Journal of Machine Learning Research 25 (2024) 1-7 Submitted 5/23; Revised 12/23; Published 1/24

Pygmtools: A Python Graph Matching Toolkit

Runzhong Wang RUNZHONG.WANG@SJTU.EDU.CN
Ziao Guo ZIAO.GUO@SJTU.EDU.CN
Wenzheng Pan PWZ1121@SJTU.EDU.CN
Jiale Ma HEATINGMA@SJTU.EDU.CN
Yikai Zhang 1065543666@SJTU.EDU.CN
Nan Yang ADVANCING-SHEEP@SJTU.EDU.CN
Qi Liu PUREWHITE@SJTU.EDU.CN
Longxuan Wei WEILONGXUAN189034@SJTU.EDU.CN
Hanxue Zhang ZHX_JIAXUE@SJTU.EDU.CN
Chang Liu ONLY-CHANGER@SJTU.EDU.CN
Zetian Jiang MAPLE_JZT@SJTU.EDU.CN
Xiaokang Yang XKYANG@SJTU.EDU.CN
Junchi Yan* YANJUNCHI@SJTU.EDU.CN
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.

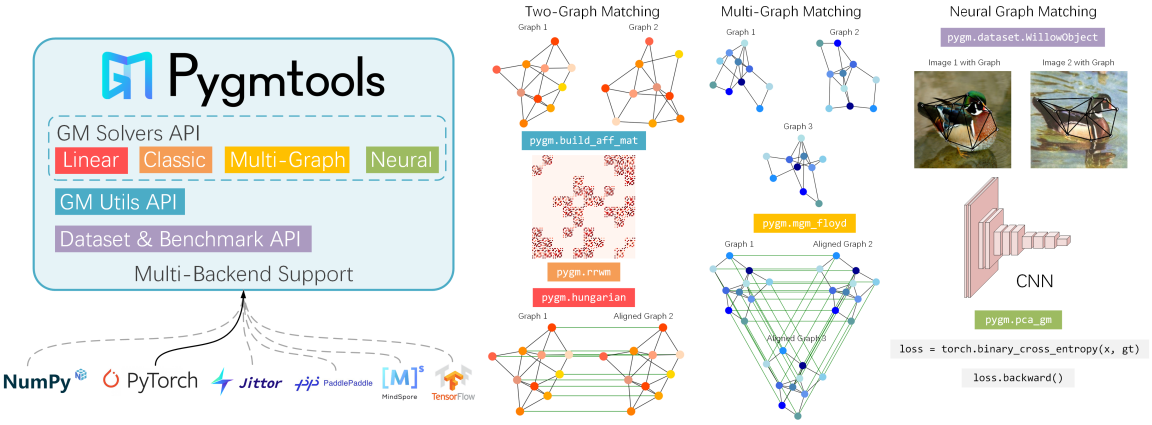

Figure 1: Showcasing `pygmtools` in two-graph matching, multi-graph matching, and deep learning tasks, with the support of switching among multiple numerical backends.

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

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

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.

```
import numpy as np; from functools import partial
import pygmtools as pygm
```

1. `https://pygmtools.readthedocs.io/en/latest/guide/benchmark.html#running-time-evaluation`

```
con1, ef1 = pygm.utils.dense_to_sparse(A1) # A1, vf1, n1 are graph 1 inputs
con2, ef2 = pygm.utils.dense_to_sparse(A2) # A2, vf2, n2 are graph 2 inputs
f = partial(pygm.utils.gaussian_aff_fn, sigma=1.) # affinity function
K = pygm.utils.build_aff_mat(vf1, ef1, con1, vf2, ef2, con2, edge_aff_fn=f)
    # build the math form of GM
S = pygm.rrwm(K, n1, n2)  # solve the GM problem
X = pygm.hungarian(S)  # to discrete solution
```

When building the math form of GM, we transform node and edge features into the affinity matrix $\mathbf{K} \in \mathbb{R}^{n_1 n_2 \times n_1 n_2}$, 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)

$$\max_{\mathbf{X}} \mathtt{vec}(\mathbf{X})^\top \mathbf{K} \mathtt{vec}(\mathbf{X}), \quad s.t.\ \mathbf{X} \in \{0,1\}^{n_1 \times n_2}, \mathbf{X}\mathbf{1}_{n_2} = \mathbf{1}_{n_1}, \mathbf{X}^\top \mathbf{1}_{n_1} \leq \mathbf{1}_{n_2}, \tag{1}$$

with $\mathtt{vec}(\cdot)$ we denote column-wise vectorization, $\mathbf{1}_n$ 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.

```
pygm.set_backend('pytorch') # set default backend globally
S = pygm.rrwm(pygm.utils.from_numpy(K), n1, n2) # S is a PyTorch Tensor
X = pygm.hungarian(S.numpy(), backend='numpy') # X is a Numpy array
```

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

David B. Blumenthal, Nicolas Boria, Johann Gamper, Sébastien Bougleux, and Luc Brun. Comparing heuristics for graph edit distance computation. *Int. J. Very Large Data Bases*, 29(1):419–458, 2020.

Minsu Cho, Jungmin Lee, and Kyoung Mu Lee. Reweighted random walks for graph matching. In *Eur. Conf. Comput. Vis.*, pages 492–505. Springer, 2010.

