811.02834*, 2018.

[44] T. Vayer, L. Chapel, R. Flamary, R. Tavenard, and N. Courty. Optimal transport for structured data. *arXiv preprint arXiv:1805.09114*, 2018.

[45] V. Vijayan, V. Saraph, and T. Milenković. MAGNA++: Maximizing accuracy in global network alignment via both node and edge conservation. *Bioinformatics*, 31(14):2409–2411, 2015.

[46] C. Villani. *Optimal transport: Old and new*, volume 338. Springer Science & Business Media, 2008.

[47] L. Wang, T. Lou, J. Tang, and J. E. Hopcroft. Detecting community kernels in large social networks. In *2011 IEEE 11th International Conference on Data Mining*, pages 784–793. IEEE, 2011.

[48] Y. Xie, X. Wang, R. Wang, and H. Zha. A fast proximal point method for Wasserstein distance. *arXiv preprint arXiv:1802.04307*, 2018.

[49] H. Xu, D. Luo, H. Zha, and L. Carin. Gromov-wasserstein learning for graph matching and node embedding. *arXiv preprint arXiv:1901.06003*, 2019.

[50] J. Yan, J. Wang, H. Zha, X. Yang, and S. Chu. Consistency-driven alternating optimization for multigraph matching: A unified approach. *IEEE Transactions on Image Processing*, 24(3):994–1009, 2015.

[51] J. Yan, H. Xu, H. Zha, X. Yang, H. Liu, and S. Chu. A matrix decomposition perspective to multiple graph matching. In *ICCV*, pages 199–207, 2015.

[52] Z. Yang, R. Algesheimer, and C. J. Tessone. A comparative analysis of community detection algorithms on artificial networks. *Scientific reports*, 6:30750, 2016.

[53] T. Yu, J. Yan, Y. Wang, W. Liu, et al. Generalizing graph matching beyond quadratic assignment model. In *NIPS*, pages 861–871, 2018.

[54] J. Zhang and S. Y. Philip. Multiple anonymized social networks alignment. In *ICDM*, pages 599–608, 2015.

A Details of Algorithms

A.1 The GWL framework for different tasks

Based on Algorithms 1 and 2, our GWL framework achieve the graph partitioning and matching tasks in Figures 1(a)-1(d). The schemes of GWL for these tasks are shown in Algorithms 3-6.

Algorithm 3 $\mathcal{S} = \text{GWL-GraphMatching}(G_s, G_t, \gamma)$

Require: $G_s = G(\mathcal{V}_s, \boldsymbol{C}_s, \boldsymbol{\mu}_s)$, $G_t = G(\mathcal{V}_t, \boldsymbol{C}_t, \boldsymbol{\mu}_t)$, hyperparameter γ.
1: Initialize correspondence set $\mathcal{S} = \emptyset$.
2: $\widehat{\boldsymbol{T}} = \text{ProxGrad}(G_s, G_t, \gamma)$.
3: **For** $v_i^s \in \mathcal{V}_s$
4: Find $j = \arg\max_j \widehat{T}_{ij}$, then $\mathcal{S} = \mathcal{S} \cup \{(v_i^s, v_j^t)\}$.
5: **return** $\mathcal{S}$

Algorithm 4 $\{G_k\}_{k=1}^K = \text{GWL-GraphPartitioning}(G, \gamma, K)$

Require: $G = G(\mathcal{V}, \boldsymbol{C}, \boldsymbol{\mu})$, hyperparameter γ, the number of clusters K.
1: Initialize a node distribution via (6): $\boldsymbol{\mu}_{\text{dc}} = \text{interpolate}_K(\text{sort}(\boldsymbol{\mu}))$
2: Construct a disconnected graph $G_{\text{dc}} = G(\mathcal{V}_{\text{dc}}, \text{diag}(\boldsymbol{\mu}_{\text{dc}}), \boldsymbol{\mu}_{\text{dc}})$, where $\mathcal{V}_{\text{dc}} = \{1, ..., K\}$.
3: $\widehat{\boldsymbol{T}} = \text{ProxGrad}(G, G_{\text{dc}}, \gamma)$.
4: Initialize $\mathcal{V}_k = \emptyset$ for $k = 1, ..., K$.
5: **For** $v_i \in \mathcal{V}$
6: Find $j = \arg\max_j \widehat{T}_{ij}$, then $\mathcal{V}_j = \mathcal{V}_j \cup \{v_i\}$.
7: **For** $k = 1, ..., K$
8: Construct a adjacency matrix by selecting rows and columns: $\boldsymbol{C}_k = \boldsymbol{C}(\mathcal{V}_k, \mathcal{V}_k)$.
9: Construct a node distribution by selecting elements and normalizing them: $\boldsymbol{\mu}_k = \frac{\boldsymbol{\mu}(\mathcal{V}_k)}{\|\boldsymbol{\mu}(\mathcal{V}_k)\|_1}$.
10: **return** $\{G_k = G(\mathcal{V}_k, \boldsymbol{C}_k, \boldsymbol{\mu}_k)\}_{k=1}^K$

