Autonomous Graph Mining Algorithm Search with Best Speed/Accuracy Trade-off

Minji Yoon
Carnegie Mellon University
minjiy@cs.cmu.edu

Théophile Gervet
Carnegie Mellon University
tgervet@andrew.cmu.edu

Bryan Hooi
National University of Singapore
bhooi@comp.nus.edu.sg

Christos Faloutsos
Carnegie Mellon University
christos@cs.cmu.edu

Abstract—Graph data is ubiquitous in academia and industry, from social networks to bioinformatics. The pervasiveness of graphs today has raised the demand for algorithms that can answer various questions: Which products would a user like to purchase given her order list? Which users are buying fake followers to increase their public reputation? Myriads of new graph mining 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 a practitioner to compare different algorithms and pick the most suitable one for a specific application. These challenges — even more severe for non-experts — create a gap in which state-of-the-art techniques developed in academic settings 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 that integrates various message-passing based graph algorithms, ranging from conventional algorithms like PageRank to graph neural networks. Then 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 incorporate a practical issue — finding the best speed-accuracy trade-off under a computation budget — into the graph algorithm generation problem. Experiments on real-world benchmark datasets demonstrate that AUTOGM generates novel graph mining algorithms with the best speed/accuracy trade-off compared to existing models with heuristic parameters.

Index Terms—automation, unified framework, optimization

I. INTRODUCTION

Many real-world problems are naturally modeled using graphs: who-buys-which-products in online marketplaces [28], who-follows-whom in social networks [19], [30], and protein relationships in biological networks [4], [24]. Graph mining provides solutions to practical problems such as classification of web documents [27], [31], clustering in market segmentation [22], recommendation in streaming services [2], and fraud detection in banking [5], [18].

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 [26] 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 [9], [17] — 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 [13], [19] — 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.

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 [14], [15]. AUTOGM defines a novel budget-aware objective function capturing the speed/accuracy trade-off, then maximizes the objective function to find the optimal param-
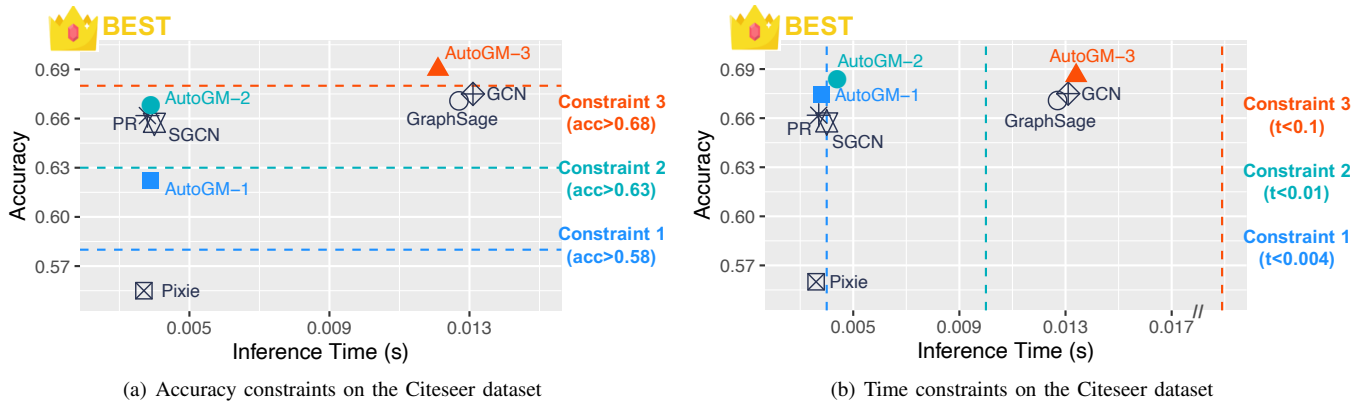
Fig. 1: **AUTOGM finds novel graph algorithms with the best accuracy/inference time trade-off**: (a) Given three accuracy lower bounds (i.e., $0.58, 0.63, 0.68$), AUTOGM generates three novel graph algorithms minimizing inference time. (b) Given three inference time upper bounds (i.e., $0.004, 0.01, 0.1$ seconds), AUTOGM generates three novel graph algorithms maximizing accuracy.

eter 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 are as follows:

- **Unification:** UNIFIEDGM is a unified framework for message-passing based graph 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 (Figure 1).

