Enhancing Machine Learning Approaches for Graph Optimization Problems with Diversifying Graph Augmentation

Chen-Hsu Yang
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
s107062525@m107.nthu.edu.tw

Chih-Ya Shen
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
chihya@cs.nthu.edu.tw

Abstract

Recently, many machine learning-based approaches that effectively solve graph optimization problems have been proposed. These approaches are usually trained on graphs randomly generated with graph generators or sampled from existing datasets. However, we observe that such training graphs lead to poor testing performance if the testing graphs are not generated analogously, i.e., the generalibility of the models trained on those *randomly generated* training graphs are very limited. To address this critical issue, in this paper, we propose a new framework, named *Learning with Iterative Graph Diversification (LIGD)*, and formulate a new research problem, named *Diverse Graph Modification Problem (DGMP)*, that iteratively generate diversified training graphs and train the models that solve graph optimization problems to significantly improve their performance. We propose three approaches to solve DGMP by considering both the performance of the machine-learning approaches and the structural properties of the training graphs. Experimental results on well-known problems show that our proposed approaches significantly boost the performance of both supervised and reinforcement learning approaches and produce near-optimal results, significantly outperforming the baseline approaches, such as graph augmentation and deep learning-based graph generation approaches.

CCS Concepts

• **Computing methodologies** → **Machine learning**; • **Mathematics of computing** → **Graph theory**.

Keywords

Graph optimization, graph augmentation, machine learning

ACM Reference Format:
Chen-Hsu Yang and Chih-Ya Shen. 2022. Enhancing Machine Learning Approaches for Graph Optimization Problems with Diversifying Graph Augmentation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '22), August 14–18, 2022, Washington, DC, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3534678.3539437

KDD '22, August 14–18, 2022, Washington, DC, USA

ACM ISBN 978-1-4503-9385-0/22/08…$15.00
https://doi.org/10.1145/3534678.3539437

1 Introduction

Solving graph optimization problems is an important research topic that has been studied actively. Well-known graph optimization problems, such as Minimum Vertex Cover (MVC), Boolean Satisfiability (SAT), Traveling Salesman Problem (TSP), and Maximum Independent Set (MIS), have various real-world applications. Due to the intractability of most graph optimization problems, exact algorithms that find the optimal solutions usually incur an unacceptable amount of time. To reduce the computational burden, one promising direction is to devise approximation algorithms that find solutions with provable performance guarantees. However, some problems are proven to have poor or even no approximation algorithms (e.g., TSP is inapproximable within any factor, and MIS is inapproximable within $O(|V|)$). Therefore, researchers usually design heuristic approaches that find good solutions efficiently. However, designing such heuristic algorithms often requires domain knowledge and huge human effort of the specific problem.

A recent line of research turns their attention to *learning* heuristic strategies for graph optimization problems with machine learning (ML) approaches. They achieve very good performance on a variety of graph optimization problems. For example, supervised learning approaches are proposed to solve TSP [22, 27], SAT and MIS [16], while reinforcement learning (RL) is also employed to solve TSP [20], MVC [12], and Vehicle Routing Problems (VRP) [14, 18, 19]. These ML approaches focus on finding good solutions for specific graph optimization problems. However, their training and testing data are usually randomly generated [12, 14] or randomly sampled from existing datasets [16]. This raises an important question: *Does the dissimilarity of training and testing graphs impact the performance when solving graph optimization problems?*

Table 1: Testing performance for *unified* and *divergent* settings on MVC (smaller approx. ratio indicates better result)

	ER-Uni	ER-Div	BA-Uni	BA-Div
Approx. ratio	**1.01**	1.26	**1.04**	1.40

Motivating example. To answer this question, Table 1 presents a motivating example. We observe that, even there is only a slight difference on the properties of testing and training graphs, the performance drops dramatically. More specifically, Table 1 compares the results of employing one of the state-of-the-art ML approaches, S2V-DQN [12], to solve the Minimum Vertex Cover (MVC) problem under two training/testing data generation settings: i) *Unified setting with ER and BA* – the training and testing graphs are generated with the same setting, which is widely adopted by current ML models that solve graph optimization problems. Here, we generate two

sets of training and testing graphs: i-a) Training and testing graphs are generated with Erdos-Renyi (ER) model [5] with edge probability fixed to 0.15 (named `ER-Uni`), and i-b) training and testing graphs are both generated with Barabasi-Albert (BA) model [1] with $m = 4$ (`BA-Uni`). ii) *Divergent setting with ER and BA* – the testing graphs are generated differently against the training graphs: ii-a) The training and testing graphs are generated by ER model with edge probabilities 0.15 and 0.3, respectively (`ER-Div`), and ii-b) the training and testing graphs are generated by BA model with $m = 4$ and $m = 8$, respectively (`BA-Div`).

