Pygmtools: A Python Graph Matching Toolkit

Runzhong Wang
Ziao Guo
Wenzheng Pan
Jiale Ma
Yikai Zhang
Nan Yang
Qi Liu
Longxuan Wei
Hanxue Zhang
Chang Liu
Zetian Jiang
Xiaokang Yang
Junchi Yan*

CSE Department and MoE Key Lab of AI, Shanghai Jiao Tong University, Shanghai, 200240, China

Editor: Alexandre Gramfort

Abstract

Graph matching aims to find node-to-node matching among multiple graphs, which is a fundamental yet challenging problem. To facilitate graph matching in scientific research and industrial applications, pygmtools is released, which is a Python graph matching toolkit that implements a comprehensive collection of two-graph matching and multi-graph matching solvers, covering both learning-free solvers as well as learning-based neural graph matching solvers. Our implementation supports numerical backends including Numpy, PyTorch, Jittor, Paddle, runs on Windows, MacOS and Linux, and is friendly to install and configure. Comprehensive documentations covering beginner’s guide, API reference and examples are available online. pygmtools is open-sourced under Mulan PSL v2 license.

Keywords: graph matching, combinatorial optimization, graph learning, python toolkit

1. Introduction and motivations

Graph matching (GM) is the NP-hard combinatorial optimization problem of finding the node-to-node matching between two graphs or more. GM is recognized for its history over fifty years (Emmert-Streib et al., 2016), and has attracted attention again in the deep learning era, e.g., Zanfir and Sminchisescu (2018) received best paper honorable mention in CVPR. Since graphs are ubiquitous, the application of GM have been vastly explored: in machine learning, Liu et al. (2022) aligns neural networks via GM to improve federated learning; in computer vision, He et al. (2021) improves multi-object trackers by viewing adjacent frames as graphs, Sarlin et al. (2020) tackles general image matching by exploiting the underlying graph structures, Fu et al. (2021); Ling and Qin (2022) align point clouds

* Correspondence author.

©2024 Runzhong Wang, Ziao Guo, Wenzheng Pan, Jiale Ma, Yikai Zhang, Nan Yang, Qi Liu, Longxuan Wei, Hanxue Zhang, Chang Liu, Zetian Jiang, Xiaokang Yang and Junchi Yan.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v25/23-0572.html.
Figure 1: Showcasing pygmtools in two-graph matching, multi-graph matching, and deep learning tasks, with the support of switching among multiple numerical backends.

by deep GM; in computer graphics, Dym et al. (2017) adopt GM for shape registration. However, GM is also found to be useful for data retrieval (Blumenthal et al., 2020), and malware detection (Li et al., 2019). In science domains, Guo et al. (2022) apply GM to understand the shape of brain arterial network, Luo et al. (2021) integrate GM in the deep learning pipeline of drug discovery. There are two reasons for the success of GM. On the one hand, GM explicitly integrates both node and edge features under a well-defined math formulation (Lawler, 1963), while other popular matching methods (e.g. Kuhn (1955); Fischler and Bolles (1981)) only consider the node features. On the other hand, the recent achievements in neural graph matching (Wang et al., 2020; Qin et al., 2022; Wang et al., 2022, 2023c) show the feasibility of integrating GM into deep learning pipelines.

Despite the merits, implementing GM algorithms can be challenging, particularly when striving for high efficiency and the off-the-shelf incorporation with deep learning. With a growing interest of GM from the research community, we are aware of the necessity of an easily-accessible high-quality GM toolkit covering the most popular GM solvers. Python is supported because it is the de facto language for deep learning and widely used in scientific computing, with a well-developed ecosystem, e.g., Paszke et al. (2017); Hu et al. (2020).

As shown in Figure 1, we integrate state-of-the-art GM methods in our toolkit for a range of scenarios: from matching two graphs and multiple graphs to neural GM that seamlessly complement other neural networks. It is worth mentioning that multi-graph matching is a more challenging task than two-graph matching and it involves a different set of solvers. Our implementation is compatible with various deep learning and scientific computing backends, including Numpy, PyTorch, Jittor, Paddle, TensorFlow, and MindSpore. A consistent API is maintained across different GM methods and diverse backends. Under the Mulan PSL v2 license, our toolkit relies solely on open-source libraries. A beginner’s guide to GM is offered with numerous examples, encompassing general graph matching, subgraph discovery, image matching, and neural network fusion, among others.

2. Existing graph matching toolkits

Table 1 compares the key features of existing GM toolkits, where most of them are open-source code delivered with technical papers. ZAC_GM is the official implementation of a classic
Pygmtools: A Python Graph Matching Toolkit

<table>
<thead>
<tr>
<th>tool name reference</th>
<th>pygmtools (ours)</th>
<th>ThinkMatch (Wang et al., 2021)</th>
<th>LPMP (Rolínek et al., 2020)</th>
<th>ZAC_GM (Wang et al., 2020)</th>
<th>multiway (Wang et al., 2018)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#linear solvers</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>#classic solvers</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>#multi-graph solvers</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>#neural solvers</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>11.9×</td>
<td>4.1×</td>
</tr>
<tr>
<td>relative runtime</td>
<td>1×</td>
<td>-</td>
<td>-</td>
<td>11.9×</td>
<td>4.1×</td>
</tr>
<tr>
<td>deep learning support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GPU support</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>online doc</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>pip install</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>switch backends</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>programming language</td>
<td>Python</td>
<td>Python</td>
<td>C++/Python</td>
<td>Matlab</td>
<td>Matlab</td>
</tr>
</tbody>
</table>

Table 1: Comparison among different GM toolkits (code links available in reference).