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

II. BACKGROUND & RELATED WORK

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 hyper-parameter 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], [12], [16] 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 [32], [33], 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 [9] 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 [9]. 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.

A. 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].

However, 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. However, before UNIFIEDGM,

TABLE I: 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 $(i = 1 \dots k)$
W_1	$(d_0 \times d)$ 1st layer transformation matrix
W_i	$(d \times d)$ i-th layer transformation matrix $(i = 2 \dots k)$
$\phi(x)$	$\begin{cases} ReLU(x) \text{ if } l = \text{True} \\ x \text{ otherwise} \end{cases}$

the hyperparameter search space for graph mining was not even defined due to the lack of unity among algorithms. Hence, our proposed UNIFIEDGM and AUTOGM allow the graph mining field to exploit state-of-the-art techniques developed in AutoML.

III. UNIFIED GRAPH MINING FRAMEWORK

In this section, we first motivate the message passing scheme (Section III-A). We then propose our unified framework UNIFIEDGM (Section III-B), explain how existing algorithms fit in the framework (Section III-C), and further analyze how UNIFIEDGM bridges the conceptual gap between conventional graph mining and graph neural networks (Section III-D). Finally, we outline how to choose parameters of UNIFIEDGM given a specific scenario (Section III-E).

A. 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.

B. 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.
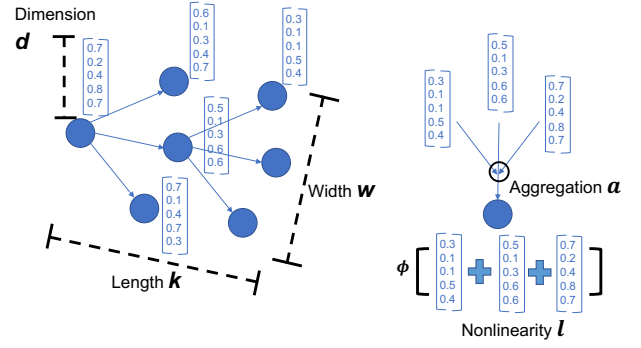

Fig. 2: UNIFIEDGM defines the message passing mechanism based on five parameters: the dimension d, length k, width w, nonlinearity l, and aggregation strategy a.

- **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 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 $A_{\text{samp}} = \text{Sample}(A)$ by randomly zeroing out entries of the binary adjacency matrix A. Message aggregation (lines 4 and 5) is defined by the aggregation strategy $a \in \{\text{SA}, \text{SS}, \text{SN}, \text{NA}, \text{NS}, \text{NN}\}$. The first letter in $\{\text{S}, \text{N}\}$ determines whether a node sends a message to itself or not (Self-loop or No-self-loop). The second letter in $\{\text{A}, \text{S}, \text{N}\}$ determines how to normalize the sum of incoming messages (Asymmetric, Symmetric, or No-normalization). Each aggregation strategy a results in an

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

---

aggregation matrix $A_{\text{agg}} = Aggregate(A_{\text{samp}})$, explained in Table II. 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:

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

X_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 $(i = 1 \dots k)$. W_1 and W_i are $(d_0 \times d)$ and $(d \times d)$ transformation matrices respectively $(i = 2 \dots k)$.

C. 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 III shows how to set initial node statistics and parameters (d, k, w, l, a) of UNIFIEDGM to reproduce the original graph algorithms.

PageRank [19] 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 $(l = \text{False})$ until scores have converged $(k = \infty)$, and aggregates messages with no self-loop and asymmetric normalization $(a = \text{NA})$. 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) [29], [30] 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.

TABLE II: 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). 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.

	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

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 $d = 1$ and $l = \text{False}$ with PPR/RWR. For the parameter a, Pixie sums up scores from the sampled neighbors (NN: no self-loop and no normalization).

Graph Convolutional Networks (GCNs) [11] 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 $(l = \text{True})$ across two-layered networks $(k = 2)$ and aggregates messages with a self-loop and symmetric normalization $(a = \text{SS})$.

GraphSAGE [8] 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) [25] 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 III presents the original message passing equations of the existing graph algorithms. Those equations can be fully reproduced from Algorithm 1 with the proper inital node statistics and parameter sets listed in Table III.