Algorithm 5 $\mathcal{S} = \text{GWL-MultiGraphMatching}(\mathcal{G}, \gamma)$

Require: A graph set $\mathcal{G} = \{G_m = G(\mathcal{V}_m, \boldsymbol{C}_m, \boldsymbol{\mu}_m)\}_{m=1}^M$, hyperparameter γ
1: Initialize correspondence set $\mathcal{S} = \emptyset$, $K = \min\{|\mathcal{V}_m|\}_{m=1}^M$, $\boldsymbol{\omega} = [\frac{1}{M}, .., \frac{1}{M}]$.
2: $\{\widehat{\boldsymbol{T}}_m\}_{m=1}^M = \text{GWB}(\{G_m\}_{m=1}^M, \gamma, K, \boldsymbol{\omega})$.
3: **For** $k = 1, ..., K$
4: $\boldsymbol{s} = \emptyset$
5: **For** $m = 1, .., M$
6: Find $i = \arg\max_i \widehat{T}_{ik}^m$, then $\boldsymbol{s} = \boldsymbol{s} \cup \{v_i^m\}$.
7: $\mathcal{S} = \mathcal{S} \cup \boldsymbol{s}$.
8: **return** $\mathcal{S}$.

Algorithm 6 $\{\mathcal{G}_k\}_{k=1}^K = \text{GWL-MultiGraphPartitioning}(\mathcal{G}, \gamma, K)$

Require: A graph set $\mathcal{G} = \{G_m = G(\mathcal{V}_m, \boldsymbol{C}_m, \boldsymbol{\mu}_m)\}_{m=1}^M$, hyperparameter γ, the number of clusters K.
1: Initialize $\boldsymbol{\omega} = [\frac{1}{M}, .., \frac{1}{M}]$.
2: $\{\widehat{\boldsymbol{T}}_m\}_{m=1}^M = \text{GWB}(\{G_m\}_{m=1}^M, \gamma, K, \boldsymbol{\omega})$.
3: Initialize $\mathcal{V}_{k,m} = \emptyset$ for $k = 1, .., K$ and $m = 1, .., M$.
4: **For** $m = 1, .., M$
5: **For** $v_i^m \in \mathcal{V}_m$
6: Find $j = \arg\max_j \widehat{T}_{ij}^m$, then $\mathcal{V}_{j,m} = \mathcal{V}_{j,m} \cup \{v_i^m\}$.
7: **For** $k = 1, ..., K$
8: $\boldsymbol{C}_{k,m} = \boldsymbol{C}_m(\mathcal{V}_{k,m}, \mathcal{V}_{k,m})$, and $\boldsymbol{\mu}_{k,m} = \frac{\boldsymbol{\mu}_m(\mathcal{V}_{k,m})}{\|\boldsymbol{\mu}(\mathcal{V}_{k,m})\|_1}$.
9: **return** $\{\mathcal{G}_k\}_{k=1}^K$, where $\mathcal{G}_k = \{G_{k,m} = G(\mathcal{V}_{k,m}, \boldsymbol{C}_{k,m}, \boldsymbol{\mu}_{k,m})\}_{m=1}^M$.

A.2 The scheme of S-GWL

Based on Algorithms 3, 5 and 6, we show the scheme of our S-GWL method for (multi-) graph matching in Algorithm 7.

Algorithm 7 $\mathcal{S} = \text{S-GWL}(\mathcal{G}_0, \gamma, K, R)$

Require: A graph set with M graphs, $i.e.$, $\mathcal{G}_0 = \{G_m = G(\mathcal{V}_m, \boldsymbol{C}_m, \boldsymbol{\mu}_m)\}_{m=1}^M$, γ, the number of partitions K and that of recursions R.