Table 1 presents the solution quality with the *approximation ratio*, which is the ratio of the objective value acquired by the ML approach divided by that of the optimal solution. The closer the approximation ratio is to 1.0, the better performance the ML approach achieves. Please note that due to the nature of optimization problems, a *tiny difference* on the approximation ratio usually indicates a *significant performance change*. The result manifests that under *Unified setting*, i.e., `ER-Uni` and `BA-Uni`, their approximation ratios are very close to 1, indicating the ML approach is able to obtain near-optimal solutions when the training and testing graphs are generated analogously. This setting is widely adopted by current ML approaches for graph optimization problems. In contrast, for *Divergent setting*, `ER-Div` and `BA-Div` both achieve very poor solution quality, because the ML approach is trained and tested on dissimilar graphs. We argue that this *Divergent* setting *fits the practical scenario better and needs to be carefully addressed* (more results are presented in Sec. 2).

In practical scenarios, the testing graphs may be very dissimilar to the training ones, leading to significant performance degradation. To tackle this problem, we observe that the *diversity* of the training graphs plays a crucial role (detailed in Sec. 2). That is, the models trained on a diverse set of training graphs perform much better than their counterparts trained on a set of similar graphs. Therefore, we explore how to construct diversified training graphs and improve the model performance accordingly. To our best knowledge, this is the first paper that proposes to improve the *ML model performance for graph optimization problems* by constructing diversified training graphs. We envisage that our insights and proposed approaches can benefit a wide variety of such ML models.

To achieve our goal, there are two major challenges: i) **The impact of training graph diversity on model performance.** Table 1 briefly illustrates the potential weaknesses when the training and testing graphs are dissimilar. However, does enhancing the diversity of training graphs significantly improves the model performance? To answer this, we provide an analysis to this question in Sec. 2. ii) **Constructing and leveraging diversified training graphs.** Since the diversity of training graphs impacts the performance of the ML models, how to effectively construct the diversified training graphs and well leverage them is the second major challenge. To tackle this challenge, we propose a framework, named *Learning with Iterative Graph Diversification (LIGD)*, and formulate a new research problem, named *Diverse Graph Modification Problem (DGMP)*. Given a set of k training graphs, a modification budget b, *DGMP* performs at most b edge modifications on each training graph to maximize their overall diversity. We propose algorithm *Performance-Aware Structure Diversifying Modification (PASDM)* to DGMP. Our framework *LIGD* monitors the performance of the ML approach and iteratively generates diversified training graphs to boost the performance.

Please note that solving graph optimization problems is different from and much more challenging than general tasks for general graph augmentation, e.g., [23, 29, 34, 35], because i) graph augmentation aims at augmenting the training graphs by *preserving the important structural information*. However, we aim at the opposite: to *destroy* common structures in the training graphs, to allow ML approaches exploring more unseen graphs. ii) Graph optimization problems have constraints that must be strictly satisfied before considering solution quality. General graph-related tasks such as node/edge/graph classification do not have constraints and thus do not produce *infeasible solutions*. Therefore, our problem is different from but more challenging than general graph-related tasks. As will be illustrated in our experiments, general graph augmentation works [23, 29, 34, 35] perform poorly for our task.

We evaluate the performance of the proposed approaches with well-known graph optimization problems: MVC, MIS, and SAT on various graphs. We demonstrate that our proposed approaches benefit both reinforcement learning and supervised learning approaches, significantly outperforming the other nine baseline approaches, including the state-of-the-art graph augmentation works. Moreover, the proposed approach is able to significantly enhance the performance when the ML approach is trained on very biased graphs, such as 1-regular graphs[1].

The contributions of this paper are summarized as follows.

- We present an analysis on how the diversity of training graphs impacts the performance of ML approaches to graph optimization problems. To the best of our knowledge, this is the first paper that studies the *training graph diversity for ML approaches to graph optimization problems*.
- We formulate a new research problem, *DGMP* and analyze its NP-hardness. We propose a new framework, *LIGD*, with a new algorithm, *PASDM*, to effectively improve the diversity of training graphs and the performance of the ML approaches.
- We perform extensive experiments on various ML approaches to well-known graph optimization problems. Our proposed PASDM significantly improves the performance and outperforms the other nine baselines, including the state-of-the-art graph augmentation and deep learning-based graph generation approaches.

The rest of this paper is organized as follows. Sec. 2 presents the analysis results for the impact on training data diversity. Sec. 3 discusses the related works. Sec. 4 formally formulates the problem. Sec. 5 details the algorithm design. Sec. 6 presents the experimental results. Sec. 7 concludes this paper.

2 Preliminary Results

To address the first challenge in Sec. 1, we analyze the impact of training graph diversity and answer the research questions below.

- **RQ1:** *Does dissimilarity of testing and training graphs lead to performance degradation?*
- **RQ2:** *Does enhancing the diversity in training graphs lead to significant performance improvement?*

[1] In an 1-regular graph, each node has exactly one neighbor.