Nadav Dym, Haggai Maron, and Yaron Lipman. DS++: A flexible, scalable and provably tight relaxation for matching problems. *ACM Trans. Graph.*, 36(6):1–14, November 2017.

Frank Emmert-Streib, Matthias Dehmer, and Yongtang Shi. Fifty years of graph matching, network alignment and network comparison. *Info. Sci.*, 346:180–197, 2016.

Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Comm. of the ACM*, 24(6):381–395, 1981.

Kexue Fu, Shaolei Liu, Xiaoyuan Luo, and Manning Wang. Robust point cloud registration framework based on deep graph matching. In *Comput. Vis. Pattern Recog.*, pages 8893–8902, 2021.

Xiaoyang Guo, Aditi Basu Bal, Tom Needham, and Anuj Srivastava. Statistical shape analysis of brain arterial networks (BAN). *The Annals of Applied Statistics*, 16(2):1130–1150, 2022.

Jiawei He, Zehao Huang, Naiyan Wang, and Zhaoxiang Zhang. Learnable graph matching: Incorporating graph partitioning with deep feature learning for multiple object tracking. In *Comput. Vis. Pattern Recog.*, pages 5299–5309, 2021.

Shi-Min Hu, Dun Liang, Guo-Ye Yang, Guo-Wei Yang, and Wen-Yang Zhou. Jittor: a novel deep learning framework with meta-operators and unified graph execution. *Sci. China Info. Sci.*, 63(222103):1–21, 2020.

Harold W Kuhn. The hungarian method for the assignment problem. *Naval Res. Logistics Quart.*, 2(1-2):83–97, 1955.

E. L. Lawler. The quadratic assignment problem. *Management Sci.*, 9(4):586–599, 1963.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *Int. Conf. Mach. Learn.*, pages 3835–3845. PMLR, 2019.

Xiao Ling and Rongjun Qin. A graph-matching approach for cross-view registration of over-view and street-view based point clouds. *J. of Photogram. and Remote Sens.*, 185: 2–15, 2022.

Chang Liu, Chenfei Lou, Runzhong Wang, Alan Yuhan Xi, Li Shen, and Junchi Yan. Deep neural network fusion via graph matching with applications to model ensemble and federated learning. In *Int. Conf. Mach. Learn.*, pages 13857–13869. PMLR, 2022.

Shitong Luo, Chence Shi, Minkai Xu, and Jian Tang. Predicting molecular conformation via dynamic graph score matching. *Neural Info. Process. Systems*, 34:19784–19795, 2021.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Tianxiang Qin, Shikui Tu, and Lei Xu. IA-NGM: A bidirectional learning method for neural graph matching with feature fusion. *Mach. Learn.*, pages 1–27, 2022.

Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. code `https://github.com/LPMP/LPMP`. In *Eur. Conf. Comput. Vis.*, pages 407–424. Springer, 2020.

Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Comput. Vis. Pattern Recog.*, pages 4938–4947, 2020.

P. Swoboda, C. Rother, H.A. Alhaija, D. Kainmuller, and B. Savchynskyy. A study of lagrangean decompositions and dual ascent solvers for graph matching. In *Comput. Vis. Pattern Recog.*, pages 1607–1616, 2017.

Paul Swoboda, Ashkan Mokarian, Christian Theobalt, Florian Bernard, et al. A convex relaxation for multi-graph matching. In *Comput. Vis. Pattern Recog.*, pages 11156–11165, 2019.

Fudong Wang, Nan Xue, Jin-Gang Yu, and Gui-Song Xia. Zero-assignment constraint for graph matching with outliers. code `https://github.com/wangfudong/ZAC_GM`. In *Comput. Vis. Pattern Recog.*, pages 3033–3042, June 2020.

Qianqian Wang, Xiaowei Zhou, and Kostas Daniilidis. Multi-image semantic matching by mining consistent features. code `https://github.com/zju-3dv/multiway`. In *Comput. Vis. Pattern Recog.*, pages 685–694, 2018.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Combinatorial learning of robust deep graph matching: an embedding based approach. *Trans. Pattern Anal. Mach. Intell.*, pages 6984–7000, 2020.

Runzhong Wang, Ziao Guo, and Junchi Yan. ThinkMatch, 2021. URL `https://github.com/Thinklab-SJTU/ThinkMatch`.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Neural graph matching network: Learning lawler's quadratic assignment problem with extension to hypergraph and multiple-graph matching. *Trans. Pattern Anal. Mach. Intell.*, pages 5261–5279, 2022.

Runzhong Wang, Ziao Guo, Shaofei Jiang, Xiaokang Yang, and Junchi Yan. Deep learning of partial graph matching via differentiable top-k. In *Comput. Vis. Pattern Recog.*, pages 6272–6281, 2023a.

Runzhong Wang, Junchi Yan, and Xiaokang Yang. Unsupervised learning of graph matching with mixture of modes via discrepancy minimization. *Trans. Pattern Anal. Mach. Intell.*, pages 10500–10518, 2023b.

Runzhong Wang, Yunhao Zhang, Ziao Guo, Tianyi Chen, Xiaokang Yang, and Junchi Yan. Linsatnet: The positive linear satisfiability neural networks. In *Int. Conf. Mach. Learn.*, 2023c.

A. Zanfir and C. Sminchisescu. Deep learning of graph matching. In *Comput. Vis. Pattern Recog.*, pages 2684–2693, 2018.