D. 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

TABLE III: Graph mining algorithms can be fully reproduced under UNIFIEDGM with the respective initial node statistics and parameters (d, k, w, l, a). Notation: 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, 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 [30].

Algorithm	Original message passing equation	Initial node statistics	d	k	w	l	a
PageRank [19]	$X_k = c(D^{-1}A)X_{k-1}$	$\frac{1}{n}$ for all nodes	1	∞	-1	False	NA
Pixie [6]	$X_k(u) = \sum_{v \in N(u)} X_{k-1}(v)$	1 for seeds, 0 others	1	sample	$\frac{2000}{k}$	False	NN
GCN [11]	$X_k = ReLU\left((D^{-\frac{1}{2}}(A + I_n)D^{-\frac{1}{2}})X_{k-1}W_k\right)$	feature vectors	64	2	-1	True	SS
GraphSAGE [8]	$X_k(u) = ReLU\left(\frac{1}{|N(u)|+1}\sum_{v \in N(u) \cup u} X_{k-1}(v)W_k\right)$	feature vectors	64	2	25	True	SA
SGCN [25]	$X_k = D^{-\frac{1}{2}}(A + I_n)D^{-\frac{1}{2}}X_{k-1}W_k$	feature vectors	64	2	-1	False	SS

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 tranformation 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.

E. 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:** k decides the size of neighborhood 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.

IV. 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 (Section IV-A). We explain why Bayesian optimization is well-suited for this unconstrained problem (Section IV-B). Finally, we describe how AUTOGM solves the optimization problem using Bayesian optimization (Section IV-C).

A. 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:

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

One of the common ways to solve a constrained optimization problem is using a barrier method [23], replacing inequality constraints by a penalizing term in the objective function. We re-formulate the original constrained problem in Equation 1 as an equivalent unconstrained problem as follows:

$$x_{opt} = \operatorname{argmin}_x g(x) + I_{h(x)-h_{\min}\geq0}(x) \quad (2)$$

where the indicator function $I_{h(x)-h_{\min}\geq0}(x) = 0$ 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:

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

$$x_{opt} = \operatorname{argmin}_x f_{GM}(x) \quad (4)$$

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, Equation 1. However, as λ approaches zero, it becomes an ever-better approximation (i.e., $-\lambda \log(h(x) - h_{\min})$ approaches $I_{h(x)-h_{\min}\geq0}(x)$) [23]. The solution of Equation 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 $h(x) - h_{\min} = acc - acc_{\min} \geq 0$

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}

---

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.

B. 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 Section II-A). 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 $f_{GM}(x)$.

C. 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.

V. 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? (Section V-B)
- **Q2. Search efficiency of AUTOGM:** How long does AUTOGM take to find the optimal graph algorithm? How efficient it is compared to random search? (Section V-C)
- **Q3. Effect of UNIFIEDGM parameters:** How do parameters (d, k, w, l, a) affect the accuracy and inference time of a graph mining algorithm? (Section V-D)

A. Experimental Setting

We evaluate the performance of graph mining algorithms on a semi-supervised node classification task. All experiments were conducted on identical machines using the Amazon EC2 service (p2.xlarge with 4 vCPUs, 1 GPU and 61 GB RAM). **Dataset:** We use the three citation networks (Cora, Citeseer, and Pubmed) [20], two Amazon co-purchase graphs (Amazon Computers and Amazon Photo) [21], and two co-authorship graphs (MS CoauthorCS and MS CoauthorPhysics) [21]. We report their statistics in Table IV. **Baseline:** Our baselines are PageRank [19], Pixie [6], GCN [11], GraphSage [8], and SGCN [25]. We generate each algorithm under UNIFIEDGM by setting the five parameters as follows:

TABLE IV: Dataset statistics: AmazonC and AmazonP denote the Amazon Computer and Amazon Photo datasets, respectively. CoauthorC and CoauthorP denote the MS Coauthor CS and Physics, repectively.