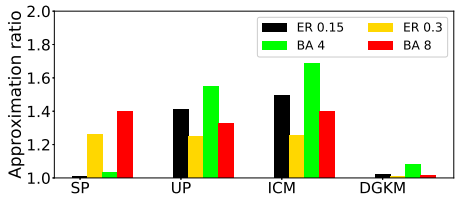


Figure 1: Testing performance for MVC (smaller approx. ratio indicates better result)

We conduct experiments with a reinforcement learning (RL) approach [12] on the well-known graph optimization problem, *Minimum Vertex Cover (MVC)*[2]. We compare the RL agent trained with four different training data generation approaches below.

Training graph generation approaches. The first three training graph generation approaches are listed below. i) *Single Edge Probability (*SP*).* This approach randomly generates 100 graphs by the Erdos-Renyi (ER) model [5] with a fixed edge probability $p = 0.15$. This generation approach is adopted by [12]. ii) *Uniform Edge Probability (*UP*)*, which randomly generates 100 graphs by the ER model with edge probability randomly sampled from $[0.1, 0.3]$. iii) *Intrinsic Curiosity Module (*ICM*).* This approach randomly generates 100 graphs by the ER model with fixed edge probability $p = 0.15$. Moreover, we enhance the RL agent by employing Intrinsic Curiosity Module (ICM) [21] that encourages the agent to explore more search space when finding the solution.

In addition to the three training data generation schemes described above, to validate the effectiveness of enhancing *diversity* to the training graphs and answer **RQ2**, we propose a baseline approach that generates diversified training graphs, named *Diverse Graph k-Means (DGKM).* Specifically, DGKM aims to construct a set of k diversified training graphs. At the beginning, DGKM randomly constructs r graphs with certain generation model, such as ER model [5]. In the next step, we compute the feature vector of each graph[3], and the feature vectors are aggregated into a feature matrix. With this feature matrix, each graph can be viewed as a point in the high dimensional space. DGKM then performs k-means on these points to form k clusters[4]. For each cluster C_i, DGKM identifies one graph that is closest to the cluster centroid. Finally, DGKM outputs the k graphs (one for each cluster) as the diversified training graphs. Please note that DGKM is an intuitive baseline which is used to demonstrate the impact on the diversity of the training graphs to model performance. For fair comparisons, we set $k = 100$ for DGKM to generate 100 training graphs.

Results. Fig. 1 presents the performance on *testing graphs* of the above four training graph generation approaches. A smaller approximation ratio (the closer to 1.0) indicates a better performance. Every result is averaged over 100 instances. Here, bars *ER 0.15* and *ER 0.3* indicate that *testing graphs* are generated by ER model [5] with edge probabilities $p = 0.15$ and $p = 0.3$, respectively. Similarly, bars *BA 4* and *BA 8* indicate that testing graphs are generated by BA model [1] with $m = 4$ and $m = 8$, respectively.

[2]Other ML approaches for various graph optimization problems, i.e., MIS and SAT, show similar trends as MVC and thus are omitted.
[3]The features include the aggregated information of *core numbers*, *node degrees*, and *clustering coefficients*, which will be introduced in Sec. 4.
[4]Here, the k-means algorithm computes the euclidean distance based on the normalized feature values.

Answer to RQ1. We first check the results of baseline SP in Fig. 1. Recall that SP generates the training graphs by ER model with a fixed edge probability $p = 0.15$. When the testing graphs are similar to the training ones, SP performs well, as shown by the bar *ER 0.15* (the similar edge connectivity distribution) and bar *BA 4* (similar density). However, when the testing graphs differ from the training ones, SP performs poorly, as shown by the bars *ER 0.3* and *BA 8* with much larger approximation ratios. This validates our claim and answers **RQ1**: *The ML approaches perform well with similar training and testing graphs. However, the performance drops significantly when training and testing graphs are dissimilar.*

Answer to RQ2. Observing the weaknesses of generating the training/testing graphs with a fixed edge probability (SP), we turn our attention to answering **RQ2**. To generate the training graphs with more diversity, one intuitive approach is to change the edge probability of ER model during the generation of training graphs (UP), to provide diversity to some degree. However, as shown in Fig. 1, UP performs even worse. This is because the generated training graphs are more like random ones and the RL agent has difficulties learning a good policy with a small number (i.e., $k = 100$) of training graphs. This motivates us to come up with a more complicated approach, i.e., employing *Intrinsic Curiosity Module* to encourage the RL agent to explore novel candidate solutions (ICM). However, ICM achieves much worse performance as shown in Fig. 1, because the poor diversity of the training graphs (recalling that this approach employs ER with fixed $p = 0.15$ to generate training graphs) limits the vision and generalization ability of the RL agent, even though ICM is equipped.

In contrast, Fig. 1 manifests that DGKM, which considers the diversity of the training graphs, significantly outperforms the other baselines. In fact, DGKM obtains the optimal solutions in most test instances. This is because it leverages the clustering algorithm to construct diversified training graphs based on certain important graph features, enabling the RL agent to effectively capture the important properties. In contrast, lacking of training graph diversity makes other baselines difficult to generalize to the testing instances.