Initialize correspondence set $\mathcal{S} = \emptyset$.
Initialize the root collection of graph sets as $\mathsf{G}_0 = \{\mathcal{G}_0\}$.
For $r = 1, ..., R$ \\ `Recursive K-partition mechanism`
  Initialize $\mathsf{G}_r = \emptyset$.
  For each graph set $\mathcal{G} \in \mathsf{G}_{r-1}$
    $\{\mathcal{G}_k\}_{k=1}^K = \text{GWL-MultiGraphPartitioning}(\mathcal{G}, \gamma, K)$.
    $\mathsf{G}_r = \mathsf{G}_r \cup \{\mathcal{G}_k\}_{k=1}^K$.
For each graph set $\mathcal{G} \in \mathsf{G}_R$
  If $M = 2$ \\ `Two-graph matching`
    $\mathcal{S}_{tmp} = \text{GWL-GraphMatching}(G_s, G_t, \gamma)$, where $\mathcal{G} = \{G_s, G_t\}$.
  Else \\ `Multi-graph matching`
    $\mathcal{S}_{tmp} = \text{GWL-MultiGraphMatching}(\mathcal{G}, \gamma)$.
  $\mathcal{S} = \mathcal{S} \cup \mathcal{S}_{tmp}$.
return $\mathcal{S}$.

A.3 Detailed complexity analysis for GWL and S-GWL

Algorithms 3 and 5 Suppose that we have a source graph with V_s nodes and E_s edges and a target graph with V_t nodes and E_t edges. The most time- and memory-consuming operation in Algorithm 3 is the $\boldsymbol{C}_s\boldsymbol{T}^{(n)}\boldsymbol{C}_t^\top$ in (3). Because $\boldsymbol{C}_s$ is with size $V_s \times V_s$ and $\boldsymbol{C}_t$ is with size $V_t \times V_t$, the computational time complexity of this step in the worst case is $\mathcal{O}(V_s^2 V_t + V_s V_t^2)$ and its memory complexity is $\mathcal{O}(V_s^2 + V_t^2 + V_s V_t)$. Taking advantage of the sparsity of edge, $\boldsymbol{C}_s\boldsymbol{T}^{(n)}\boldsymbol{C}_t^\top$ can be implemented by sparse matrix multiplications ($i.e.$, save $\boldsymbol{C}_s$, $\boldsymbol{C}_t$ as "csr" matrix in Python), whose computational time complexity and memory cost can be reduced to $\mathcal{O}(E_s V_t + V_s E_t)$ and $\mathcal{O}(V_s V_t)$[1], respectively. Assuming that these two graphs are with comparable size, we ignore the number of graphs and the subscripts and rewrite the time and memory complexity as $\mathcal{O}(VE)$ and $\mathcal{O}(V^2)$, as shown in the "GWL" column of Table 1.

Algorithm 5 is a natural extension of Algorithm 3 based on GWB. Suppose that we have M graphs. We assume that these graphs and the target barycenter graph are with comparable size. The computational time complexity of Algorithm 5 is $\mathcal{O}(MVE)$ and its memory complexity is $\mathcal{O}(MV^2)$.

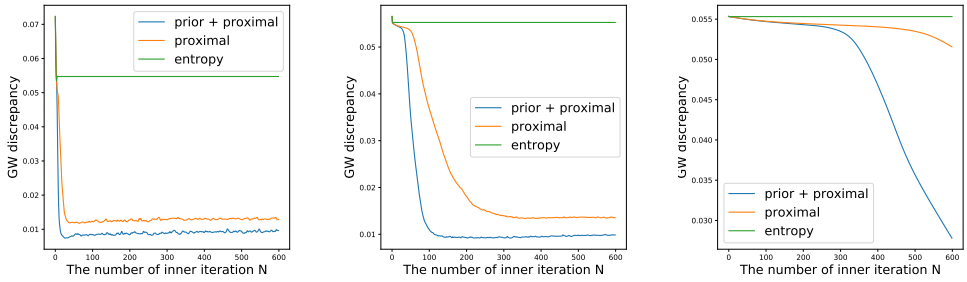
Algorithms 4 and 6 The main difference between Algorithm 4 and Algorithm 3 is that the size of target graph is much smaller than that of source graph, $i.e.$, $K = V_t \ll V_s$ and $K = E_t$, because the target graph is disconnected, whose number of nodes indicates the number of partitions in the source graph. According to the analysis above, the time and memory complexity of Algorithm 4 is $\mathcal{O}(E_s K + V_s K)$ and $\mathcal{O}(E_s + V_s K)$[2]. Ignoring the subscripts, we obtain the complexity shown in Table 2.