Dataset	Node	Edge	Feature	Label	Train/Val/Test
Cora	2,485	5,069	1,433	7	140/500/1,000
Citeseer	2,110	3,668	3,703	6	120/500/1,000
Pubmed	19,717	44,324	500	3	60/500/1,000
AmazonC	13,381	245,778	767	10	410/1,380/12,000
AmazonP	7,487	119,043	745	8	230/760/6,650
CoauthorC	18,333	81,894	6,805	15	550/1,830/15,950
CoauthorP	34,493	247,962	8,415	5	1,030/3,450/30,010

- PageRank: $d = 1, k = 30, w = -1, l = $ False$, a = $ NA
- Pixie: $d = 1, k = 10, w = 25, l = $ False$, a = $ NN
- GCN: $d = 64, k = 2, w = -1, l = $ True$, a = $ SS
- GraphSAGE: $d = 64, k = 2, w = 25, l = $ True$, a = $ SA
- SGCN: $d = 64, k = 2, w = -1, l = $ False$, a = $ SS

When w is larger than the number of neighbors, we sample neighbors with replacement. For PageRank and Pixie, the original algorithms output 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 and Pixie with other algorithms in terms of their main features in UNIFIEDGM: their low dimension ($d = 1$) and Pixie's sampling strategy ($w = 25$).

Bayesian optimization: We use an open-sourced Bayesian optimization package[2]. 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 VI. 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 [10] 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.

B. Effectiveness of AUTOGM

In this section, we demonstrate how AUTOGM trades off accuracy and inference time in practice. 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 Figures 1 and 3. 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

---

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

(a) Accuracy constraints on the Cora dataset

(b) Time constraints on the Cora dataset

(c) Accuracy constraints on the Pubmed dataset

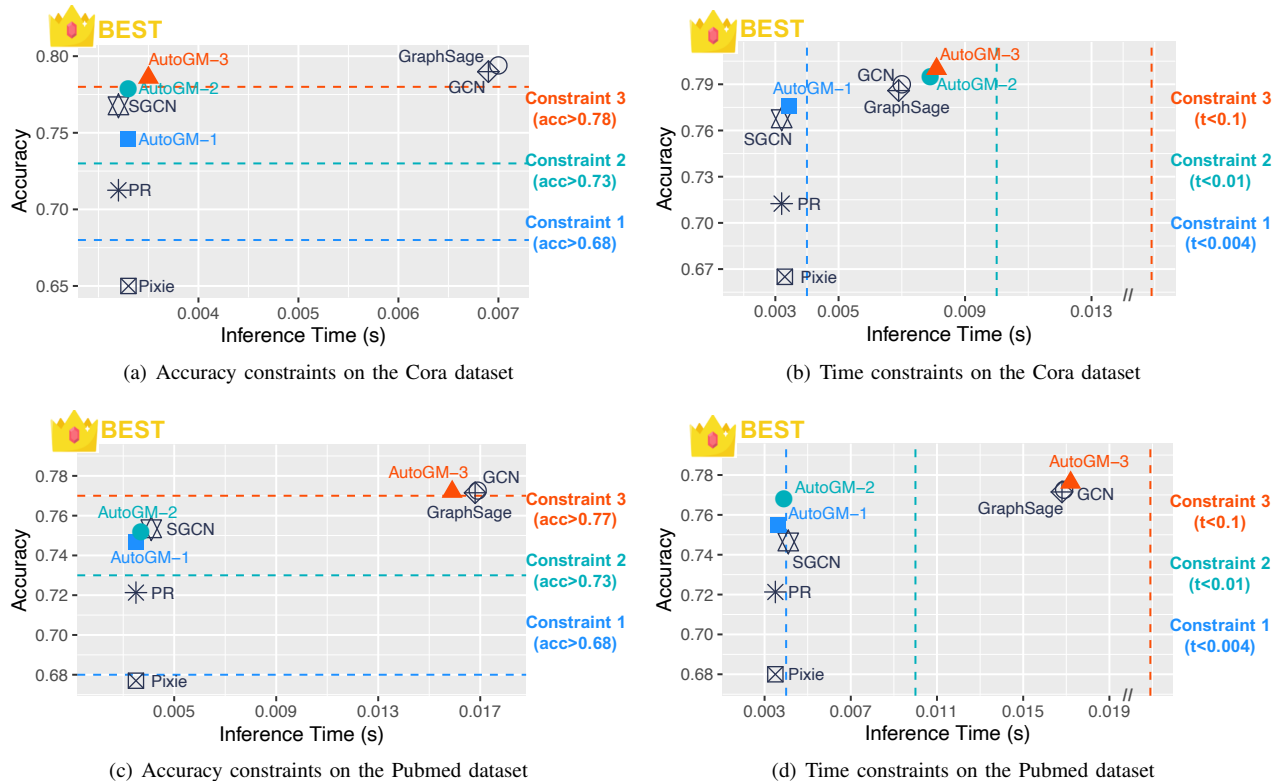(d) Time constraints on the Pubmed dataset