The above results answer **RQ2**: *Enhancing the diversity of the training graphs indeed leads to significant performance improvements, as shown by* DGKM *in Fig. 1.* However, intuitively cluttering the training graphs (UP) or enhancing the capability of the RL agent (ICM) does not work. This also confirms that the problem studied in this paper is very challenging.

3 Related Works

Solving graph optimization problems with ML approaches. Conventional graph optimization problems are usually solved with hand-crafted algorithms, e.g., [4, 10, 25, 26]. Recently, employing machine learning (ML) approaches to solve graph optimization problems gains more research attention. Dai et al. propose to solve the well-known graph optimization problems, Minimum Vertex Cover (MVC), MaxCut, and Traveling Salesman Problem (TSP), with reinforcement learning [12]. Li et al. propose a supervised learning approach to predict the optimal solutions for the Maximum Independent Set (MIS) [16]. Selsam et al. propose a message passing neural network to predict the satisfiability of the Boolean Satisfiability (SAT) [24]. Vinyals et al. propose a sequence to sequence

architecture, named Pointer Net, with supervised learning to solve TSP [27]. Lauri et al. propose a learning-based approach to reduce the search space for Maximum Clique (MC) problem [15]. Nazari applies reinforcement learning to Capacitated Vehicle Routing Problem (CVRP) [19]. Kool et al. propose a reinforcement learning approach to a series of TSP extensions [14], while reinforcement learning-based approaches are proposed to effectively solve the extensions of team formation problems [3, 30]. Witnessing the popularity and importance of adopting machine learning approaches to solve graph optimization problems, we envisage that our analyses and the proposed approaches can be complement to these machine learning approaches that can effectively improve their performance.

Graph augmentation. Graph augmentation aims at augmenting the graphs (e.g., node attributes or graph topological features) to improve the robustness of graph neural networks [7, 13, 23, 33–35]. For example, GraphCL augments the graphs with four operations, i.e., node dropping, edge perturbation, attribute masking, and subgraph sampling, to produce positive/negative pairs for contrastive learning [34]. Zhu et al. employ contrastive learning with graph augmentation to improve the performance of node classification by separating the positive/negative pairs with the graph augmentation [35]. In addition, Feng et al. propose a graph augmentation approach to improve the robustness of GNNs against the over-smoothing problem [7], while Rong et al. propose to randomly drop edges in the graph to enhance the robustness of GNNs against overfitting and over-smoothing problems [23]. In addition, Wang et al. propose a graph augmentation approach by mixing up the receptive field subgraphs of nodes [29].

As mentioned earlier in Sec. 1, the goal of the above graph augmentation approaches are totally different from ours, because they aim to generate graphs that are *not significantly different* from the original graph. In contrast, our proposed approach improves the machine learning model's performance by *diversifying* the training graphs, i.e., to generate *significantly dissimilar* graphs to allow the model to explore more unseen graphs and improve its generalibility. Later in Sec. 6, we demonstrate that the proposed PASDM outperforms the state-of-the-art graph augmentation approaches significantly.

Graph generation with deep learning. A recent line of research studies graph generation with deep learning [17, 28, 32]. These methods aim to extract the patterns of the underlying edge connection probability of the given training graphs. For example, GraphGAN learns the graph representation by sampling edges and trains the neural network with an approach similar to Generative Adversarial Nets (GAN) [28]. GraphRNN generates new nodes and edges with RNNs [32]. The above works parameterize the edge distribution to generate the graphs. However, they neither aim to optimize the model performance trained with the generated training graphs nor diversify the generated training graphs. Therefore, their approaches are orthogonal to ours and cannot be directly applied.

4 Problem Formulation

To improve the performance and generalizability of the machine learning approaches, we propose an iterative learning framework, named *Learning with Iterative Graph Diversification (LIGD)*, as shown in Fig. 2. LIGD includes two major components, i) the machine

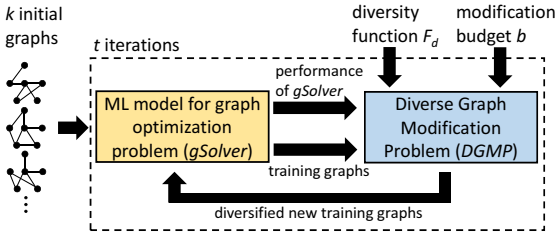


Figure 2: Overview of framework *LIGD*

learning model for solving the graph optimization problem (called *gSolver* hereafter), and ii) the diversified training graph producer (i.e., the algorithm that solves *Diverse Graph Modification Problem (DGMP)*). Here, *gSolver* can be any machine learning model that solves the specific graph optimization problem, which is viewed as a black-box module in the framework, whereas *DGMP* is the core module we aim to explore in this paper, which is formally formulated later. The main purpose of LIGD is to iteratively assess the performance of *gSolver* and construct diversified training graphs based on *gSolver*'s performance by solving DGMP.