Similarly, Algorithm 6 is an extension of Algorithm 4 for M graphs, whose time and memory complexity is $\mathcal{O}(MK(E+V))$ and $\mathcal{O}(M(E+VK))$, respectively.

Algorithm 7 Given M graphs with comparable sizes, each of which has about V nodes and E edges, we can apply $R = \lfloor \log_K V \rfloor$ recursions. In the r-th recursion, the G_r in Algorithm 7) contains K^r sub-graph sets. If we assume that each partitioning operation partition a graph into K sub-graphs with comparable sizes, the m-th sub-graph in each set should be with $\mathcal{O}(\frac{V}{K^r})$ nodes and $\mathcal{O}(\frac{E}{K^r})$ edges. For each sub-graph set, we calculate its barycenter graph by Algorithm 6, thus, its time and memory complexity is $\mathcal{O}(MK(\frac{E}{K^r} + \frac{V}{K^r}))$ and $\mathcal{O}(\frac{M}{K^r}(E+VK))$, respectively. At the end of recursion,

[1]The memory complexity actually should be $\mathcal{O}(E_s + E_t + V_s V_t)$. Based on the sparsity of edge, we ignore the edge-related terms.

[2]Even if edges are sparse, E_s is often comparable to $V_s K$. Therefore, different from the analysis for Algorithms 3 and 5, here we do not ignore E_s.



(a) $\gamma = 0.001$ (b) $\gamma = 0.01$ (c) $\gamma = 0.1$

Figure 3: Illustrations of the improvements on convergence achieved by our proximal gradient method regularized by node prior ($i.e.$, "prior + proximal" compared with the entropy-based method in [37]) and the vanilla proximal gradient method in [49].

we obtain K^R sub-graph sets. Each sub-graph is very small, with size $\mathcal{O}(\frac{V}{K^R})$. As long as K^R is comparable to V, the computations in lines 8-13 of Algorithm 7 can be ignored compared with the computations in the recursions.

In summary, we run $\lfloor \log_K V \rfloor$ recursions, and in the r-th recursion we need to calculate K^r barycenter graphs. The overall time complexity of S-GWL is $\mathcal{O}(MK(E+V)\log_K V)$, and its memory complexity is $\mathcal{O}(M(E+VK))$, respectively, as shown in Proposition 3.1. Choosing $K = 2$ and ignoring the number of graphs, we obtain the complexity shown in Table 1.

A.4 Usefulness of node prior

With the help of the prior knowledge of node ($i.e.$, $\boldsymbol{C}_{\text{node}}$), our regularized proximal gradient method can achieve a stable optimal transport with few iterations, whose rate of convergence is faster than the entropy-based method in [37] and the vanilla proximal gradient method in [49]. Figure 3 illustrates the improvements on convergence achieved by our method. Given two synthetic graphs with 1,000 nodes, we calculate their GW discrepancy by different methods. Our method can reach lower GW discrepancy with fewer iterations, and its superiority is consistent with respect to the change of the hyperparameter γ.

B More Experimental Results

B.1 Implementation details

For each baseline, we list its source and language below:

- Graph Partitioning:
  - Metis (C): http://glaros.dtc.umn.edu/gkhome/views/metis
  - FastGreedy (Python): https://networkx.github.io/documentation/networkx-2.2/reference/algorithms/generated/networkx.algorithms.community.modularity_max.greedy_modularity_communities.html#networkx.algorithms.community.modularity_max.greedy_modularity_communities
  - Louvain (Python): https://github.com/taynaud/python-louvain
  - Fluid (Python): https://networkx.github.io/documentation/networkx-2.2/reference/algorithms/generated/networkx.algorithms.community.asyn_fluid.asyn_fluidc.html#networkx.algorithms.community.asyn_fluid.asyn_fluidc
- Graph Matching:
  - PISwap (Python): http://cb.csail.mit.edu/cb/piswap/webserver/
  - GHOST (C): http://www.cs.cmu.edu/~ckingsf/software/ghost/
  - MI-GRAAL (C): http://www0.cs.ucl.ac.uk/staff/natasa/MI-GRAAL/index.html
  - MAGNA++ (C): https://www3.nd.edu/~cone/MAGNA++/

Table 6: The settings of hyperparameters in different experiments.

