Knowledge and Information Systems
https://doi.org/10.1007/s10115-022-01683-8

REGULAR PAPER

Autonomous graph mining algorithm search with best performance trade-off

Minji Yoon[1] · **Théophile Gervet**[1] · **Bryan Hooi**[2] · **Christos Faloutsos**[1]

Received: 29 January 2021 / Revised: 21 April 2022 / Accepted: 23 April 2022

Abstract
The pervasiveness of graphs today has raised the demand for algorithms to answer various questions: Which products would a user like to purchase given her order list? Which users are buying fake followers? Myriads of new graph algorithms are proposed every year to answer such questions—each with a distinct problem formulation, computational time, and memory footprint. This lack of unity makes it difficult for practitioners to compare different algorithms and pick the most suitable one for their application. These challenges create a gap in which state-of-the-art techniques developed in academia fail to be optimally deployed in real-world applications. To bridge this gap, we propose AUTOGM, an automated system for graph mining algorithm development. We first define a unified framework UNIFIEDGM for message-passing-based graph algorithms. UNIFIEDGM defines a search space in which five parameters are required to determine a graph algorithm. Under this search space, AUTOGM explicitly optimizes for the optimal parameter set of UNIFIEDGM using Bayesian Optimization. AUTOGM defines a novel budget-aware objective function for the optimization to find the best speed-accuracy trade-off in algorithms under a computation budget. On various real-world datasets, AUTOGM generates novel graph algorithms with the best speed/accuracy trade-off compared to existing models with heuristic parameters.

Keywords Graph mining · Graph neural networks · Neural architecture search · Automation · Unified framework · Optimization

✉ Minji Yoon
minjiy@cs.cmu.edu

Théophile Gervet
tgervet@andrew.cmu.edu

Bryan Hooi
bhooi@comp.nus.edu.sg

Christos Faloutsos
christos@cs.cmu.edu

[1] Carnegie Mellon University, Pittsburgh, USA

[2] National University of Singapore, Queenstown, Singapore

Published online: 21 May 2022

1 Introduction

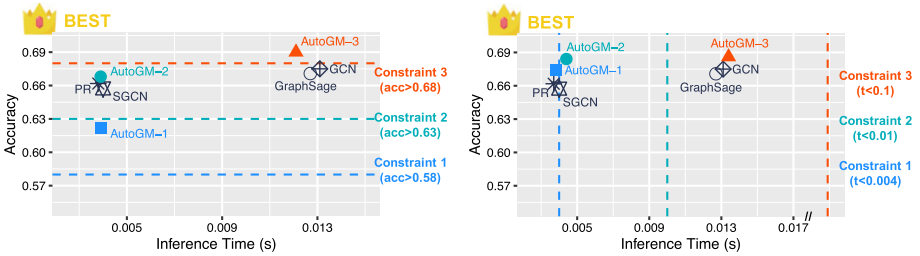
Many real-world problems are naturally modeled using graphs: who-buys-which-products in online marketplaces [45], who-follows-whom in social networks [27, 43], and protein relationships in biological networks [4, 36]. Graph mining provides solutions to practical problems such as classification of web documents [42, 49], clustering in market segmentation [33], recommendation in streaming services [2], and fraud detection in banking [5, 26].

A dizzying array of new graph mining algorithms is introduced every year to solve these real-world problems, giving rise to the question: Which algorithm should we choose for a specific application? Graph mining algorithms designed to solve the same task often have distinct conceptual formulations. Concretely, to estimate the similarity between two nodes—in social recommender systems for example—classical graph mining algorithms (like Personalized PageRank [1]) compute similarity scores by iterating a closed-form expression, while graph neural network algorithms [41] first learn node embeddings using deep learning, then estimate similarity scores with a distance metric in this embedding space. This lack of unity makes it hard for practitioners to determine which aspect of a method contributes to differences in computation time, accuracy, and memory footprint—significantly complicating the choice of the algorithm. Currently, selecting a graph mining algorithm suitable for a specific task among dozens of candidates is a resource-intensive process requiring expert experience and brute-force search.

To mitigate the cost and complexity of the algorithm selection process, the machine learning community has developed AutoML [15, 23]—which automates the process of algorithm selection and hyperparameter optimization. The success of AutoML depends on the size of the search space: it should be small enough to be tractable in a reasonable amount of time. However, AutoML techniques cannot be directly applied to graph mining because the hyperparameter search space is not even defined due to the lack of unity among graph mining algorithms.

Hence, in this paper, we first unify various graph mining algorithms under our UnifiedGM framework, then propose an automated system for graph algorithm development, AutoGM. We target graph algorithms that pass messages—propagate scores in the PageRank terminology [19, 27]—along edges to summarize the graph structure into nodes statistics. UnifiedGM manipulates five parameters of the message passing mechanism: the dimension of the communicated messages, the number of neighbors to communicate with (width), the number of steps to communicate for (length), the nonlinearity of the communication, and the message aggregation strategy. Different parameter settings yield novel graph algorithms, as well as existing algorithms, ranging from conventional graph mining algorithms (like PageRank) to graph neural networks.

Additionally, we introduce UnifiedGM- ext that extends UnifiedGM to embrace various attention and sampling methodologies in the message aggregation step. Recently, graph neural networks have adopted attention and importance sampling methodologies to improve their performance and scalability. The attention methodology computes importance/relevance scores of each neighbor with regard to a source node, then uses those scores as weights when we aggregate messages from the neighbors. Importance sampling goes one step further from attention and samples only neighbors with high relevance scores. By extending UnifiedGM to embrace attention and importance sampling concepts, we can apply techniques used by graph neural networks to the conventional graph mining field.



(a) Accuracy constraints on the Citeseer dataset

(b) Time constraints on the Citeseer dataset

Fig. 1 Method finds novel graph algorithms with the best accuracy/inference time trade-off on the node classification task: **a** Given three accuracy lower bounds (i.e., 0.58, 0.63, 0.68), AUTOGM generates three novel graph algorithms minimizing inference time. **b** Given three inference time upper bounds (i.e., 0.004, 0.01, 0.1 seconds), AUTOGM generates three novel graph algorithms maximizing accuracy

Based on UNIFIEDGM (and UNIFIEDGM- EXT), we propose an automated system for graph algorithm development, AUTOGM. AUTOGM leverages the parameter search space defined in UNIFIEDGM to address a practical problem: given a real-world scenario, what is the graph mining algorithm with the best speed/accuracy trade-off? In real-world scenarios, practitioners optimize performance under a computational budget [21, 22]. AUTOGM defines a novel budget-aware objective function capturing the speed/accuracy trade-off, then maximizes the objective function to find the optimal parameter set of UNIFIEDGM, resulting in a novel graph mining algorithm tailored for the given scenario.

The goal of our work is to empower practitioners without much expertise in graph mining to deploy algorithms tailored to their specific scenarios. The main contributions of this paper [1] are as follows:

- *Unification* UNIFIEDGM unifies various message-passing-based graph algorithms as instantiations of a message-passing framework with five parameters: dimension, width, length, nonlinearity, and aggregation strategy. UNIFIEDGM- EXT extends UNIFIEDGM with attention and sampling options in the message aggregation step.
- *Design space for graph mining algorithms* UNIFIEDGM provides the parameter search space necessary to automate graph mining algorithm development.
- *Automation* AUTOGM is an automated system for graph mining algorithm development. Based on the search space defined by UNIFIEDGM, AUTOGM finds the optimal graph algorithm using Bayesian optimization.
- *Budget awareness* AUTOGM maximizes accuracy of an algorithm under a computational time budget, or minimizes the computational time of an algorithm under a lower bound constraint on accuracy.
- *Effectiveness* AUTOGM finds novel graph mining algorithms with the best speed/accuracy trade-off compared to existing models with heuristic parameters (Fig. 1).

Table 1 gives a list of symbols and definitions.
Reproducibility Our code is publicly available [2].

[1] This paper is an extended version of [46].

[2] https://github.com/minjiyoon/ICDM20-AutoGM.

Table 1 Commonly used notation

Symbol	Definition
G	Input graph
n, m	Numbers of nodes and edges in G
A	$(n \times n)$ binary adjacency matrix of G
d_0	Dimension of input feature vectors
d	Dimension of communicated messages
k	Number of message passing steps
w	Number of neighbors sampled per node
l	Binary indicator for nonlinearity
a	Categorical aggregation strategy
X_0	$(n \times d_0)$ input feature vectors
X_i	$(n \times d)$ i-th layer message vectors $(i = 1 \ldots k)$
W_1	$(d_0 \times d)$ 1st layer transformation matrix
W_i	$(d \times d)$ i-th layer transformation matrix $(i = 2 \ldots k)$
$\phi(x)$	$\begin{cases} ReLU(x) \text{ if } l = \text{True} \\ x \text{ otherwise} \end{cases}$

2 Background and related work

2.1 AutoML

AutoML is the closest line of related work and the main inspiration for this paper. AutoML algorithms are developed to automate the process of algorithm selection and hyperparameter optimization in the machine learning community. The most closely related to our work in AutoML is Neural Architecture Search (NAS), which focuses on the problem of searching for the deep neural network architecture with the best performance. The search space includes the number of layers, the number of neurons, and the type of activation functions, among other design decisions. NAS broadly falls into three categories: evolutionary algorithms (EA), reinforcement learning (RL), and Bayesian optimization (BO).

EA-based NAS [7, 18, 24] explores the space of architectures by making a sequence of changes (inspired by evolutionary mutations) to networks that have already been evaluated. In RL-based NAS [51, 52], a recurrent neural network iteratively decides if and how to extend a neural architecture; the non-differentiable cost function is optimized with stochastic gradient techniques borrowed from the RL literature. Finally, BO-based NAS [15] models the cost function probabilistically and carefully determines future evaluations to minimize the total number of evaluated architectures. Since EA and RL-based NAS need to evaluate a vast number of architectures to find the optimum, these approaches are not ideally suited for neural architecture search [15]. On the other hand, BO emphasizes being cautious in selecting which architecture to try next to minimize the number of evaluations. As we discuss later, this makes BO suitable for our problem. In the following section, we give a brief description of Bayesian Optimization.

2.2 Bayesian optimization

Given a black-box objective function f with domain $\mathcal{X}$, BO sequentially updates a Gaussian Process prior over f. At time t, it incorporates results of previous evaluations $1, ..., t-1$ into a posterior $P(f|\mathcal{D}_{1:t-1})$ where $\mathcal{D}_{1:t-1} = \{x_{1:t-1}, f(x_{1:t-1})\}$. BO uses this posterior to construct an acquisition function $\phi_t(x)$ that is an approximate measure of evaluating $f(x)$ at time t. BO evaluates f at the maximizer of the acquisition function $x_t = \arg\max_{x \in \mathcal{X}} \phi_t(x)$. The evaluation $f(x_t)$ is then incorporated into the posterior $P(f|\mathcal{D}_{1:t})$, and the process is iterated.

The evaluation point x_t chosen by the acquisition function is an approximation of the maximizer of f. After T iterations, BO returns the parameter set of the maximum f among $x_{1:T}$. When choosing the point x_t to evaluate, the acquisition function $\phi_t(x)$ trades off exploration (sampling from areas of high uncertainty) with exploitation (sampling areas likely to offer an improvement over the current best observation). This cautious trade-off helps to minimize the number of evaluations of f. More details about BO can be found in [3].

These AutoML techniques cannot be directly applied to graph mining, as they require first formalizing autonomous algorithm selection as an optimization problem in a hyperparameter search space. Before UNIFIEDGM, the hyperparameter search space for graph mining was not even defined due to the lack of unity among algorithms. Hence, our proposed UNIFIEDGM allows the graph mining field to exploit state-of-the-art techniques developed in AutoML.