Before formulating the DGMP, we first describe an important notion, *diversity (dissimilarity) of graphs*. For a graph g, we denote the node and edge sets of g by $V(g)$ and $E(g)$, respectively. Given two graphs g_1 and g_2 with the same number of nodes, i.e., $|V(g_1)| = |V(g_2)|$, a diversity function F_d measures the dissimilarity of g_1 and g_2, where a larger value of $F_d(g_1, g_2)$ indicates a more significant dissimilarity among them. Please note that the diversity function is viewed as a given parameter to our problem formulation and can flexibly include any factors. Further, given a set of k graphs $G = \{g_1, .., g_k\}$, the *total diversity* of G in terms of diversity function F_d, denoted as $\mathcal{D}_{F_d}(G)$, is the sum of the all-pair diversity among the k graphs in G, i.e., $\mathcal{D}_{F_d}(G) = \sum_{i=1}^{k} \sum_{j=i+1}^{k} F_d(g_i, g_j)$.

To measure the diversity of the graphs for function F_d, node embedding vectors or hand-crafted feature vectors can be employed. In this paper, to demonstrate that the proposed framework and algorithms effectively improves the performance of *gSolver* even with a very simple implementation F_d, we employ the following node features: *core number (core)*, *degree (deg)*, and *local clustering coefficients (cc)*. Please note that embedding vectors can be used as alternatives for these simple node features. In our implementation, each node in the graph g is associated with the above three feature values. For g, we compute the *mean*, *variance*, *minimum*, Q_1, Q_2, Q_3, and *maximum* values among the node features in g to form the feature vector to represent g. $F_d(g_i, g_j)$ is thus computed as the Euclidean distance between the feature vectors of g_i and g_j.

Specifically, given k initial training graphs $\widehat{G} = \{\widehat{g}_1, ..\widehat{g}_k\}$ where each graph has exactly s nodes, a modification budget b, and a diversity function F_d, the proposed problem, *Diverse Graph Modification Problem (DGMP)*, aims at performing at most b *edge modifications*, i.e., edge insertions or deletions, on each graph $\widehat{g}_i$ to generate a new set of k graphs $G = \{g_1, .., g_k\}$ such that the total diversity of the generated G (i.e., $\mathcal{D}_{F_d}(G)$) is maximized. Here, we focus on edge modifications but not node insertions/deletions because generating the graphs with a fixed node number is adopted by most graph generators, e.g., the Erdos-Renyi (ER) [5] and Barabasi-Albert (BA) [1] models. The proposed DGMP is formulated as follows.

Problem: Diverse Graph Modification Problem (DGMP)

Given: k initial graphs $\widehat{G} = \{\widehat{g}_1, ..\widehat{g}_k\}$ each having s nodes, modification budget b, and diversity function F_d.
Find: To construct k graphs $G = \{g_1, .., g_k\}$ by modifying each input graph $\widehat{g}_i \in \widehat{G}$ with at most b edge modifications (insertion/deletion).
Objective: To maximize the total diversity of the k generated graphs G, i.e., $\mathcal{D}_{F_d}(G)$, based on the diversity function F_d.

The above DGMP is challenging because the diversity should be optimized subject to a budget constraint b. Actually, the proposed DGMP is NP-hard as shown in Theorem 4.1 below.

Theorem 4.1. *DGMP is NP-hard.*

Proof. Please refer to Appendix A for details. □

5 Algorithm Design

Based on the LIGD framework illustrated in Fig. 2, we propose two approaches. The first approach, named *DSS*, is a baseline approach that generates diversified training graphs with the idea of *random modification*. To tackle the weaknesses of *DSS*, we further propose the second approach, PASDM, to construct diversified training graphs to effectively improve the performance of the ML model to graph optimization problems (i.e., *gSolver*).

5.1 Baseline Approach - Dense Subgraph Selection (DSS)

The first proposed approach, *DSS*, is a baseline approach which includes three major steps: i) *Random modification*, which randomly modifies the input training graphs to produce a set of new training graphs, ii) *performance evaluation*, which evaluates the testing performance of *gSolver* on each graph, and iii) *diversified training graph extraction*, which first transforms the set of training graphs into a *selection graph* g_x and then extracts the set of k mutually dissimilar training graphs with the help of g_x for the next iteration. The pseudo code of DSS is presented in Algo. 1 in Appendix B.

In the beginning, the set of k initial training graphs $G = \{g_1, ..., g_k\}$ are generated by any graph generator, where each g_i has exactly s nodes. Here, we employ DGKM (introduced in Sec. 2) to generated a set of diversified initial training graphs G. With the set of training graphs G, DSS then trains *gSolver* with each training graph $g_i \in G$.

Step 1. Random modification. Then, DSS enters the first step mentioned above. For each training graph g_i in G, DSS generates α new graphs by randomly modifying (adding or deleting) b edges in g_i. The generated new graphs form the set G', representing the candidate training graphs for the next iteration. Afterward, DSS proceeds to the second step, *performance evaluation*.