Fig. 3: **AUTOGM finds the algorithms with the best accuracy/inference time trade-off on the Cora and Pubmed datasets**: given three different accuracy/inference time constraints 1, 2, 3, AUTOGM generates three novel graph algorithms, AUTOGM-1, 2, 3, respectively.

TABLE V: Parameters corresponding to algorithms found by AU-TOGM in Figures 1. The Budget column denotes the constraint input to AUTOGM to generate an algorithm.

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

Cora, Citeseer, and Pubmed datasets. Performance on other datasets is reported in Table VI.

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 Figure 1(a), 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 Figure 3(d), AUTOGM-1 has the highest accuracy below time constraint 1 among PR, Pixie, 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 Section III-E. PageRank and Pixie achieve fast inference time with a low dimension of messages ($d = 1$)

and no nonlinearities ($l = $ False), but sacrifice accuracy. Pixie underperforms because it is the only algorithm where messages are aggregated without normalization. This aggregation strategy is not well-suited to our semi-supervised node classification task because it leads to node embeddings monotonically increasing with every aggregation step. 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 V 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 Section III-E. Vastly different parameter sets for each algorithm in Table V show that AUTOGM searches the parameter space beyond human intuition, which underlines the value of autonomous graph mining algorithm development.

C. 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

TABLE VI: 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.

Dataset	Search(s)	Min.Acc.	Fastest Inference (s)		Accuracy		Max.Time(s)	Highest Accuracy		Inference (s)	
			AutoGM	Random	AutoGM	Random		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	1,800	0.75	**0.021**	-	0.77	-	0.004	**0.76**	0.71	0.0036	0.0039
AmazonC	5,700	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	2,500	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

same maximum search time and budget constraints to AU-TOGM 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, Pixie, 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 VI 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 VI 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 the target metric time (accuracy) and brings the best trade-off between computation time and accuracy.

D. 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 = $ True, $a = $ 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 4(a) shows that inference time increases linearly with d, while accuracy increases only until $d > 20$. For the Pubmed dataset, 20-dimensional messages
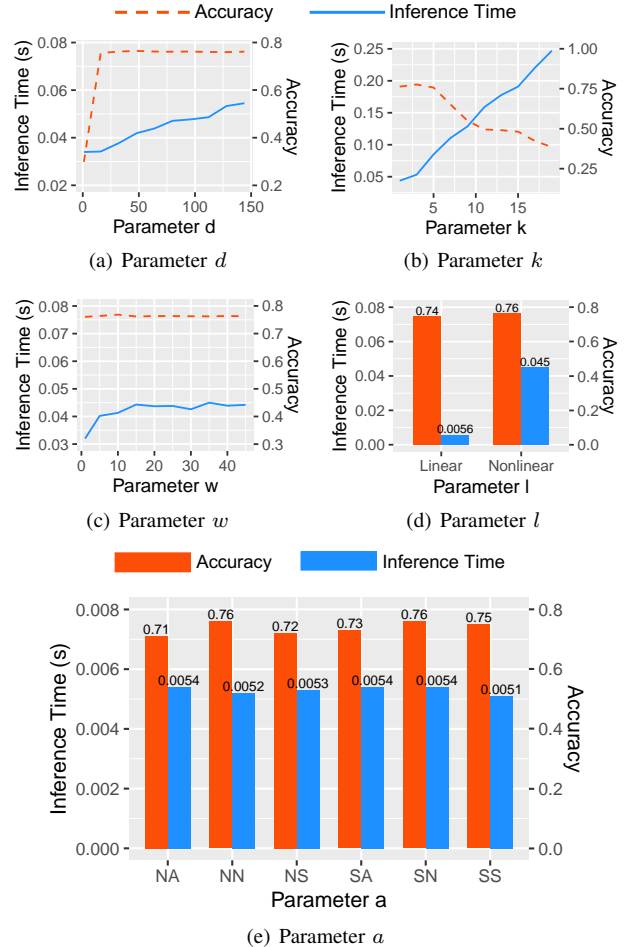

Fig. 4: Effects of the five parameters (d, k, w, l, a) of UNIFIEDGM on the performance of graph algorithms (i.e. accuracy and time).