2.3 Graph neural architecture search

Various discussions [50] on graph neural architecture search have been initiated. [8] adopts the RL-based neural architecture search approach. [8] uses a recurrent neural network to generate variable-length strings that describe the architectures of graph neural networks, and trains the recurrent network with policy gradient to maximize the expected accuracy of the generated architectures on a validation data set. [47] focuses on designing general space for graph neural networks, that includes three crucial aspects of graph neural architecture design: intra-layer design, inter-layer design, and learning configuration. Based on this design space, [47] develops a controlled random search evaluation procedure to understand the trade-offs of each design dimension. [9, 35] and [48] focus on how to make the graph neural architecture search process more scalable. While most previous works focus on graph neural networks, we broaden the scope to embrace conventional graph mining algorithms such as PageRank [27], Pixie [6], and K-core [34]. We analyze why conventional graph mining algorithms and recent graph neural networks look unrelated at first glance and describe how they could be unified under one framework (Sect. 3.4).

3 Unified graph mining framework

In this section, we first motivate the message passing scheme (Sect. 3.1). We then propose our unified framework UNIFIEDGM (Sect. 3.2), explain how existing algorithms fit in the framework (Sect. 3.3), and further analyze how UNIFIEDGM bridges the conceptual gap between conventional graph mining and graph neural networks (Sect. 3.4). Finally, we outline how to choose parameters of UNIFIEDGM given a specific scenario (Sect. 3.5).

3.1 Message passing

A goal common to many graph mining algorithms is to answer queries at the node level (e.g., node clustering, classification, or recommendation) based on global graph information (e.g., edge structure and feature information from other nodes). To transmit the information necessary to answer such queries, in classical graph mining algorithms, nodes *propagate scalar scores to* their neighbors, while in graph neural networks, nodes *aggregate feature vectors from* their neighbors. In short, both families of algorithms pass messages among neighbors: scalars or vectors, inbound or outbound. The intuition behind these message passing algorithms is that whatever the task at hand, connectivity/locality matters: connected/nearby nodes are more similar (clustering), informative (classification), or relevant (recommendation) to each other than disconnected/distant nodes. Our unified framework targets graph algorithms that use the message passing mechanism.

3.2 UnifiedGM

We propose a unified framework UnifiedGM for graph mining algorithms that employ the message passing scheme. UnifiedGM defines the message passing mechanism based on five parameters:

- *Dimension* $d \in \mathbb{Z}_{>0}$ of passed messages. If $d = 1$, messages are scalar scores, otherwise they are d-dimensional embedding vectors.
- *Width* $w \in \mathbb{Z} \cup \{-1\}$ decides the number of neighbors each node communicates with. If $w = -1$, nodes communicate with all their neighbors.
- *Length* $k \in \mathbb{Z}$ decides the number of message passing steps.
- *Nonlinearity* $l \in \{\text{True}, \text{False}\}$ decides whether to use nonlinearities in the message passing or not.
- *Aggregation strategy* a decides if a node sends a message to itself and how to normalize the sum of incoming messages.

Figure 2 shows how each parameter regulates message passing under UnifiedGM.

The input of UnifiedGM is a graph $G = (V, E)$ and a matrix X_0 of size $(n \times d_0)$ containing d_0-dimensional initial node statistics for all n nodes—either scalar scores or feature vectors. Note that d_0 could be different from d, the dimension of the passed messages. The output of UnifiedGM is a set of d-dimensional node embeddings. These embeddings

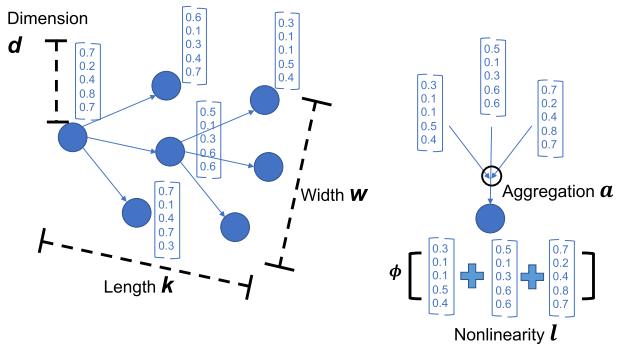


Fig. 2 UnifiedGM defines the message passing mechanism based on five parameters: the dimension d, length k, width w, nonlinearity l, and aggregation strategy a

Algorithm 1: UNIFIEDGM Algorithm

Require: initial node statistics X_0, binary adjacency matrix A
Ensure: node embeddings X_k
1: Initialize node statistics X_0
2: **for** message passing step $i = 1; i \leq k; i$++ **do**
3: Sample neighbors for each node: $A_{\text{samp}} \leftarrow Sample(A)$
4: Generate aggregation matrix: $A_{\text{agg}} \leftarrow Aggregate(A_{\text{samp}})$
5: Aggregate messages $X_i \leftarrow A_{\text{agg}} X_{i-1}$
6: Multiply with transformation matrix: $X_i \leftarrow X_i W_i$
7: Pass through nonlinear function: $X_i \leftarrow \phi(X_i) = \begin{cases} ReLU(X_i) \text{ if } l = \text{True} \\ X_i \text{ otherwise} \end{cases}$
8: **end for**
9: **return** X_k

contain information from the node's neighborhood and can be exploited in an output layer which is specialized to a given application (e.g., a logistic regression for node classification.)

Algorithm 1 outlines how UNIFIEDGM passes messages across a graph based on a set of five parameters (d, k, w, l, a). UNIFIEDGM first initializes node statistics (line 1), then iteratively passes messages among neighboring nodes k times. In the i-th message passing step, UNIFIEDGM randomly samples w neighbors to communicate with for each node (line 3) and aggregates messages from sampled neighbors with a strategy decided by the parameter a (lines 4 and 5). Then UNIFIEDGM transforms the aggregated messages linearly with a matrix W_i (line 6) and finally passes them through a function ϕ decided by the parameter l (line 7).

Let us explain in further detail the neighbor sampling and message aggregation steps. Neighbor sampling (line 3) can be expressed as generating a matrix $A_{\text{samp}} = Sample(A)$ by randomly zeroing out entries of the binary adjacency matrix A. Message aggregation (lines 4 and 5) is defined by the aggregation strategy $a \in \{\text{SA, SS, SN, NA, NS, NN}\}$. The first letter in {S, N} determines whether a node sends a message to itself or not (Self-loop or No-self-loop). The second letter in {A, S, N} determines how to normalize the sum of incoming messages (Asymmetric, Symmetric, or No-normalization). Each aggregation strategy a results in an aggregation matrix $A_{\text{agg}} = Aggregate(A_{\text{samp}})$, explained in Table 2. Multiplying messages X_{i-1} from the previous step by the matrix A_{agg} corresponds to aggregating messages from neighboring nodes.

Letting f_i denote the ith layer of message passing, we can summarize UNIFIEDGM as follows:

$$\begin{aligned} A_{\text{samp}} &= Sample(A) \\ A_{\text{agg}} &= Aggregate(A_{\text{samp}}) \\ X_k &= f_k(X_{k-1}) = \phi(A_{\text{agg}} X_{k-1} W_k) \\ &= f_k(f_{k-1}(\ldots f_1(X_0))) \end{aligned}$$

vX_0 is the $(n \times d_0)$ matrix of initial statistic vectors, X_i is the $(n \times d)$ matrix of statistic vectors at step i for $(i = 1 \ldots k)$. W_1 and W_i are $(d_0 \times d)$ and $(d \times d)$ transformation matrices, respectively $(i = 2 \ldots k)$.

Table 2 The aggregation strategy a decides if a node sends a message to itself or not (Self-loop or No-self-loop) and how to normalize the sum of incoming messages (Asymmetric, Symmetric, or No-normalization)

	Self-loop (S)	No-self-loop (N)
Asymmetric (A)	$D^{-1}(A + I_n)$	$D^{-1}A$
Symmetric (S)	$D^{-1/2}(A + I_n)D^{-1/2}$	$D^{-1/2}AD^{-1/2}$
No-normalization (N)	$(A + I_n)$	A

Each combination corresponds to an aggregation matrix $A_{\text{agg}} = Aggregate(A)$ in the table. Notation: n is the number of nodes in a graph, A is a $(n \times n)$ binary adjacency matrix, D is a $(n \times n)$ diagonal matrix where $D_{ii} = \sum_j A_{ij}$, and I_n is an identity matrix of size n

3.3 Reproduction of existing algorithms

In this section, we introduce the most popular graph mining algorithms exploiting the message passing scheme and show how they can be presented under UnifiedGM. Table 3 shows how to set initial node statistics and parameters (d, k, w, l, a) of UnifiedGM to reproduce the original graph algorithms.

PageRank [27] scores nodes in a graph based on their global relevance/importance, and was initially used by Google for webpage recommendation. PageRank initializes all n nodes in the graph with a score of $\frac{1}{n}$. Then, every node iteratively propagates its score across the graph with a decay coefficient $0 < c < 1$ to ensure convergence. Under UnifiedGM, PageRank propagates scalar scores ($d = 1$) to all neighbors ($w = -1$) with no nonlinear unit ($l =$ False) until scores have converged ($k = \infty$), and aggregates messages with no self-loop and asymmetric normalization ($a =$ NA). Note that the $(d \times d)$ transformation matrix W in UnifiedGM becomes a scalar value and corresponds to the decay coefficient c.

Personalized PageRank (PPR) [1] and **Random Walk with Restart** (RWR) [43, 44] build on PageRank to estimate the relevance of nodes in the perspective of a specific set of seed nodes thus enable personalized recommendation. Under UnifiedGM, the only difference of PPR/RWR from PageRank is the initial node scores: RWR/PPR place varying positive scores on the set of seed nodes and zero scores on others. PPR/RWR have the same set of (d, k, w, l, a) as PageRank.

Pixie [6], introduced by Pinterest, complements the ideas of PPR and RWR with neighbor sampling to deal with billions of nodes in real-time. Pixie fixes the number of message passing operations and stays within a computation budget. To reproduce this under UnifiedGM, Pixie fixes the product of k and w to a constant number (e.g., 2, 000 from [6]): after k is sampled, w is decided as $\frac{2{,}000}{k}$. Pixie has the same initial node statistics and parameter $d = 1, l =$ False, and $a =$ NA with PPR/RWR.

Graph Convolutional Networks (GCNs) [17] are a variant of Convolutional Neural Networks that operates directly on graphs. GCNs stack layers of first-order spectral filters followed by a nonlinear activation function to learn node embeddings. Under UnifiedGM, given node feature vectors as initial node statistics, GCN passes message vectors ($d = 64$) to all neighbors ($w = -1$) with nonlinear units ($l =$ True) across two-layered networks ($k = 2$) and aggregates messages with a self-loop and symmetric normalization ($a =$ SS).