Step 2. Performance evaluation. In this step, we measure the performance of *gSolver*. If *gSolver* has good testing performance on graph $g'_i \in G'$, it indicates that *gSolver* well captures the properties of g'_i, and in the next iteration, we can include new and dissimilar training graphs to replace g'_i. To measure the performance of *gSolver* on each graph g'_i, we leverage the *performance estimator* Z, which is a function (either an exact, approximate, heuristic algorithm, or even *gSolver* trained previously for the current graph optimization problem) that produces the solution for g'_i. For example, for small-scale training graphs, an exact algorithm can be employed as Z to find the optimal solutions; Moreover, as *gSolver* is iteratively trained to improve its performance, we can also adopt the *gSolver* trained at the previous iteration as the performance estimator Z at the current iteration.

To evaluate the performance of *gSolver* on $g'_i \in G'$, DSS computes the objective value with Z for g'_i, i.e., $Z(g'_i)$. Then, DSS compares $Z(g'_i)$ with the objective value o_i *gSolver* obtained by calculating the ratio $r_i = \frac{o_i}{Z(g'_i)}$. If *gSolver* is solving a *minimization* problem, a smaller r_i value indicates that *gSolver* performs better on graph g'_i, i.e., *gSolver* is able to minimize the objective value; In contrast, if *gSolver* is solving a *maximization* problem, a larger r_i indicates *gSolver* performs better[5].

Step 3. Diversified training graph extraction. After measuring how *gSolver* performs on each graph g'_i and deriving r_i, DSS keeps the subset $G'' \subseteq G'$ of the graphs on which *gSolver* does not perform well. The purpose of keeping G'' is to allow *gSolver* having the chance to be trained on these graphs in the next iteration for better training performance. Here, parameter β controls the ratio of the graphs in G' that are retained in G''.

Recall that DGMP formulated in Sec. 4 aims at maximizing the *total diversity* of the k generated training graphs. To achieve this, DSS extract a set of k training graphs from G'' such that they are mutually dissimilar. DSS first constructs a new *selection graph* $g_x = \{V_x, E_x\}$, which is a complete graph such that each graph $g_i \in G''$ is represented as a node in V_x. For two nodes $v_i, v_j \in V_x$ (corresponding to $g_i, g_j \in G''$, respectively), the edge weight of edge (v_i, v_j) is assigned as the dissimilarity of g_i and g_j, i.e., $F_d(g_i, g_j)$.

Therefore, extracting a subgraph $\widehat{g} \subseteq g_x$ of $|V(\widehat{g})|$ nodes indicates that exactly $|V(\widehat{g})|$ graphs in G'' are extracted, and the total edge weights of $\widehat{g}$ is indeed the *total diversity* of the $|V(\widehat{g})|$ graphs. Here, DSS extracts $\widehat{g} \subseteq g_x$ such that i) $|V(\widehat{g})| = k$, i.e., k training graphs should be identified from G'' for the next iteration, and ii) the total edge weight is maximized in order to maximize the total diversity $\mathcal{D}_{F_d}(\widehat{g})$. To achieve the above goal, DSS extracts the *Weighted Dense k-Subgraph (DkS)* [6] from g_x with the state-of-the-art approach [2]. After extracting $\widehat{g}$, DSS identifies the corresponding k new training graphs to form $G = \{g_1, .., g_k\}$ and passes G to *gSolver* to start the next iteration until t iterations has been performed.

Weaknesses of DSS. Although the above DSS generates k training graphs by maximizing their total diversity, there are some major weaknesses: i) At the random modification step, DSS randomly modifies the edges in the graphs, which does not specifically *diversify* the graphs by examining the structural properties of the graphs. As a consequence, the diversity of the graphs may be very limited. ii) In the early iterations, the ML model *gSolver* is not well-trained and is more like a random policy. In this case, DSS tends to keep g'_i with smaller $Z(g'_i)$ and causes bias. In fact, sometimes the training of DSS does not converge. iii) For each graph g_i in G, a set of α new graphs are generated, leading to a massive number of graphs in G'. In step 2, *gSolver* needs to obtain the objective value on each $g' \in G'$, which is time consuming.

5.2 Proposed Algorithm - PASDM

To address the weaknesses of DSS introduced previously, we propose a more advanced algorithm, Performance-Aware Structure

[5] In fact, if the performance estimator Z is the exact algorithm that returns the optimal solution to g'_i, r_i is indeed the *approximation ratio*.

Diversifying Modification (PASDM) to better maximize the diversity of the constructed training graphs, which further diversifies the graphs by considering structural information. Please note that the *passive* strategy of DSS i.e., first randomly modifying original graphs and then filtering them in respect to the diversity, may greatly limit the diversity of the generated graphs. In contrast to DSS, the proposed PASDM adopts an *active* strategy that directly examines the structural property of the graphs and actively adds/deletes the edges to maximize the diversity. As later shown in the experiments (Sec. 6), the proposed PASDM is able to significantly boosts the performance of the ML model, outperforming the other baselines. Further, as illustrated in Sec. 6.3, with the help of PASDM, *gSolver* is able to recover from heavily-biased initial training data, i.e., 1-regular graph, where each node in the graph has degree exactly 1, and perform well on the testing data of various graph distributions.