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

- **Length** k: In Figure 4(b), 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 Figure 4(c), 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 4(d) shows that adding nonlinearities (l = True) increases accuracy due to richer expressiveness, but also inference time.
- **Aggregation strategy** a: Figure 4(e) 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 4 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.

VI. CONCLUSION & FUTURE WORK

In this paper, we introduce an automated system AUTOGM for graph mining algorithm development. Our main contributions are:

- **Unification:** UNIFIEDGM allows conventional and GNN algorithms to be unified in the same framework for the first time, which is necessary to establish the parameter space for algorithm search.
- **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.

REFERENCES

[1] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. Proceedings of the VLDB Endowment, 4(3), 2010.

[2] J. Bennett, S. Lanning, et al. The netflix prize. In Proceedings of KDD cup and workshop, volume 2007. Citeseer, 2007.

[3] E. Brochu, V. M. Cora, and N. De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599, 2010.

[4] S. Brohee and J. Van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. BMC bioinformatics, 7(1), 2006.

[5] R. Drezewski, J. Sepielak, and W. Filipkowski. The application of social network analysis algorithms in a system supporting money laundering detection. Information Sciences, 295:18–32, 2015.

[6] C. Eksombatchai, P. Jindal, J. Z. Liu, Y. Liu, R. Sharma, C. Sugnet, M. Ulrich, and J. Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In Proceedings of the 2018 world wide web conference, 2018.

[7] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. Evolutionary intelligence, 1(1), 2008.

[8] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In Advances in neural information processing systems, 2017.

[9] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing. Neural architecture search with bayesian optimisation and optimal transport. In Advances in Neural Information Processing Systems, pages 2016–2025, 2018.

[10] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[11] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907, 2016.

[12] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. Complex systems, 4(4), 1990.

[13] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. Journal of the ACM (JACM), 46(5), 1999.

[14] C. Lemaire, A. Achkar, and P.-M. Jodoin. Structured pruning of neural networks with budget-aware regularization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 9108–9116, 2019.

[15] X. Li, Y. Zhou, Z. Pan, and J. Feng. Partial order pruning: for best speed/accuracy trade-off in neural architecture search. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 9145–9153, 2019.

[16] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436, 2017.

[17] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055, 2018.

[18] K. Michalak and J. Korczak. Graph mining approach to suspicious transaction detection. In 2011 Federated conference on computer science and information systems (FedCSIS). IEEE, 2011.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[20] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. AI magazine, 29(3), 2008.

[21] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann. Pitfalls of graph neural network evaluation. arXiv preprint arXiv:1811.05868, 2018.

[22] A. Strehl and J. Ghosh. A scalable approach to balanced, high-dimensional clustering of market-baskets. In International Conference on High-Performance Computing. Springer, 2000.

[23] R. J. Vanderbei. Linear programming: Foundations and extensions. 2001. Google Scholar Google Scholar Cross Ref Cross Ref, 2001.

[24] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Global protein function prediction from protein-protein interaction networks. Nature biotechnology, 21(6), 2003.

[25] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger. Simplifying graph convolutional networks. arXiv preprint arXiv:1902.07153, 2019.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. arXiv preprint arXiv:1901.00596, 2019.

[27] L. Yao, C. Mao, and Y. Luo. Graph convolutional networks for text classification. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, 2019.

[28] M. Yoon, B. Hooi, K. Shin, and C. Faloutsos. 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 & Data Mining, 2019.

[29] M. Yoon, W. Jin, and U. Kang. Fast and accurate random walk with restart on dynamic graphs with guarantees. In Proceedings of the 2018 World Wide Web Conference, 2018.

[30] M. Yoon, J. Jung, and U. Kang. 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). IEEE, 2018.

[31] O. Zamir and O. Etzioni. Web document clustering: A feasibility demonstration. In Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, 1998.

[32] Z. Zhong, J. Yan, and C.-L. Liu. Practical network blocks design with q-learning. arXiv preprint arXiv:1708.05552, 6, 2017.

[33] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578, 2016.