Table 3 Graph mining algorithms can be fully reproduced under UNIFIEDGM with the respective initial node statistics and parameters (d, k, w, l, a)

Algorithm	Original message passing equation	Initial node statistics	d	k	w	l	a
PageRank	$X_k = c(D^{-1}A)X_{k-1}$	$\frac{1}{n}$ for all nodes	1	∞	-1	False	NA
Pixie	$X_k(u) = \sum_{v \in \mathrm{N}(u)} X_{k-1}(v)$	1 for seeds, 0 others	1	sample	$\frac{2000}{k}$	False	NA
GCN	$X_k = ReLU\left((D^{-\frac{1}{2}}(A + I_n)D^{-\frac{1}{2}})X_{k-1}W_k\right)$	feature vectors	64	2	-1	True	SS
GraphSAGE	$X_k(u) = ReLU\left(\frac{1}{\lvert\mathrm{N}(u)\rvert+1} \sum_{v \in \mathrm{N}(u) \cup u} X_{k-1}(v)W_k\right)$	feature vectors	64	2	25	True	SA
SGCN	$X_k = D^{-\frac{1}{2}}(A + I_n)D^{-\frac{1}{2}}X_{k-1}W_k$	feature vectors	64	2	-1	False	SS

n is the number of nodes, A denotes an $(n \times n)$ binary adjacency matrix, D denotes an $(n \times n)$ diagonal matrix where $D_{ii} = \sum_j A_{ij}$, I_n denotes an identity matrix of size n, $\mathrm{N}(u)$ denotes the set of sampled neighbors of node u, and $0 < c < 1$ is a decay coefficient. For PageRank, see the formulation given in [43]

GraphSAGE [10] extends GCN with neighbor sampling. GraphSage with a mean aggregator averages statistics of a node and its sampled neighbors. Under UNIFIEDGM, GraphSAGE-mean has the same parameters as GCN except w and a. GraphSAGE-mean samples a fixed number of neighbors to communicate with ($w = 25$) and normalizes the aggregated messages asymmetrically ($a = \text{SA}$).

Simplified GCN (SGCN) [40] reduces the excess complexity of GCN by removing the nonlinearities between GCN layers and collapsing the resulting function into a single linear transformation. With fewer parameters to train, SGCN is computationally more efficient than GCN but shows comparable performance on various tasks. Under UNIFIEDGM, SGCN has the same parameters with GCN except l. SGCN does not use any nonlinear unit ($l = \text{False}$).

Table 3 presents the original message passing equations of the existing graph algorithms. Those equations can be fully reproduced from Algorithm 1 with the proper initial node statistics and parameter sets listed in Table 3.

Here, we introduce two more graph algorithms that are unified under UNIFIEDGM with slight modifications: K-cores [34] and Belief Propagation [28] — two of the most popular graph mining algorithms. Although these algorithms do not contain trainable parameters and thus do not benefit from AUTOGM, they fit under UNIFIEDGM's message-passing framework. This shows that UNIFIEDGM is general enough to cover various graph mining algorithms.

- **K-core** [34] is the maximal subgraph in which every node is adjacent to at least k nodes. The most straightforward algorithm to compute k-cores is the so-called shaving method [32]: repeatedly deleting nodes with a degree less than k until no such node is left. The shaving method is presented in an iterative equation as follows:

$$x_{k+1} = \phi(A_k x_k - k\mathbf{1})$$

  where x_k is an indicator vector for k-cores where $x_k(i)$ is 1 when i-th node is part of k-cores, otherwise set to 0; A_k is the binary adjacency matrix where only edges among $x_k(i) = 1$ are set to 1, otherwise 0. When we multiply A_k with x_k, the output vector contains the degree of each node in k-cores. $\phi(x)$ is a nonlinear operation where $\phi(x) = 1$ when $x > 0$ else $\phi(x) = 0$. In $A_k x_k - k\mathbf{1}$, only nodes whose degree is higher than k have positive values. Thus, by passing $A_k x_k - k\mathbf{1}$ to $\phi(x)$, we output x_{k+1}, the indicator vector for $k + 1$-cores. Under UNIFIEDGM, k-cores propagates scalar scores ($d = 1$) to all neighbors ($w = -1$) with a nonlinear unit ($l = \text{True}$) k times, and aggregates messages with no self-loop and no normalization ($a = NN$). The slight modifications to UNIFIEDGM are that the adjacency matrices A_k are iteratively updated, and we perform a subtraction operation ($-k\mathbf{1}$) instead of the transformation operation (W). Note that the $(d \times d)$ transformation matrix W in UNIFIEDGM becomes the constant value 1.
- **Belief Propagation** (BP) [28] calculates the marginal belief distribution for unobserved nodes, conditional on any observed nodes' belief. FastBP [20] is one of the most widely used approximation algorithms for BP. While BP does not guarantee convergence, FastBP provides convergence in addition to speed and accuracy improvement. FastBP linearizes BP as follows:

$$[I + a'D - b'A]x = \phi_{BP}$$

  where I, D, and A denotes $n \times n$ identity, diagonal, and adjacency matrices, respectively; a' and b' are hyperparameters decided by the BP propagation matrix; x is the final belief vector and ϕ_{BP} is the prior beliefs. The equation is presented in an iteration equation as follows:

$$x = [b'A - a'D]x + \phi_{BP}$$

This iteration equation has the same form as PageRank. Under UNIFIEDGM, the only difference between FastBP and PageRank is the initial node scores and the aggregation strategy: FastBP sets the initial node scores X_0 with the prior beliefs ϕ_{BP} and does not normalize the aggregation but adds self-loop with coefficients $(a'D - b'A)$. FastBP has the same set of parameters (d, k, w, l) as PageRank.

3.4 Conventional GM vs. GNNs

As shown, conventional graph algorithms (e.g., PPR, RWR, Pixie) and recent GNNs are unified under UNIFIEDGM. However, before this work, these algorithms were not analyzed in the same framework. What has prevented them from being combined? Two main differences—the use of node feature information and trainability—are the culprits. While GNNs exploit additional node feature information and labels with semi-supervised learning, conventional graph algorithms do not. We analyze this apparent gap and show how UNIFIEDGM reconciles both families of algorithms.

Node feature information Conventional graph algorithms do not exploit node features, but instead, choose a set of seed nodes to initialize with scores suitable for a given application. Under UNIFIEDGM, these algorithms are also applicable with node features by maintaining the same values for parameters $(d = 1, k, w, l, a)$, but setting initial input dimension d_0 to be the input feature dimension and using a 1st layer transformation matrix W_1 of size $(d_0 \times 1)$. This would yield a new version of PageRank or PPR that exploits feature information.

Semi-supervised learning In GNNs, the transformation matrix W is trained with semi-supervised learning using node labels. On the other hand, conventional graph algorithms do not have a training phase in advance of an inference phase. However, conventional algorithms are trainable: the decay coefficient c in PageRank, PPR, and RWR corresponds to an (1×1) transformation matrix W under UNIFIEDGM. Because of its low dimension, the (1×1) transformation matrix could be set heuristically (e.g., $c = 0.85$ in PageRank). But we could use label information to train this (1×1) matrix W with gradient descent as we train it in GNNs.

In our experiments, we show how to train conventional algorithms (PageRank and Pixie) with feature information.

3.5 Parameter selection

We explain the effects of parameters (d, k, w, l, a) on the performance of graph algorithms and how to choose the proper parameters by illustrating the existing algorithm design.

- **Dimension** d: High dimensions of messages enrich the expressiveness of graph algorithms by sacrificing speed. If an application prioritizes fast and simple algorithms, scalar messages (e.g., $d = 1$ in Pixie) are suitable. In contrast, when applications prioritize rich expressiveness of messages and accuracy, high dimensional vectors (e.g., $d = 64$ in GNNs) are more appropriate.
- **Length** k: By deciding the number of message passing steps, k regulates the size of neighborhoods where a graph algorithm assumes locality — where nearby nodes are considered informative. For instance, GCNs assume that a small neighborhood is relevant $(k = 2)$. However, when there are label sparsity issues, GNNs propagate toward large

scopes ($k = 7$) to transmit label information from distant nodes. Large k results in a long computation time but does not guarantee a high accuracy.

- **Width** w: Large w lets algorithms aggregate information from more neighbors, leading to a possible increase in accuracy. At the same time, large w requires more message passing operations, resulting in longer computation time. In graphs with billions of nodes, like the Pinterest social network, small w is necessary to answer queries in real-time (as done by Pixie).
- **Nonlinearity** l: Nonlinearities enhance the expressiveness of graph algorithms at the cost of speed. They are suitable for anomaly detection systems that require high accuracy (e.g., GNNs for infection detection in medical applications). In contrast, omitting nonlinearity is appropriate for fast recommender systems in social networks (e.g., Pixie in Pinterest).
- **Aggregation strategy** a: The self-loop decides whether a node processes its own embedding during message passing. GNNs include a self-loop to complement a node's features with information from its neighborhood. Conversely, PageRank and RWR do not include a self-loop as they want to spread information from a source node to the rest of the graph to figure out the graph structure. Normalization prevents numerical instabilities and exploding/vanishing gradients in GNNs.

In our experiments, we explore how the five parameters affect the performance of graph algorithms empirically.

4 Extended UnifiedGM

We introduce UnifiedGM- ext that extends the message-aggregation step in UnifiedGM with two additional building blocks: attention and importance sampling. Recently, graph neural networks have been improved in various ways to improve their performance and scalability. These improvements focus on the aggregation step in the message passing mechanism. In the original form of the message passing mechanism, nodes pass/receive messages uniformly from their directly connected neighbors, assuming that these neighbors are informative. Questions have been raised about this assumption: are all neighbors informative enough to communicate with? In real-world graphs, few connections are made by mistake, and some are valid only for a specific application. For instance, in member-to-member networks in LinkedIn, connections could be made not only among colleagues but also among personal friends and families. When we apply the message passing mechanism on the LinkedIn network to make job recommendations, we aggregate information not only from the colleagues who are crucial information for the job recommendation task, but also from personal friends and families who are from different areas and often irrelevant. The motivations are summarized as follows: which neighbors are informative to pass/receive messages? and how trustworthy are they?

Attention and importance sampling methodologies are proposed on graph neural networks to handle these problems. The attention-based GNNs compute importance/relevance scores of each neighbor with regard to a source node, then use those scores as weights when they aggregate messages from the neighbors and compute a weighted sum of the aggregated messages. Importance sampling goes one step further from attention and samples only neighbors with high relevance scores. Importance sampling does not only handle different importance scores among neighbors, but also solves scalability issues by reducing the size of graphs through sampling.

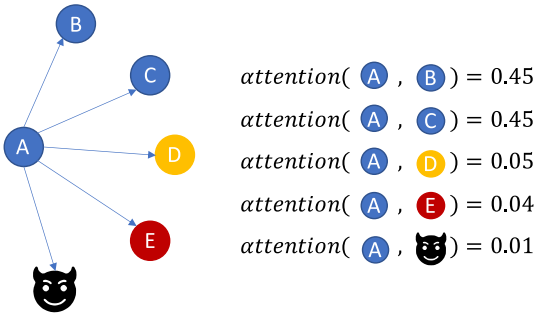


Fig. 3 UNIFIEDGM computes attention between a source node A and its neighbors based on their relevance to node A. Unrelated or adversarial neighbors have small attentions

As we described in Sect. 3.4, UNIFIEDGM unifies the conventional graph mining algorithms and graph neural networks. By extending UNIFIEDGM to embrace attention and importance sampling concepts, UNIFIEDGM- EXT allows applying techniques used by graph neural networks to the conventional graph mining field. To adopt attention into UNIFIEDGM- EXT, we add additional options to the aggregation parameter a. To apply importance sampling on UNIFIEDGM- EXT, we add a new parameter s that decides the sampling strategy. The following section shows how UNIFIEDGM- EXT embraces the concepts of attention and importance sampling concretely.

4.1 Attention

Graph attention networks (GAT) [37] is the first attention-based graph neural network model. It estimates the relevance between a source node and its neighbors using their hidden embeddings. Then the computed relevance scores (attentions) are used as weights in a weighted sum in the aggregation step. How to estimate relevance between two nodes or how to design the attention model varies across different methods. UNIFIEDGM- EXT extends the aggregation parameter a with new options: concatenation-based attention, dot-product-based attention, and low-pass attention. We describe how each attention model works under UNIFIEDGM- EXT.