Experiments	τ	a	b	γ	K	R
Synthetic partitioning (Table 2)	0	0	1	1e-2	—	—
EU-Email partitioning (Table 3)	0	0	1e-3	5e-7	—	—
Indian-Village partitioning (Table 3)	0	5e-1	1	5e-5	—	—
Synthetic matching (Figure 4)	1e1	0	1	2e-1	2	3
Yeast graph matching (Table 4)	1e3	0	1	2.5e-2	2	3
MC3 network matching (Table 4)	1e1	1	1e-1	1e-3	2	3
Yeast multi-graph matching (Table 5)	1e3	0	1	2.5e-2	8	1
Yeast-Human matching (Table 7)	1	0	5e-1	5e-2	2	4

- HubAlign and NETAL (C): https://ttic.uchicago.edu/~hashemifar/
- CPD+Emb (Python): node2vec is from https://snap.stanford.edu/node2vec/, CPD is from https://github.com/siavashk/pycpd.
- GWL+Emb (Python): https://github.com/HongtengXu/gwl.

All the baselines are tested under their default settings. For our GWL framework and S-GWL method, their hyperparameters are set empirically in different experiments, which are shown in Table 6.

Note that using non-uniform node distributions is important for our method, especicially for the cases involving multi-graph partitioning and matching. When doing multi-graph partitioning, the key step of our S-GWL, the adjacency matrix of the barycenter graph is initialized as a diagonal matrix and its node distribution is estimated by the node distributions of observed graphs. The node distribution based on node degree enhances the consistency of the partitioning across different graphs. For example, given two graphs G_A and G_B, we jointly partition them into two subgraph pairs $\{G_A^1, G_B^1\}$ and $\{G_A^2, G_B^2\}$. If we use uniform node distributions, the barycenter will be initialized with uniform node distribution $[0.5, 0.5]^\top$ and adjacency matrix $0.5\boldsymbol{I}_2$, and we may have an identification problem — G_B^2 can be finally paired with G_A^1.

B.2 Performance on some challenging cases

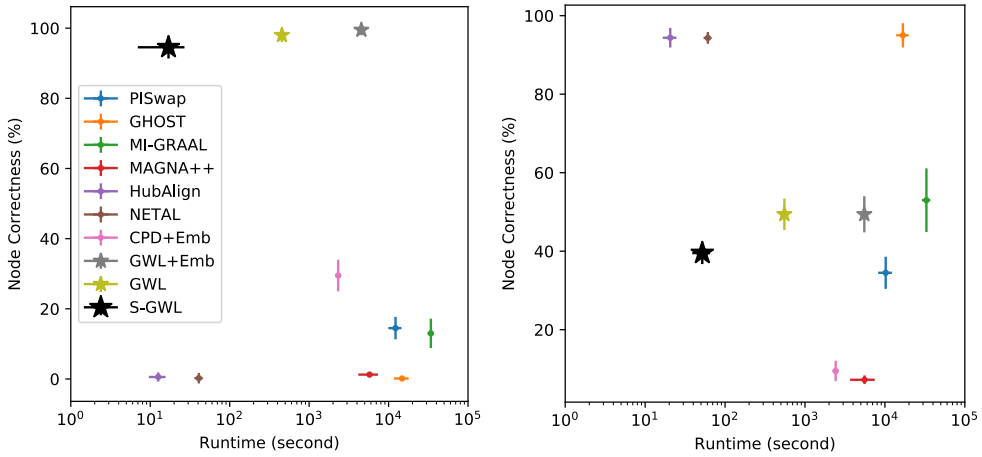
Although our GWL framework and S-GWL method perform well in most of our experiments, we find some challenging cases that point out our future research direction.

Matching Barabási-Albert (BA) graphs Figure 1(e) shows the averaged matching results in 10 trials. In five of these trials, we match synthetic graphs obeying to Gaussian random partition model. In the remaining five trials, we match synthetic graphs obeying to Barabási-Albert (BA) model. The overall performance shown in Figure 1(e) demonstrates the superiority of our S-GWL method. This outstanding result is mainly contributed by the experiments on Gaussian partition graphs. Specifically, when matching Gaussian partition graphs, all the GW discrepancy-based methods achieves very high node correctness, and the speed of our method is almost the same with the fastest HubAlign method, as shown in Figure 4(a). When it comes to BA graphs, Figure 4(b) indicates that although GW discrepancy-based methods still outperform many baselines, there is a gap between them and the state-of-the-art methods in the aspect of node correctness.