The proposed PASDM consists of two major steps, *performance evaluation* and *structural-aware graph diversification*. Intuitively, given a set of training graphs G, PASDM trains *gSolver* on G and evaluates the performance. If *gSolver* performs well on some training graph $g_i \in G$, PASDM actively modifies g_i to produce g_i' based on structural properties to maximize the dissimilarity of g_i to other graphs in G. The pseudo code of PASDM is presented in Algo. 2 in Appendix B.

PASDM works as follows. In the beginning, similar to DSS, we generate the k initial training graphs G with DGKM. Then, *gSolver* is trained on G, and we enter the *performance evaluation* step.

Step 1. Performance evaluation. After *gSolver* is trained on G, we evaluate its performance on each graph g_i in G. If *gSolver* performs well on certain graph g_i, it is not necessary to keep g_i in the training set of the next iteration. In contrast, if *gSolver* performs poorly on g_i, the model needs to pay more attention on g_i in the next iteration and thus g_i should be kept in the training set. Similar to DSS, we measure *gSolver*'s performance with the help of the performance estimator Z, i.e., to calculate $r_i = \frac{o_i}{Z(g_i)}$, where o_i is the objective value obtained by *gSolver* on g_i. Here, hyperparameter γ acts as a performance threshold. If $r_i \geq \gamma$ (for *minimization* problems) or $r_i \leq \gamma$ (for *maximization* problems), *gSolver* is considered performing poorly on g_i and thus g_i should be retained in the training set for the next iteration; Otherwise, *gSolver* performs well on g_i and we enter the second step, *structural-aware graph diversification*.

Step 2. Structural-aware graph diversification. If *gSolver* performs well on graph g_i, PASDM then actively modifies g_i with at most b edge modifications and constructs the corresponding training graph $\widetilde{g_i}$ to maximize the dissimilarity among g_i and other graphs in G. This step, unlike the random modification step in DSS, significantly improves the diversity of the training graphs and boosts the performance of *gSolver*.

Intuitively, if two graphs g_i and g_j are similar to each other, there should exist certain common or similar structures for g_i and g_j. Therefore, our idea is to identify the nodes that are involved in these similar structures and then eliminate such similar structures with edge modifications on these nodes. In this way, we are able to significantly increase the dissimilarity among the graphs g_i and g_j. In the proposed PASDM, when *gSolver* performs well on g_i, PASDM then proceeds to modify g_i with b edge modifications to produce a new training graph $\widetilde{g_i}$. To achieve this, our algorithm includes two sub-steps: *node matching for similar structures* and *edge modification*, as detailed below.

Step 2-a. Node matching for similar structures. This step is outlined in Algo. 3 in Appendix B. Given two graphs g and g', we first compute the standardized feature (or embedding) vector $\mathbb{X}_i$ for each node v_i in g and g'. After deriving the features of each node in g and g', we now turn our attention to identify the set of node pairs that are structurally-similar in terms of the feature vectors. Since the number of nodes in g and g' are both s, our task is to find a set of s pairs of nodes $\{(v_{(1)}, v'_{(1)}), (v_{(2)}, v'_{(2)}), ..., (v_{(s)}, v'_{(s)})\}$ such that $v_{(i)} \in g, v'_{(i)} \in g', \forall 1 \leq i \leq s$, and $\{v_{(i)}\}_{1 \leq i \leq s}, \{v'_{(j)}\}_{1 \leq j \leq s}$ each is a permutation of the node sets of g (i.e., $V(g)$) and g' (i.e., $V(g')$), respectively.

A straightforward approach for the above task is to employ a greedy approach that sequentially selects one pair of un-selected nodes $v_{(i)} \in g$ and $v'_{(i)} \in g'$ with the minimum distance of their feature vectors. However, this greedy approach may lead to sub-optimal solutions. After a careful examination, we formulate the above task as a weight bipartite matching problem. Specifically, we construct a complete bipartite graph $g_p = (V_1, V_2, E_p)$, where V_1 consists of the nodes in g, and V_2 includes the nodes in g'. For any edge (v_i, v_j) where $v_i \in V_1$ and $v_j \in V_2$, the edge weight is assigned as the distance among their feature (embedding) vectors[6].

After the complete bipartitle graph g_p is constructed, we extract the minimum weight bipartite matching [11] on g_p to find the s pairs of the matched nodes. We then store the distances of each matched node pair in a list U. For example, if $v_i \in g$ and $v'_i \in g'$ are matched and the distance between their feature vectors is 0.3, then $U(v_i, v'_i) = 0.3$.

Step 2-b. Edge modification. In the second step, given the graph to be modified g and a reference graph g', we start modifying graph g to maximizes its dissimilarity toward g'. Here, PASDM chooses g' as the graph in G which is most similar to g, i.e., $g' = \arg\min_{\overline{g} \in G} F_d(\overline{g}, g)$. This allows g to provide more diversity against g' after g is modified. Recall that the nodes in g and g' were matched in the previous step, and for a pair of matched nodes $v_i \in g$ and $v'_i \in g'$, their feature vectors are very similar, i.e., they share similar or common structural properties in the graphs. Therefore, by modifying the edges incident to v_i, we are very likely to increase the dissimilarity among the two graphs. This step is outlined in Algo. 4 in Appendix B.