- **Concatenation-based attention** proposed in [37] computes a relevance score $\alpha_l(i, j)$ between node i and j at the l-th layer as follows:

$$\alpha_l(i, j) = \frac{\sigma(a_{att} \cdot [h_l(i)||h_l(j)])}{\sum_{k \in N(i)} \sigma(a_{att} \cdot [h_l(i)||h_l(k)])}$$

  where a_{att} denotes a $(1 \times 2d)$ learnable parameter, $\sigma(x) = exp(LeakyReLU(x))$ is a nonlinear operation for attention computation, $h_l(i) = x_l(k)W_l$ denotes hidden embedding of node i at the l-th layer after multiplying with the transformation matrix W_l, and $N(i)$ denotes the neighbors of node i. Since we concatenate the hidden embeddings ($[h_l(i)||h_l(j)]$)), we name it as a concatenation-based attention model. Then we aggregate messages ($h_l(j)$) from neighbors using the computed attention $\alpha_l(i, j)$ as follows:

$$x_{l+1}(i) = \phi(\sum_{j \in N(i)} \alpha_l(i, j) h_l(j))$$

  where $\phi(x)$ is the operation decided by the nonlinearity parameter l in UNIFIEDGM (refer to Table 1). Then x_{l+1} is used as $(l + 1)$-th layer hidden embeddings.

- **Dotproduct-based attention** calculates a relevance score $\alpha_l(i, j)$ between node i and j as follows:

$$\alpha_l(i, j) = \frac{\sigma(W_{att} h_l(i) \cdot W_{att} h_l(j))}{\sum_{k \in N(i)} \sigma(W_{att} h_l(i) \cdot W_{att} h_l(k))}$$

  where W_{att} denotes a $(d_{att} \times d)$ learnable parameter which maps hidden embedding $h_l(i)$ from d-dimensional space to d_{att}-dimensional space. On the d_{att}-dimensional space, we compute the relevance score between node i and j by dot-producing their hidden embeddings.
- **Low-pass attention** reduces the impact of adversarial edge additions/deletions on graphs. To filter out adversarial nodes, [38] introduces a low-pass filter to GCNs that decrease weights of neighbors who are excessively different from a source node in the hidden embedding space. They define a low-pass attention as follows:

$$\beta_l(i, j) = \frac{R}{max(R, ||h_l(i) - h_l(j)||)}$$

$$\alpha_l(i, j) = \begin{cases} \beta_l(i, j)/d_i & \text{if } j \in N(i) \setminus \{i\} \\ 1 - \sum_{j \in N(i) \setminus \{i\}} \beta_l(i, j)/d_i & \text{if } j = i \\ 0 & \text{otherwise} \end{cases}$$

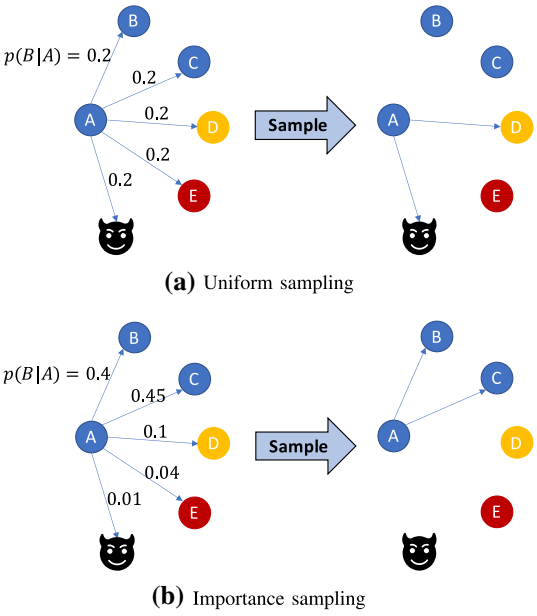
  where $R > 0$ is a threshold for controlling the low-pass message passing and $d_i = |N(i)|$ denotes the number of neighbors of node i. $\beta_l(i, j)$ assigns a weight of 1 if $h_l(i)$ and $h_l(i)$ are less than R apart, while gradually reducing the weight as the distance between them exceeds R. More distant from the source node i in the embedding space, the neighbor node j has a smaller weight $\beta_l(i, j)$. Then, the final attention $\alpha_l(i, j)$ acts as a low-pass filter to prevent the source node from being excessively affected by suspicious neighbors by giving small attentions.

In Sect. 3, the aggregation parameter a has six options (NN, NS, NA, SN, SS, SA from Table 2). By merely adding three more options (concatenation, dot product, low-filter attentions) to the parameter a, UNIFIEDGM- EXT successfully embraces attention-based models.

4.2 Importance sampling

In Sect. 3, we introduce uniform sampling in UNIFIEDGM- EXT where the sampling number is decided by the sampling parameter w. While uniform sampling resolves high computation and memory footprints problems by reducing the graph's size, it leads to a possible loss of crucial information. Uniform sampling does not discriminate informative neighbors from uninformative ones in the sampling process. Thus it could sample only uninformative or irrelevant neighbors for aggregation, resulting in low-quality embeddings. To deal with this limitation of uniform sampling, importance sampling has been adopted in GCNs. Importance sampling samples each neighbor following their distinct sampling probabilities, and the sampling probabilities are computed based on their relevance with regard to a source node.

We expand UNIFIEDGM- EXT with new parameter s that regulates the message passing mechanism's sampling strategy. The sampling strategy decides (1) how to define sampling probabilities, (2) whether the sampling probabilities are learnable or not, and (3) when to sample neighbors. We describe each component of the sampling strategy and its effect on the sampling process in UNIFIEDGM- EXT.



(a) Uniform sampling

(b) Importance sampling

Fig. 4 Uniform sampling samples informative and uninformative neighbors with an equal probability. It could sample only uninformative or irrelevant neighbors. On the other hand, importance sampling samples each neighbor following their distinct sampling probabilities, and the sampling probabilities are computed based on their relevance with regard to a source node

- *How to define sampling probabilities* Sampling probability $p(j|i)$ of a neighbor node j given a source node i could be computed individually or by a shared function with other edges. *Individual* sampling probability $p(j|i)$ is given as a scalar value that is decided independently from other edges' sampling probabilities. The only requirement is the sum should be 1 ($\sum_{j \in N(i)} p(j|i)$). The *individual* sampling probabilities could be all same (uniform sampling, $p(j|i) = 1/|N(i)|$) or proportional to their degree as follows:

$$p(j|i) = \frac{|N(j)|}{\sum_{k \in N(i)} |N(k)|}$$

  where $N(i)$ denotes the degree of node i. The *shared function* $p(j|i)$ could have various forms including the three attention models we described above. While *individual* sampling probabilities are intuitive and easy to interpret, *shared* sampling probability functions share a small number of parameters, thus less likely to be overfitted to the training set.
- *Learnability of sampling probabilities* The sampling probabilities defined as either individual scalar values or a shared function could be set heuristically and fixed to the initial values. They could also be trained by gradient back-propagation. While *heuristic* sampling probabilities are cost-efficient without additional sampling probability training, *learnable* sampling probabilities are customized to a given application, leading to a performance improvement.
- *When to sample neighbors: Static sampling* samples neighbors of each node before the message passing mechanism. In *static sampling*, a set of sampled neighbors is fixed during the message passing mechanism. Thus nodes keep interacting with the same set of their sampled neighbors in every layer. On the other hand, *dynamic sampling* samples a new set of neighbors every time a source node is engaged in the message passing mechanism as described in Algorithm 1. In *dynamic sampling*, nodes receive/pass messages with a new set of the sampled neighbors at each layer. While *static sampling* reduces the computation

Table 4 UnifiedGM- ext defines a sampling strategy based on (1) where the sampling probabilities are learnable, (2) how the sampling probabilities are designed, and 3) when the sampling is executed

Learnability	Sampling probability model form	Sampling timing
Heuristic	Uniform distribution	Static
		Dynamic
	Proportional to degree	Static
		Dynamic
Learnable	Concatenation-based attention model	Static
		Dynamic
	Dot-product-based attention model	Static
		Dynamic
	Low-pass filter attention model	Static
		Dynamic

time by running the sampling process only once, *dynamic sampling* increases accuracy. Randomness in the *dynamic sampling* brings a regularization effect, which helps with generalization.

In Table 4, the sampling parameter s has 5 (two heuristics and three shared models) $\times 2$ (heuristic/learnable) $\times 2$ (static/dynamic) $= 20$ options. With the addition of the parameter s, UnifiedGM- ext now has six parameters (d, w, k, l, a, s) to define a graph mining algorithm based on a message-passing mechanism. Users decide whether to append new attention options to the parameter a and add a sampling parameter s to UnifiedGM. This section showed UnifiedGM- ext is general enough to embrace new approaches with very few modifications. The following section about an automated system for graph algorithm development is based on the original UnifiedGM from Sect. 3. However, the number of parameters or the number of options for each parameter does not affect the algorithm we describe in the next section.

5 Automation of graph mining algorithm development

With the proper parameter selection, UnifiedGM could output a graph algorithm tailored for a specific application. However, the parameter selection process still relies on the intuition and domain knowledge of practitioners, which would prevent non-experts in graph mining from fully exploiting UnifiedGM. How can we empower practitioners without much expertise to deploy customized algorithms? We introduce AutoGM, which generates an optimal graph algorithm autonomously given a user's scenario.

When designing an algorithm for an application, we need to consider two primary metrics: computation time and accuracy, which usually trade off each other. Take, for example, a developer who aims to develop an online recommender system that makes personalized recommendations to a large number of users at the same time. At first, she employs a state-of-the-art GNN model (in terms of accuracy) but finds that the computation time is too long for her application. Then the developer seeks an alternative simple graph algorithm that runs faster than a time budget by sacrificing accuracy. AutoGM incorporates this practical issue of finding the best speed-accuracy trade-off into the graph algorithm generation problem. AutoGM answers two questions: (1) given the maximum acceptable computation time,

which graph algorithm maximizes accuracy? (2) given minimum accuracy requirements, which graph algorithm minimizes computation time?

We first formalize our budget-aware graph algorithm generation problem as a constrained optimization problem. Then we replace the constrained problem with an unconstrained optimization problem using barrier methods (Sect. 5.1). We explain why Bayesian optimization is well-suited for this unconstrained problem (Sect. 5.2). Then we describe how AUTOGM solves the optimization problem using Bayesian optimization (Sect. 5.3). Finally, we analyze the time complexities of AUTOGM (Sect. 5.4).