Additionally, the BA graphs also have a negative influence on our recursive mechanism. For Gaussian partition graphs, it is relatively easy to partition them into several sub-graphs with comparable size. In such a situation, the power of our recursive mechanism can be maximized, which helps us achieve over 100 times acceleration. However, for BA graphs, the sub-graphs we get are often with incomparable size. The largest sub-graph decides the runtime of our S-GWL method. As a result, our S-GWL method only achieves about 10$\sim$20 times acceleration.

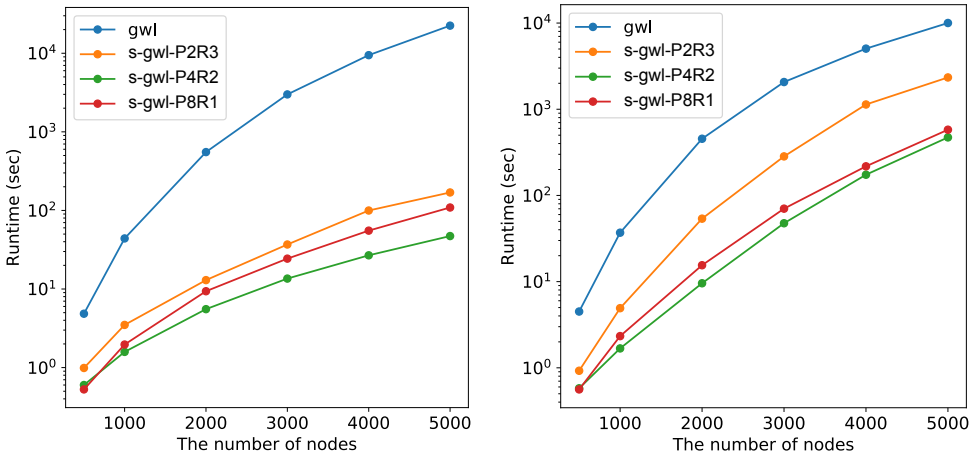
Currently, we are making efforts to improve the performance and the speed of our method on BA graphs. To solve this problem, we may need to use some node information, $e.g.$, introducing node embedding into our S-GWL method.

Matching incomparable graphs The second challenging case is matching incomparable graphs. This case is common in the field of bioinformatics, $e.g.$, matching the PPI networks from different species. When the networks are with incomparable size, the performance of GW discrepancy-based methods degrades. For example, in Table 7, we match the PPI network of yeast to that of human. This yeast network has 2,340 proteins (nodes), while the human network has 9,141 proteins. Because the ground truth correspondence between these proteins is unknown, we use edge correctness to evaluate



(a) Gaussian Partition: Accuracy v.s. efficiency

(b) Barabási-Albert: Accuracy v.s. efficiency



(c) Gaussian Partition: Acceleration

(d) Barabási-Albert: Acceleration

Figure 4: The performance of our method on different kinds of graphs. (a, b) For each method, its standard deviation of node correctness and that of runtime are shown as well.

Table 7: Comparisons for graph matching methods on edge correctness (%).

Method	IsoRank	PISwap	MI-GRAAL	GHOST	NETAL	HubAlign	GWL	S-GWL
Yeast↔Human	2.12	2.16	13.87	17.04	28.65	21.59	19.56	18.89

The results of baselines are from [19].

our method. Specifically, edge correctness calculates the percentage of yeast's edges appearing in the human network.

Experimental results show that both GWL and S-GWL outperform most of their competitors except HubAlign and NETAL. The main reason for this phenomenon, in our opinion, is because the constraint of optimal transport. The constraint $\boldsymbol{T} \in \Pi(\boldsymbol{\mu}_s, \boldsymbol{\mu}_t)$ implies that each node in the target graph is assigned to a source node with a probability as long as its probability in $\boldsymbol{\mu}_t$ is nonzero. When the number of target nodes is much larger than that of source nodes, the real correspondence will be oversmoothed because each source node transports to too many target nodes. To overcome this issue, we need to propose a preprocess to remove potentially-useless nodes from the large graph, which is another future work for us.