GM solver (Wang et al., 2020), and multiway is the official code of a multi-graph solver (Wang et al., 2018). These two packages also include a collection of Matlab code from related previous methods. As a common limitation, they pay less attention to the accessibility of users (especially those who are new to GM) and are not actively maintained.

LPMP and ThinkMatch are the most relevant packages. LPMP implements several GM solvers by the same authors, covering various topics in GM (Swoboda et al., 2017, 2019; Rolínek et al., 2020). However, LPMP has imperfections in GPU support and accessibility. For example, the GPU implementation is limited for CNNs, and the solver modules are CPU-only. While it supports pip install, the requirement of configuring a local compiler often causes problems for users. ThinkMatch is an alternative to LPMP, offering more comprehensive neural GM solvers and better GPU support. Our design philosophy for pygmtools sets it apart from the two mentioned earlier. The goal is to offer a Python GM package that is user-friendly for both GM experts and beginners. Installing pygmtools is straightforward with a simple pip command, relieving users from the burden of environment configurations. Additionally, we support switching between different deep learning backends.

Table 1 also provides insights into the relative running times of pygmtools (PyTorch backend) compared to ZAC_GM (Matlab), using RRWM on graphs containing 50 nodes. Additionally, it is compared to multiway (Matlab) with GAMGM on 10 graphs comprising 250 nodes. It’s important to mention that Table 1 only includes a high-level comparison, and the performance may vary among different solvers and different backends. For more comprehensive information, readers can refer to the detailed online benchmark.1

3. Toolkit usage and design details

Essentially, pygmtools is designed as a user-friendly Python library for general audience of practitioners that are interested in applying graph matching in their downstream applications. GM can be divided into two steps: build the math form of GM and mathematically solving the GM problem. In the following example, denote A1, A2 as two adjacency matrices, vf1, vf2 as the node features, n1, n2 are their numbers of nodes, GM can be performed with the following lines of code.

```python
1. import numpy as np; from functools import partial
2. import pygmtools as pygm
```

When building the math form of GM, we transform node and edge features into the affinity matrix \( K \in \mathbb{R}^{n_1 \times n_2} \), where its diagonal elements are node-to-node affinities, off-diagonal elements are edge-to-edge affinities. The function `pygm.utils.build_aff_mat` builds the math form of GM, known as Quadratic Assignment Problem (QAP) (Lawler, 1963)

\[
\max \mathbf{vec}(X)^\top K \mathbf{vec}(X), \quad \text{s.t. } X \in \{0,1\}^{n_1 \times n_2}, \quad X_1 n_2 = 1_{n_1}, \quad X^\top 1_{n_1} \leq 1_{n_2}, \tag{1}
\]

with \( \mathbf{vec}(\cdot) \) we denote column-wise vectorization, \( 1_n \) is an \( n \)-length column vector with all 1s, and it is assumed that \( n_1 \leq n_2 \). This example shows a recommended practice of GM, where the GM problem is solved by the Reweighted Random Walk Matching (RRWM) solver (Cho et al., 2010), followed by a discretization step where the Hungarian algorithm (Kuhn, 1955) projects RRWM’s output (in the continuous domain) to a discrete matching solution.

**Multi-backend support.** In `pygmtools`, a user can select various numerical backends. It allows flexibility as the preferred backend can vary depending on different situations, considering computer compatibility, existing toolchains, and conventions. By default, Numpy serves as the backend, but users can configure other backends with a better support of GPU and deep learning. In our code design, we maintain a consistent front-end API, and the backend library is imported only when needed.

**Examples.** Several examples and notebooks are offered with `pygmtools` to presents typical applications of GM. GM solvers are illustrated on matching synthetic graphs, and representative real-world applications such as matching images (Zanfir and Sminchisescu, 2018; Wang et al., 2023a,b) and fusing deep neural networks (Liu et al., 2022).

4. **Conclusions and outlook**

In summary, we present `pygmtools`, a Python toolkit for graph matching. It’s accessible under the Mulan PSL v2 license, accompanied by extensive documentation and easy installation, making it suitable for a wide audience, especially for researchers and developers integrating graph matching into their projects. Our toolkit features a range of matching solvers, including linear, classic two-graph, multi-graph, and state-of-the-art deep learning solvers. These solvers are compatible with a wide range of numerical backends like Numpy, PyTorch, Jittor, Paddle, TensorFlow and MindSpore, enabling versatile applications in scientific research, education, and industry. The detailed guidance and practical examples can be found in our comprehensive online documentation. Moreover, `pygmtools` is a dynamic project, continually evolving and actively maintained. We eagerly encourage contributions and suggestions from the open-source community to enhance its capabilities further.
Acknowledgments

This work was partially supported by National Key Research and Development Program of China (2020AAA0107600), National Natural Science Foundation of China (62222607) and Shanghai Committee Science and Technology Project (22511105100). The authors Runzhong Wang, Zetian Jing, Chang Liu were also in part sponsored by Wen-Tsun Wu Honorary Doctoral Scholarship, AI Institute, Shanghai Jiao Tong University. We thank all community members, as well as all students attending the course AI3607 (in 2022&2023) at Shanghai Jiao Tong University for their valuable feedback and bug reports on this toolkit.

References