5.1 Budget-aware objective function

Letting x denote a graph algorithm, $g(x)$ and $h(x)$ indicate the computation time and accuracy of x, respectively. Then an optimal graph algorithm generation problem with an accuracy lower bound $h_{\min}$ is presented as a constrained optimization as follows:

$$x_{opt} = \text{argmin}_x \; g(x) \text{ subject to } h(x) - h_{\min} \geq 0 \tag{1}$$

One of the common ways to solve a constrained optimization problem is using a barrier method [29], replacing inequality constraints by a penalizing term in the objective function. We re-formulate the original constrained problem in Eq. 1 as an equivalent unconstrained problem as follows:

$$x_{opt} = \text{argmin}_x \; g(x) + I_{h(x)-h_{\min}\geq 0}(x) \tag{2}$$

where the indicator function $I_{h(x)-h_{\min}\geq 0}(x) = 0$ if $h(x) - h_{\min} \geq 0$ and ∞ if the constraint is violated. Equation 2 eliminates the inequality constraints, but introduces a discontinuous objective function, which is challenging to optimize. Thus we approximate the discontinuous indicator function with an optimization-friendly log barrier function. The log barrier function, defined as $-\log(h(x) - h_{\min})$ is a continuous function whose value on a point increases to infinity ($-\log 0$) as the point approaches the boundary $h(x) - h_{\min} = 0$ of the feasible region. Replacing the indicator function with the log barrier function yields the following optimization problem:

$$f_{GM}(x) = g(x) - \lambda \log(h(x) - h_{\min}) \tag{3}$$

$$x_{opt} = \text{argmin}_x \; f_{GM}(x) \tag{4}$$

f_{GM} is our novel budget-aware objective function and $\lambda > 0$ is a penalty coefficient. Equation 4 is not equivalent to our original optimization problem, Eq. 1. However, as λ approaches zero, it becomes an ever-better approximation (i.e., $-\lambda \log(h(x) - h_{\min})$ approaches $I_{h(x)-h_{\min}\geq 0}(x)$) [29]. The solution of Eq. 4 ideally converges to the solution of the original constrained problem. Now, our budget-aware graph algorithm generation problem is formulated as a minimization problem of f_{GM}.

Given a minimum accuracy constraint $acc_{\min}$, we set $g(x) = time$ to minimize and $h(x) - h_{\min} = acc - acc_{\min} \geq 0$ as a constraint. On the other hand, given a maximum inference time constraint $time_{\max}$, we want to maximize accuracy while observing the time constraint. Then we set $g(x) = -acc$ to minimize and $h(x) - h_{\min} = time_{\max} - time \geq 0$ as a constraint.

Algorithm 2: AUTOGM Algorithm

Require: minimum accuracy (or maximum inference time) constraint, target dataset, BO search budget
Ensure: a graph algorithm (i.e., five parameters of UNIFIEDGM)
1: **for** iteration $i = 1$; $i <$ BO search budget; i++ **do**
2: Choose a point (d, k, w, l, a) to evaluate
3: Generate a graph mining algorithm A from (d, k, w, l, a)
4: Train A on the training set
5: Evaluate A and measure $acc, time$ on the validation set
6: Evaluate $f_{GM}(acc, time)$ and update posterior of f_{GM}
7: **end for**
8: **return** a parameter set with the minimum f_{GM}

5.2 Bayesian optimization

Under UNIFIEDGM, a graph algorithm x is defined by a set of parameters (d, k, w, l, a). Then search space $\mathcal{X}$ for the optimization problem becomes a five-dimensional space of parameters (d, k, w, l, a). Suppose we set cardinalities for each parameter as 300, 30, 50, 2, and 6, respectively (i.e., $0 < d \in \mathbb{Z} \leq 300, 0 < k \in \mathbb{Z} \leq 30, 0 < w \in \mathbb{Z} \leq 50, l \in \{True, False\}, a \in \{NA, NS, NN, SA, SS, SN\}$). Then the number of unique architectures within our search space is $300 \times 30 \times 50 \times 2 \times 6 = 5.4 \times 10^6$, which is quite overwhelming. Moreover, training and validating a graph algorithm, especially on large datasets, takes significant time. Thus it is impractical to search the space $\mathcal{X}$ exhaustively. Most importantly, even if we could measure the computation time and accuracy ($g(x)$ and $h(x)$) of a graph algorithm and calculate the objective function $f_{GM}(x) = g(x) - \lambda \log(h(x) - h_{\min})$, we do not know the exact closed-form of $f_{GM}(x) = f_{GM}(d, k, w, l, a)$ in terms of the parameters (d, k, w, l, a) nor its derivatives. Thus, we cannot exploit classical optimization techniques that use derivative information. To cope with these problems—expensive evaluation and no closed-form expression nor derivatives—which optimization technique is appropriate?

Bayesian optimization (BO) [3] is the most widely-used approach to find the global optimum of a black-box cost function—a function that we can evaluate but for which we do not have a closed-form expression or derivatives. Also, BO is cost-efficient with as few expensive evaluations as possible (more details in Sect. 2.2). Therefore, BO is well-suited to our problem to find the best parameter set (d, k, w, l, a) given the expensive black-box objective function $f_{GM}(x)$.

5.3 AUTOGM

Users supply three inputs to AUTOGM: (1) a budget constraint (the minimum accuracy or maximum computation time), (2) a target dataset on which they want an optimized algorithm—containing a graph, initial node scores, and labels for supervised learning—and (3) a search budget for Bayesian Optimization. The search budget is given as the total number of evaluations in BO. Then AUTOGM outputs the optimal graph mining algorithm (i.e., parameter set of UNIFIEDGM).

Algorithm 2 outlines how AUTOGM works. Until it has exhausted its search budget, AUTOGM repeats the process: (1) Pick a point $x = (d, k, w, l, a) \in \mathcal{X}$ to evaluate using an acquisition function of BO (line 2) then generate a graph algorithm A from parameters (d, k, w, l, a) (line 3). (2) Train A on the training set (line 4) and measure accuracy and inference time of A on the validation set (line 5). (3) Evaluate the objective function f_{GM}

given the accuracy and inference time of A, then update a posterior model for f_{GM} in BO (line 6). After all iterations, AUTOGM returns the parameter set $x = (d, k, w, l, a)$ with the minimum f_{GM} among the evaluated points.

The search space of AUTOGM is not affected by the input but fixed to a five-dimensional space of parameters (d, k, w, l, a). The search time of AUTOGM is determined by the BO search budget (total number of evaluations) and evaluation time. Since the evaluation time of a graph algorithm is often proportional to the input dataset's size, the total search time of AUTOGM is decided by the dataset. BO's minimization of the number of evaluations is especially efficient for large datasets which result in the long evaluation time. Our main contribution is defining the graph algorithm generation problem as an optimization problem on a novel search space.

5.4 Time complexity analysis

We analyze the time complexities of UNIFIEDGM and AUTOGM.

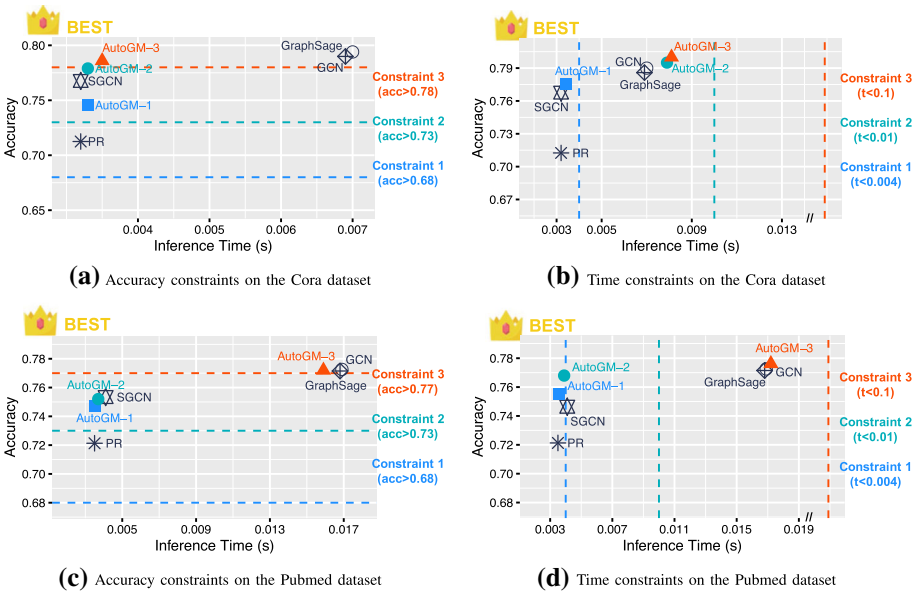
Theorem 1 *(Time Complexity of* UNIFIEDGM*) A graph mining algorithm A generated from* UNIFIEDGM *with a parameter set* (d, k, w, l, a) *takes* $O(kdwn)$ *time where n is the number of nodes in a given graph.*

Proof Under UNIFIEDGM, matrix-vector multiplication operations represent the bulk of the computation time. In the matrix-vector multiplication operations, the matrix corresponds to the adjacency matrix whose number of nonzeros is $O(wn)$. Under UNIFIEDGM, every node samples w neighbors, summing up to $O(wn)$ edges in the sampled graph. In the matrix-vector multiplication operations, the vector corresponds to node embeddings $X \in \mathbb{R}^{n \times d}$. Then one matrix-vector multiplication operation takes $O(dwn)$. Under UNIFIEDGM, the matrix-vector multiplication operation is executed k times across k layers. The nonlinear operation is decided by l and the normalization operation is decided by a take $O(n)$ time each. Thus the algorithm A takes $O(kdwn)$ time in total. □

In Theorem 1, we show the time complexity of the graph algorithm generated from UNIFIEDGM in terms of the parameters d, k, w, l, a. However, in Theorem 2, we express the time complexity of AUTOGM in different terms. Intuitively, the computation time of AUTOGM is proportional to the number of times we train a graph algorithm—i.e., evaluate a configuration (d, k, w, l, a)—times the time it takes to train the algorithm. To represent the time it takes to train an algorithm, we cannot rely on the parameters (d, k, w, l, a) as they keep changing while AUTOGM searches the space. Instead, we note that the computation time of one epoch of training is mainly decided by the size of the graph, which we represent by the number of edges m. The training time is then proportional to Em, where E is the number of training epochs. The total time to BEm where B is the number of times we train the algorithm (the number of evaluations allowed by the BO search budget).

Theorem 2 *(Time Complexity of* AUTOGM*)* AUTOGM *takes* $O(BEm)$ *for searching the optimal graph mining algorithm where m is the number of edges in a given graph, E is the number of epochs for the training, and B is the BO search budget.*

Proof The computation time of AUTOGM is proportional to the number of times we train a graph algorithm (evaluate a hyper-parameter configuration) times the time it takes to train the algorithm. The graph algorithm executes several adjacency matrix-embedding vector multiplication operations in the forward and backward pass, which take $O(m)$. This is repeated for



(a) Accuracy constraints on the Cora dataset

(b) Time constraints on the Cora dataset

(c) Accuracy constraints on the Pubmed dataset

(d) Time constraints on the Pubmed dataset

Fig. 5 AUTOGM finds the algorithms with the best accuracy/inference time trade-off on the node classification task: given three different accuracy/inference time constraints 1, 2, 3, AUTOGM generates three novel graph algorithms, AUTOGM-1, 2, 3, respectively

every batch (E times), for a total training time Em. Finally, AUTOGM trains the algorithm B times as described in Algorithm 2. Thus the overall computation time of AUTOGM is $O(BEm)$. □

6 Experiments

In this section, we evaluate the performance of AUTOGM compared to existing models with heuristic parameters. We aim to answer the following questions:

- *Q1. Effectiveness of AutoGM* Do algorithms found by AUTOGM outperform their state-of-the-art competitors? Given an upper bound on inference time/a lower bound on accuracy, does AUTOGM find the algorithm with the best accuracy/the fastest inference time? (Sect. 6.2)
- *Q2. Search efficiency of AutoGM* How long does AUTOGM take to find the optimal graph algorithm? How efficient it is compared to random search? (Sect. 6.3)
- *Q3. Effect of* UNIFIEDGM *parameters* How do parameters (d, k, w, l, a) affect the accuracy and inference time of a graph mining algorithm? (Sect. 6.4)

6.1 Experimental setting

We evaluate the performance of graph mining algorithms on two semi-supervised tasks, node classification and link prediction. All experiments were conducted on identical machines using the Amazon EC2 service (p2.xlarge with 4 vCPUs, 1 GPU and 61 GB RAM).

Table 5 Dataset statistics: AmazonC and AmazonP denote the Amazon Computer and Amazon Photo datasets, respectively

Dataset	Node	Edge	Feature	Label	Train/Val/test
Cora	2485	5069	1433	7	140/500/1000
Citeseer	2110	3668	3703	6	120/500/1000
Pubmed	19,717	44,324	500	3	60/500/1000
AmazonC	13,381	245,778	767	10	410/1380/12,000
AmazonP	7487	119,043	745	8	230/760/6650
CoauthorC	18,333	81,894	6805	15	550/830/15,950
CoauthorP	34,493	247,962	8415	5	1030/3450/30,010

CoauthorC and CoauthorP denote the MS Coauthor CS and Physics, respectively

Dataset We use the three citation networks (Cora, Citeseer, and Pubmed) [30], two Amazon co-purchase graphs (Amazon Computers and Amazon Photo) [31], and two co-authorship graphs (MS CoauthorCS and MS CoauthorPhysics) [31]. We report their statistics in Table 5.
Baseline Our baselines are PageRank [27], GCN [17], GraphSage [10], and SGCN [40]. We generate each algorithm under UNIFIEDGM by setting the five parameters as follows:

- PageRank: $d = 1, k = 30, w = -1, l = \text{False}, a = \text{NA}$
- GCN: $d = 64, k = 2, w = -1, l = \text{True}, a = \text{SS}$
- GraphSAGE: $d = 64, k = 2, w = 25, l = \text{True}, a = \text{SA}$
- SGCN: $d = 64, k = 2, w = -1, l = \text{False}, a = \text{SS}$

When w is larger than the number of neighbors, we sample neighbors with replacement. For PageRank, the original algorithm outputs the sum of intermediate scores that each node receives ($\sum X_i$), but we use only the final scores X_k in our experiments. The goal of our experiments is to compare PageRank with other algorithms in terms of its main feature in UNIFIEDGM, low dimension ($d = 1$).

Bayesian optimization: We use an open-sourced Bayesian optimization package[3]. For the parameters d, k, and w which take integer values, we round the real-valued parameters chosen by BO to integer values. For the parameter l and a, which take Boolean and categorical values, we bound the search space ($0 < l < 1$ and $0 < a < 6$), round the real-valued parameters chosen by BO to the closest integer values, and map (0: False, 1: True, 0: NN, 1: NS, 2: NA, 3: SN, 4: SS, 5: SA). We set the BO search budget (total number of evaluations) as 20 for all datasets. The resulting search time of each dataset is reported in Table 7. For the penalty coefficient λ, the smaller λ brings the tighter budget constraints. To make our budget constraints strict, we set λ as 10^{-19}.

We use the Adam optimizer [16] and tune each baseline with a grid search on each dataset. Most baselines perform best on most datasets with a learning rate of 0.01, weight decay of 5×10^{-4}, and dropout probability of 0.5. We fix these parameters in our autonomous graph mining algorithm search through Bayesian Optimization. We report the average performance across 10 runs for each experiment.

[3] https://github.com/fmfn/BayesianOptimization.

6.2 Effectiveness of AUTOGM

In this section, we demonstrate how AUTOGM trades off accuracy and inference time on real-world graphs with two different tasks, node classification and link prediction. We compare the best algorithms found by AUTOGM with baselines in terms of accuracy and inference time. For each dataset, we run AUTOGM with three different accuracy lower bounds and three inference time upper bounds, as illustrated in Figs. 1 and 5. For each constraint, AUTOGM generates a novel graph algorithm corresponding to a set of five parameters of UNIFIEDGM. For space efficiency, we show the result on the Cora, Citeseer, and Pubmed datasets.

6.2.1 Node classification

In the node classification task, each graph mining algorithm predicts the label of a given node. Among algorithms satisfying an accuracy lower bound, the algorithms generated by AUTOGM show the best trade-off between accuracy and inference time. For instance, in the Citeseer dataset in Fig. 1a, AUTOGM-2 has the fastest inference time above accuracy constraint 2 among PageRank (PR), GCN, SGCN, and GraphSage. Given the highest or tightest accuracy constraint 3, only AUTOGM-3 satisfies it. Conversely, among algorithms satisfying inference time upper bounds, the algorithms generated by AUTOGM have the highest accuracy. For instance, in the Pubmed dataset in Fig. 5d, AUTOGM-1 has the highest accuracy below time constraint 1 among PR and SGCN. Given the most generous time constraint 3, AUTOGM-3 achieves the highest accuracy among all algorithms.

The empirical performance of our baselines is consistent with our guidelines for how to choose the parameters (d, k, w, l, a) in Sect. 3.5. PageRank achieves fast inference time with a low dimension of messages ($d = 1$) and no nonlinearities ($l =$ False), but sacrifice accuracy. GCN and GraphSage achieve high accuracy with a high dimension of messages ($d = 64$) and nonlinearities ($l =$ True) at the cost of a high inference time. SGCN removes nonlinearities ($l =$ False) to decrease the inference time while maintaining high accuracy.

Table 6 shows the parameter set of UNIFIEDGM that corresponds to the algorithms found by AUTOGM on the Citeseer dataset. When encouraged to find higher accuracy algorithms (through a larger time upper bound or higher accuracy lower bound), AUTOGM is likely to use high values of d and w and nonlinearities ($l =$ True). For instance, AUTOGM chooses higher values $d = 255, w = 45$ for the larger time upper bound $time < 0.01$ than the values $d = 70, w = 25$ for the bound $time < 0.004$. With the largest upper bound $time < 0.1$, AUTOGM chooses $l = True$ to use nonlinearities. This result is consistent with our intuition over the parameter selection in Sect. 3.5. Vastly different parameter sets for each algorithm in Table 6 show that AUTOGM searches the parameter space beyond human intuition, which underlines the value of autonomous graph mining algorithm development.

6.2.2 Link prediction

In the link prediction task, the algorithm predicts whether there exists an edge between two given nodes. To build our training set, we randomly hide 30% of edges in the original graph, use the remaining edges as positive ground-truth labels, and sample an equal number of disconnected node pairs as negative ground-truth labels. Our test set consists of the hidden 30% edges as positive ground-truth labels and an equal number of random disconnected node pairs as negative ground-truth labels. After we get the node embeddings for a graph algorithm, we dot-product each pair of node embeddings to predict the probability of edge existence for the given node pair.

Table 6 Parameters corresponding to algorithms found by AUTOGM in Fig. 1

Dataset	Budget	d	k	w	l	a	Time	Acc
Citeseer	t<0.004	70	4	25	F	SA	0.0039	0.674
	t<0.01	255	4	45	F	SS	0.004	0.683
	t<0.1	68	1	47	T	SS	0.0134	0.686
	a>0.58	138	1	36	F	SA	0.0039	0.622
	a>0.63	25	4	54	F	NA	0.0039	0.665
	a>0.68	39	1	10	T	SS	0.0121	0.69

The Budget column denotes the constraint input to AUTOGM to generate an algorithm

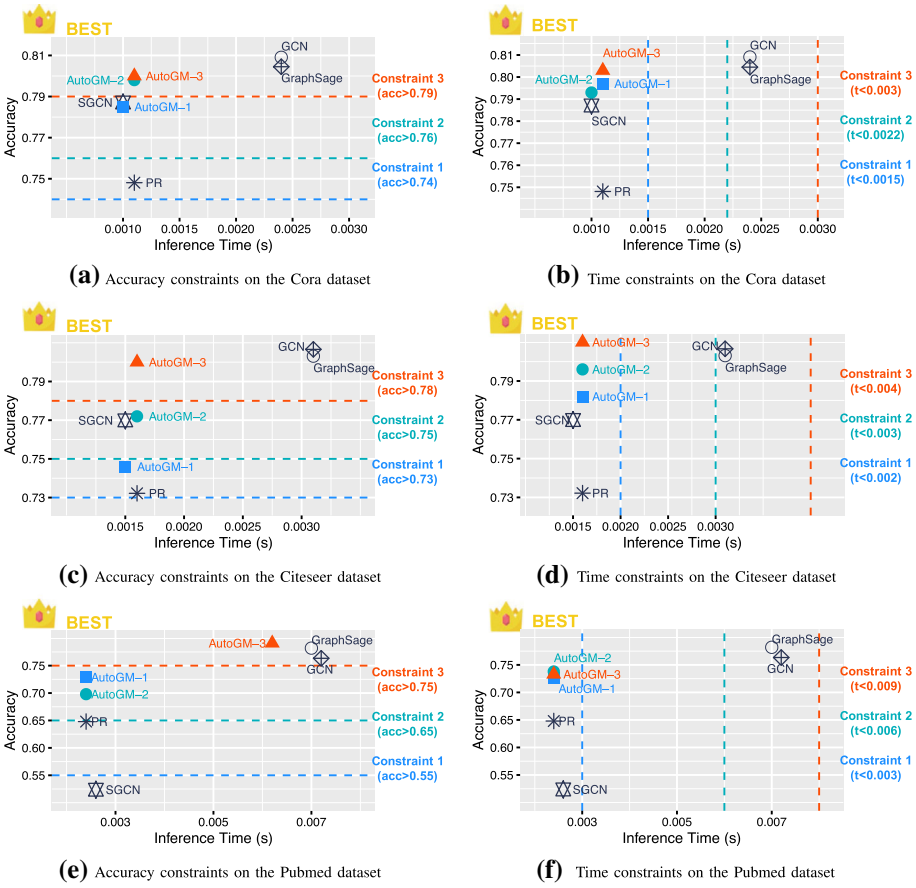
Among algorithms satisfying accuracy lower bounds, the algorithms generated by AUTOGM have the fastest inference time. For instance, in the Pubmed dataset in Fig. 6e, AUTOGM-3 has the fastest inference time above accuracy constraint 3 among GCN and GraphSage. Conversely, among algorithms satisfying inference time upper bounds, the algorithms generated by AUTOGM show the best trade-off between accuracy and inference time. For instance, in the Citeseer dataset in Fig. 6d, AUTOGM-1 has the highest accuracy below time constraint 1 among PR and SGCN. A noteworthy phenomenon in the link prediction task is that algorithms generated from AUTOGM have similar inference times but diverse accuracies. This shows that it's easier to manipulate accuracy by designing graph algorithms, while the dataset largely determines the inference time.

6.3 Search efficiency of AUTOGM

AUTOGM searches for the optimal graph algorithm in a five-dimensional space (d, k, w, l, a) defined by UNIFIEDGM. To show the search efficiency of AUTOGM, we give the same maximum search time and budget constraints to AUTOGM and RandomSearch, then compare the performance of the best graph algorithms each method finds. RandomSearch samples each parameter (d, k, w, l, a) randomly and defines a graph algorithm based on the sampled parameters. We set the maximum search time proportional to the size of the dataset. The budget constraints are chosen based on the best performance among the baseline methods (PageRank, GCN, GraphSage, SGCN). We select the tightest constraints (i.e., fastest inference time and highest accuracy among the baselines) to examine the search efficiency.

Table 7 shows the inference time and accuracy of the optimal graph algorithms AUTOGM and RandomSearch find. RandomSearch fails to find any algorithm satisfying the given accuracy constraints on the Cora, Pubmed, and CoauthorP datasets. It also fails to find any algorithm satisfying the inference time constraints on the Citeseer, AmazonC, and AmazonP datasets. When RandomSearch finds graph algorithms satisfying the given constraints, their performance is still lower than the algorithms found by AUTOGM. For instance, given the inference time upper bound ($t < 0.02$) on the CoauthorC dataset, AUTOGM finds an algorithm with accuracy 0.83 while RandomSearch finds an algorithm with accuracy 0.75.

Table 7 presents how much accuracy/inference time is used under the given budgets to find the optimal graph algorithms (column 6, 7 and 11, 12). AUTOGM generates algorithms whose accuracy (time) is as close as possible to the given accuracy (time) budgets. For instance, AUTOGM finds the fastest graph algorithm with an accuracy of 0.8 when the accuracy lower bound is given as 0.8 on the CoauthorC dataset. By exhausting the budget, AUTOGM improves



(a) Accuracy constraints on the Cora dataset

(b) Time constraints on the Cora dataset

(c) Accuracy constraints on the Citeseer dataset

(d) Time constraints on the Citeseer dataset

(e) Accuracy constraints on the Pubmed dataset

(f) Time constraints on the Pubmed dataset

Fig. 6 AUTOGM finds the algorithms with the best accuracy/inference time trade-off on the link prediction task: given three different accuracy/inference time constraints 1, 2, 3, AUTOGM generates three novel graph algorithms, AUTOGM-1, 2, 3, respectively

the target metric time (accuracy) and brings the best trade-off between computation time and accuracy.

6.4 Effect of UNIFIEDGM parameters

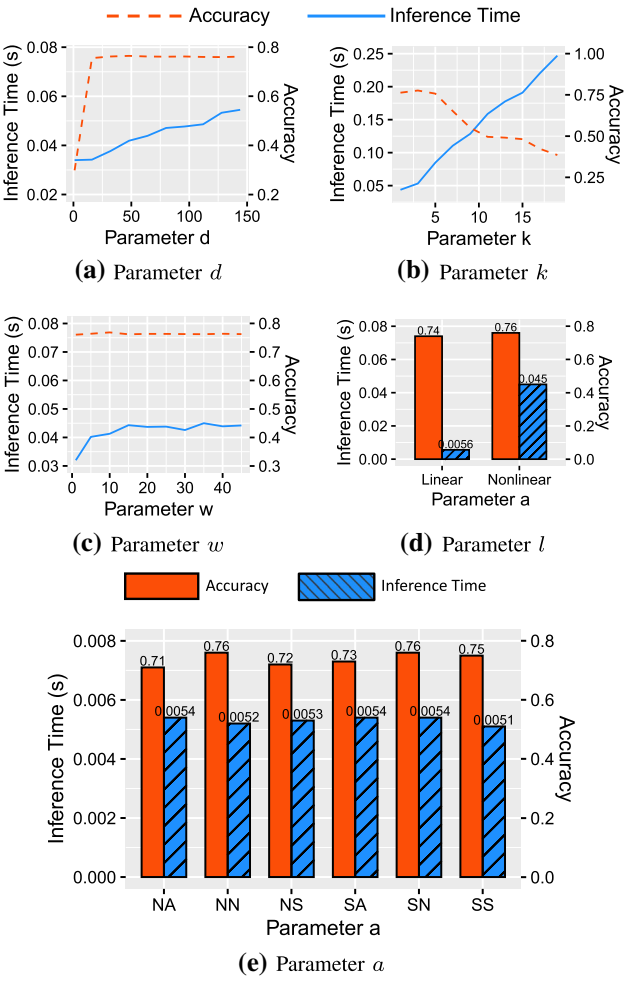
In this section, we investigate the effects of parameters of UNIFIEDGM on the performance of a graph mining algorithm. Given a set of parameters ($d = 64, k = 2, w = -1, l = \text{True}, a = \text{SS}$), we vary one parameter while fixing the others and measure the performance of the generated algorithm. For the experiment where we vary the aggregation parameter a, we use a different set of parameters ($d = 16, k = 2, w = 10, l = False$) to better illustrate changes in accuracy and inference time. For brevity, we show the result on the Pubmed dataset.

- *Dimension d* Figure 7a shows that inference time increases linearly with d, while accuracy increases only until $d > 20$. For the Pubmed dataset, 20-dimensional messages are

Table 7 Search efficiency of AUTOGM: given the same search time (column 2) and accuracy lower bounds (column 3), AUTOGM finds faster algorithms than RandomSearch across all datasets; similarly, given the same search time (column 2) and inference time upper bounds (column 8), AUTOGM finds more accurate algorithms than RandomSearch across all datasets

			Fastest inference (s)		Accuracy			Highest accuracy		Inference (s)	
Dataset	Search(s)	Min. Acc.	AutoGM	Random	AutoGM	Random	Max. Time(s)	AutoGM	Random	AutoGM	Random
Cora	450	0.78	**0.0034**	–	0.79	–	0.004	0.77	**0.77**	0.0036	0.0033
Citeseer	800	0.67	**0.0039**	**0.0039**	0.67	0.67	0.004	**0.67**	–	0.0039	–
Pubmed	1800	0.75	**0.021**	–	0.77	–	0.004	**0.76**	0.71	0.0036	0.0039
AmazonC	5700	0.85	**0.032**	0.033	0.89	0.87	0.04	**0.85**	–	0.032	–
AmazonP	18,000	0.93	**0.047**	0.065	0.94	0.93	0.05	**0.94**	–	0.048	–
CoauthorC	2500	0.8	**0.015**	0.016	0.8	0.82	0.02	**0.83**	0.75	0.015	0.02
CoauthorP	1,500	0.9	**0.01**	–	0.91	–	0.01	**0.92**	0.86	0.01	0.01

Bold values are either the highest accuracy or fastest inference time between AutoGM and Random



(a) Parameter d **(b)** Parameter k

(c) Parameter w **(d)** Parameter l

(e) Parameter a

Fig. 7 Effects of the five parameters (d, k, w, l, a) of UNIFIEDGM on the performance of graph algorithms (i.e., accuracy and time)

expressive enough that the accuracy stops increasing. Larger datasets would likely benefit from higher dimensional messages.

- *Length k* In Fig. 7b, when k increases, inference time increases linearly, but accuracy decreases for $k > 3$. The decrease in accuracy is due to oversmoothing: repeated graph aggregations eventually make node embeddings indistinguishable.
- *Width w* In Fig. 7c, when w increases, inference time increases until $w > 15$, but accuracy does not change noticeably. The plateau in accuracy is due to most nodes having few neighbors and nearby nodes sharing similar feature information, which makes a single sampled node be a representative of a node's whole neighborhood. The plateau in inference time indicates that nodes have fewer than 15 neighbors on average on the Pubmed dataset.
- *Nonlinearity l* Figure 7d shows that adding nonlinearities (l = True) increases accuracy due to richer expressiveness, but also inference time.

- *Aggregation strategy a* Figure 7e shows that the choice of aggregation strategy a has a considerable effect on the accuracy of a graph mining algorithm. Still, we cannot conclude that any aggregation strategy is always superior to others.

Figure 7 shows the general tendency in the effects of the parameters. Different datasets have slightly different results (e.g., which w stops increasing accuracy or which k starts bringing oversmoothing). This shows the need for AUTOGM, which chooses the best parameter set automatically for the dataset we employ.

6.5 Discussion

In this section, we discuss few interesting observations we find during the experiments.

- *Linear model is fast:* As shown in Sect. 6.2, linear models including PageRank and SGCN are faster than nonlinear models. The fast speed of linear models does not merely come from the absence of a nonlinear operation at each layer. Without the nonlinear operation, multi-layers of linear transformation operations could be compressed to one layer as follows:

$$\begin{aligned} X_3 &= \phi(A\phi(AX_0W_0)W_1) \\ &= A(AX_0W_0)W_1 = A^2X_0W_* \end{aligned}$$

  where $\phi(x) = x$ is a linear operation, X_i denotes hidden embeddings at the i-th layer, A is the adjacency matrix, W_i denotes the transformation matrix at the i-th layer, and $W_* = W_0W_1$ is the compressed matrix. Then, in the linear model, we can precompute A^2X_0 in advance and multiply it only with W_* during training. On the other hand, in the nonlinear model, we need to execute matrix multiplication (AX_iW_i) in every layer. This explains the fast speed of linear models compared to nonlinear models.
- *Winning strategy* It is hard to define a single winning strategy for graph mining algorithm development that is generalizable to various graphs and applications. However, we observe a few tendencies. High dimensions of messages generally bring high accuracy with a negligible increase in computation time. For instance, SGCN which has high dimension ($d = 64$) shows higher accuracy than PageRank which has low dimension ($d = 1$) across different tasks (Figs. 5 and 6). Second, nonlinear operations are not necessarily required for high accuracy. For example, SGCN shows comparably high accuracy with GCN and GraphSage in both node classification and link prediction tasks. While maintaining similar accuracy, SGCN is faster than GCN and GraphSage due to the absence of nonlinear operations.
- *Performance is affected by input graphs* In Fig. 6b, d, f, the graph algorithms AUTOGM-1,2,3 that are generated by AUTOGM with different time constraints show similar inference times (sometimes even similar accuracies). The performance of graph algorithms is not only decided by the algorithms but also by input graphs. When graphs are sparse and have simple structures, the graph algorithms will have short inference times regardless of how long inference time constraints we give to AUTOGM. Likewise, when graphs are well-clustered, and features are well-aligned with labels, the tasks become easy, and any graph algorithms would easily get high accuracy. In Fig. 6b and d, the algorithms generated with longer time constraints show higher accuracies while having similar inference times. Longer time constraints allow AUTOGM to explore broader scope in the search space and find better algorithms with higher accuracies, while all algorithms end up showing similar inference times thanks to simple input graph structures.

7 Future work

In AUTOGM, we choose Bayesian Optimization to search the parameter space. Even though BO tries to minimize the number of evaluations, each evaluation corresponds to training a graph mining algorithm, leading to the long computation time of AUTOGM. Various multi-fidelity methods [13, 14] have been proposed to handle this problem. They use cheap approximations to the function of interest (in our case, budget-aware objective functions) to speed up the overall optimization process. Multi-fidelity approaches could significantly improve the computation time of AUTOGM.

Given a graph and a target task, AUTOGM receives the maximum inference time or minimum accuracy constraints as input, then generates an optimal graph algorithm satisfying those constraints. How can we decide proper maximum inference time or minimum accuracy constraints initially? Currently, we do this by trial-and-error—we run AUTOGM with constraints either chosen randomly or from previous experimental results, then calibrate the constraints until AUTOGM finds algorithms with satisfactory inference time and accuracy. Future work should focus on minimizing this constraint-tuning time. We can refer to the history of constraints used on other graphs and modify them based on differences between the referred graphs and a target graph (e.g., if a target graph is bigger than a referred graph, we can increase the referred graph's inference time constraint and use it for the target graph).

8 Conclusion

Graph mining is generally application-driven. The development of a new mining algorithm is usually motivated by solving a specific real-world problem. Given how general graphs are as an abstraction, the resulting algorithm is usually customized to a dataset, application, and domain. Sometimes, to squeeze out the best performance, graph mining uses various heuristics specialized to certain scenarios—0.85 for the decaying coefficient in PageRank for web recommendation [27], 2-layered GCNs for citation networks [10], 3-layer GCNs for open academic graphs [12]. These heuristics make graph mining algorithms less generalizable. Practitioners cannot simply apply existing graph algorithms to their problems but must do trial-and-errors until they find optimal (sometimes suboptimal) algorithms for their scenarios. This widens a gap in which state-of-the-art techniques developed in academic settings fail to be optimally deployed in real-world applications.

This paper shows graph mining has enough room to be further generalized. Various message-passing-based graph algorithms stem from the same intuition, homophily, applied in different ways. Based on this shared intuition, UNIFIEDGM unifies graph algorithms using five parameters of the message-passing mechanism: the dimension of the communicated messages, the number of neighbors to communicate with, the number of steps to communicate for, the nonlinearity of the communication, and the message aggregation strategy. UNIFIEDGM-EXT extends UNIFIEDGM with attention and sampling methodologies and unifies a broader scope of graph algorithms under one framework. This unification helps users understand which aspect of algorithms leads to different accuracy/computation time/memory efficiency and which part of algorithms they should tune to achieve their goals. Furthermore, we automate graph mining algorithm development under this unified framework to prevent users from running trial-and-error and reaching suboptimal algorithms. Our main contributions are:

- *Unification* UNIFIEDGM and UNIFIEDGM- EXT allow conventional graph mining and graph neural network algorithms to be unified under the same framework for the first time, helping practitioners to understand the first principles in message-passing-based algorithms.
- *Design space for graph mining algorithms* UNIFIEDGM provides the parameter search space necessary to automate graph mining algorithm development.
- *Automation* Based on the search space defined by UNIFIEDGM, AUTOGM finds the optimal graph algorithm using Bayesian optimization.
- *Budget awareness* AUTOGM maximizes the performance of an algorithm under a given time/accuracy budget.
- *Effectiveness* AUTOGM finds novel graph algorithms with the best speed/accuracy trade-off on real-world datasets.

We hope this paper will spark further research in this direction and empower practitioners without much expertise in graph mining to deploy graph algorithms tailored to their scenarios. In this era of big data, new graphs and tasks are generated every day. We believe automated graph mining will bring even more impact on a wider range of users across academia and industry in the future.

References

1. Bahmani B, Chowdhury A, Goel A (2010) Fast incremental and personalized pagerank. Proc VLDB Endow 4(3):173–184
2. Bennett J, Lanning S (2007) August. The netflix prize. In Proceedings of KDD cup and workshop (Vol. 2007, p. 35)
3. Brochu E, Cora VM, De Freitas N (2010) A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599
4. Brohee S, Van Helden J (2006) Evaluation of clustering algorithms for protein-protein interaction networks. BMC Bioinf 7(1):1–19
5. Drężewski R, Sepielak J, Filipkowski W (2015) The application of social network analysis algorithms in a system supporting money laundering detection. Inf Sci 295:18–32
6. Eksombatchai C, Jindal P, Liu JZ, Liu Y, Sharma R, Sugnet C, Ulrich M, Leskovec J (2018) April. Pixie: a system for recommending 3+ billion items to 200+ million users in real-time. In: Proceedings of the 2018 world wide web conference. pp. 1775–1784
7. Floreano D, Dürr P, Mattiussi C (2008) Neuroevolution: from architectures to learning. Evol Intel 1(1):47–62
8. Gao Y, Yang H, Zhang P, Zhou C, Hu Y (2020) July. Graph neural architecture search. IJCAI 20:1403–1409
9. Guo M, Yi T, Zhu Y, Bao Y (2021) Jitune: Just-in-time hyperparameter tuning for network embedding algorithms. arXiv preprint arXiv:2101.06427
10. Hamilton W, Ying Z, Leskovec J (2017) Inductive representation learning on large graphs. Advances in neural information processing systems, 30
11. Hooi B, Song HA, Beutel A, Shah N, Shin K, Faloutsos C (2016) August. Fraudar: Bounding graph fraud in the face of camouflage. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. pp. 895–904
12. Hu Z, Dong Y, Wang K, Sun Y (2020) April. Heterogeneous graph transformer. In: Proceedings of The Web Conference 2020. pp. 2704–2710
13. Kandasamy K, Dasarathy G, Oliva JB, Schneider J, Póczos B (2016) Gaussian process bandit optimisation with multi-fidelity evaluations. Advances in neural information processing systems, 29
14. Kandasamy K, Dasarathy G, Schneider J, Póczos B (2017) July. Multi-fidelity bayesian optimisation with continuous approximations. In: International Conference on Machine Learning. PMLR. pp. 1799–1808
15. Kandasamy K, Neiswanger W, Schneider J, Poczos B, Xing EP (2018) Neural architecture search with bayesian optimisation and optimal transport. Advances in neural information processing systems, 31

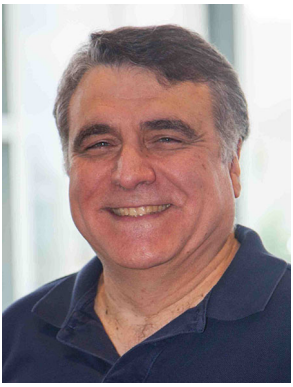
16. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. Proceedings of the 3rd International Conference on Learning Representations 2014
17. Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations 2017
18. Kitano H (1990) Designing neural networks using genetic algorithms with graph generation system. Complex Syst 4:461–476
19. Kleinberg JM (1999) Authoritative sources in a hyperlinked environment. J ACM (JACM) 46(5):604–632
20. Koutra D, Ke TY, Kang U, Chau DHP, Pao HKK, Faloutsos C (2011) September. Unifying guilt-by-association approaches: theorems and fast algorithms. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 245–260). Springer, Berlin, Heidelberg
21. Lemaire C, Achkar A, Jodoin PM (2019) Structured pruning of neural networks with budget-aware regularization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9108–9116
22. Li X, Zhou Y, Pan Z, Feng J (2019) Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9145–9153
23. Liu H, Simonyan K, Yang Y (2019) Darts: differentiable architecture search. International Conference on Learning Representations, ICLR 2019
24. Liu H, Simonyan K, Vinyals O, Fernando C, Kavukcuoglu K (2017) Hierarchical representations for efficient architecture search. International Conference on Learning Representations, ICLR 2018
25. Metropolis N, Ulam S (1949) The Monte Carlo method. J Am Stat Assoc 44(247):335–341
26. Michalak K, Korczak J (2011) September. Graph mining approach to suspicious transaction detection. In 2011 Federated conference on computer science and information systems (FedCSIS) (pp. 69–75). IEEE
27. Page L, Brin S, Motwani R, Winograd T (1999) The PageRank citation ranking: bringing order to the web. Stanford InfoLab
28. Pearl J (2022) Reverend Bayes on inference engines: a distributed hierarchical approach. In Probabilistic and Causal Inference: The Works of Judea Pearl. pp. 129–138
29. Robert JV (2021) Linear programming: foundations and extensions. Springer, Berlin
30. Sen P, Namata G, Bilgic M, Getoor L, Galligher B, Eliassi-Rad T (2008) Collective classification in network data. AI Mag 29(3):93–93
31. Shchur O, Mumme M, Bojchevski A, Günnemann S (2018) 2018. Pitfalls of graph neural network evaluation, Relational Representation Learning Workshop, NeurIPS
32. Shin K, Eliassi-Rad T, Faloutsos C (2016) December. Corescope: Graph mining using k-core analysis-patterns, anomalies and algorithms. In: 2016 IEEE 16th international conference on data mining (ICDM) (pp. 469–478). IEEE
33. Strehl A, Ghosh J (2000) December. A scalable approach to balanced, high-dimensional clustering of market-baskets. In International Conference on High-Performance Computing (pp. 525–536). Springer, Berlin, Heidelberg
34. Szekeres G, Wilf HS (1968) An inequality for the chromatic number of a graph. J Comb Theory 4(1):1–3
35. Tu K, Ma J, Cui P, Pei J, Zhu W (2019) July. Autone: Hyperparameter optimization for massive network embedding. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 216–225
36. Vazquez A, Flammini A, Maritan A, Vespignani A (2003) Global protein function prediction from protein-protein interaction networks. Nat Biotechnol 21(6):697–700
37. Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2018) 6th International Conference on Learning Representations , ICLR 2018
38. Wang Y, Liu S, Yoon M, Lamba H, Wang W, Faloutsos C, Hooi B (2020) November. Provably robust node classification via low-pass message passing. In: 2020 IEEE International Conference on Data Mining (ICDM) (pp. 621–630). IEEE
39. Wright S, Nocedal J (1999) Numerical optimization. Springer Sci 35(67–68):7
40. Wu F, Souza A, Zhang T, Fifty C, Yu T, Weinberger K (2019) May. Simplifying graph convolutional networks. In International conference on machine learning (pp. 6861–6871). PMLR
41. Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY (2020) A comprehensive survey on graph neural networks. IEEE Trans ctions Neural Networks Learn Systems 32(1):4–24
42. Yao L, Mao C, Luo Y (2019) July. Graph convolutional networks for text classification. In Proceedings of the AAAI conference on artificial intelligence (Vol. 33, No. 01, pp. 7370-7377)
43. Yoon M, Jung J, Kang U (2018) April. Tpa: fast, scalable, and accurate method for approximate random walk with restart on billion scale graphs. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE) (pp. 1132–1143). IEEE

44. Yoon M, Jin W, Kang U (2018) April. Fast and accurate random walk with restart on dynamic graphs with guarantees. In: Proceedings of the 2018 World Wide Web Conference (pp. 409–418)
45. Yoon M, Hooi B, Shin K, Faloutsos C (2019) July. Fast and accurate anomaly detection in dynamic graphs with a two-pronged approach. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 647–657)
46. Yoon M, Gervet T, Hooi B, Faloutsos C (2020) November. Autonomous graph mining algorithm search with best speed/accuracy trade-off. In: 2020 IEEE International Conference on Data Mining (ICDM) (pp. 751–760). IEEE
47. You J, Ying Z, Leskovec J (2020) Design space for graph neural networks. Adv Neural Inf Process Syst 33:17009–17021
48. Yuan Y, Wang W, Coghill GM, Pang W (2021) A novel genetic algorithm with hierarchical evaluation strategy for hyperparameter optimisation of graph neural networks. arXiv preprint arXiv:2101.09300
49. Zamir O, Etzioni O (1998) August. Web document clustering: a feasibility demonstration. In Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval (pp. 46–54)
50. Zhang Z, Wang X, Zhu W (2021) Automated machine learning on graphs: a survey. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21), Survey Track
51. Zhong Z, Yan J, Liu CL (2017) Practical network blocks design with q-learning. arXiv preprint arXiv:1708.05552, 6
52. Zoph B, Le QV (2017) Neural architecture search with reinforcement learning. International Conference on Learning Representations, ICLR 2017

Minji Yoon is a Ph.D. student in the Computer Science Department of Carnegie Mellon University. She received B.S. and M.S. in Computer Science and Engineering at Seoul National University. Her research interests include graph mining and deep learning with emphasis on graph neural networks.

Théophile Gervet is a Ph.D. student in the Machine Learning Department of Carnegie Mellon University. He received a B.S. in Computer Science from McGill University. His research focuses on practical applications of deep learning.

Bryan Hooi is an assistant professor in the Computer Science Department, School of Computing, and the Institute of Data Science in National University of Singapore. He received his PhD degree in Machine Learning from Carnegie Mellon University, USA in 2019. His research interests include machine learning on graph-structured data, robustness and novelty detection, and spatiotemporal data mining. His work aims to develop efficient and practical approaches, for applications including fraud detection, online commerce, and automatic monitoring of medical, industrial, weather and environmental sensor data.



Christos Faloutsos is a Professor at Carnegie Mellon University. He is the recipient of the Fredkin Professorship in Artificial Intelligence (2020); he has received the Presidential Young Investigator Award by the National Science Foundation (1989), the Research Contributions Award in ICDM 2006, the SIGKDD Innovations Award (2010), the PAKDD Distinguished Contributions Award (2018), 30 "best paper" awards (including 8 "test of time" awards), and four teaching awards. He is an ACM Fellow, he has published over 500 refereed articles, 17 book chapters and three monographs. His research interests include large-scale data mining with emphasis on graphs and time sequences; anomaly detection, tensors, and fractals.