Specifically, for a pair of nodes $v_i, v_j \in g$ and their corresponding matched nodes $v'_i, v'_j \in g'$, we denote $\Delta(v_i, v_j) = U(v_i, v'_i) + U(v_j, v'_j)$ the *difference of edge modification among v_i and v_j*, i.e., the potential difference made by modifying the edge (v_i, v_j) in g. The intuition is that, if $\Delta(v_i, v_j) = U(v_i, v'_i) + U(v_j, v'_j)$ is small, it indicates that $v_i \in g$ with its matched node $v'_i \in g'$, as well as $v_j \in g$ with its matched nodes $v'_j \in g'$ are structurally similar. Therefore, if we modify the edge in graph g, i.e., we delete edge (v_i, v_j) from g if edge (v_i, v_j) was originally in g, or we add edge (v_i, v_j) to g if the edge was not in g, may increase the dissimilarity of g and g' significantly. Following this idea, we extract the top-b pairs of nodes in g, and modify these b edges to improve the dissimilarity of g and g'. Finally,

[6] We employ Euclidean distance in our implementations. However, other distance measures can also be applied.

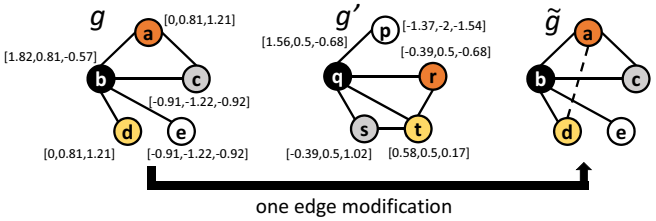


Figure 3: Example of edge modification (step 2-b) with $b = 1$

the revised graphs are returned and added to the training graph set G of the next iteration.

One may wonder if edge (v_i, v_j) does not exist in the original graph g, but its matched counterpart (v'_i, v'_j) exists in graph g', does adding edge (v_i, v_j) make g more similar to g'? The answer is *NO*. Although adding edge (v_i, v_j) gives the impression (based on local connectivity) that g and g' become similar, this is *not true* in a higher-order perspective of graph structure. Actually, adding (v_i, v_j) significantly changes v_i and v_j's common neighbors' clustering coefficients and also the core numbers of many other nodes in g, resulting in significant increase of dissimilarity. When designing PASDM, our preliminary results on 2,000 graphs show that 81% of such cases indeed increase the dissimilarity (clustering coefficients change most significantly).

Example 5.1. Fig. 3 presents an example of the above edge modification. The two graphs g and g' each has 5 nodes, and the feature vector is plotted besides each node. The pairs of nodes matched in step 2-a are: $\{(v_a, v_r), (v_b, v_q), (v_c, v_s), (v_d, v_t), (v_e, v_p)\}$, which are marked in same color in Fig. 3. We assume $b = 1$. Among the potential edges to be modified, $\Delta(v_a, v_d) = U(v_a, v_r) + U(v_d + v_t)$ is 1.29, which is the minimum among all the other pairs. Therefore, we modify edge (v_a, v_d) in g, i.e., add edge (v_a, v_d) to g to produce the new graph $\widetilde{g}$.

The time complexity of PASDM (excluding the training and inference time of *gSolver*) is $O(tk(sd + b))$, assuming each node in g_p is connected to the top-d most similar nodes. Please note that PASDM is performed in the *training time* of the models and takes much less time than training *gSolver*.

6 Experimental Results

In the following, we evaluate the performance of the proposed approach by comparing with multiple baselines on various graph optimization problems.

6.1 Setup

Compared methods. We compare seven approaches for training graph generation. The first three baselines are those described in Sec. 2: i) SP, ii) UP, and iii) ICM, where each generates 1,000 training graphs here. Additional four approaches are: iv) *GraphGAN (*GAN*)* [28], which generates graph representations by training with the generator and discriminator. We train GraphGAN with ER model ($p = 0.15$) and sample 1,000 graphs from its generator. v) DGKM, the same as described in Sec. 2, where $r = 10,000, k = 1,000$. vi) DSS and vii) PASDM, which are the approaches proposed in Sec. 5.1 and Sec. 5.2, respectively. We also employ three state-of-the-art graph augmentation approaches as baselines: iix) DE [23], which augments the graphs by randomly dropping edges (dropping probability set as 0.2). ix) GCA [35], which drops edges based on degree centrality (edge removing probability set as 0.2. x) GraphCL [34], which augments the graphs by subgraph sampling with random walk. Here, we sample 10 nodes for each augmentation with subgraph sampling probability set as 0.2. xi) Mixup [29], which randomly pairs nodes and mixes their receptive field subgraphs as augmentation. We set $\lambda = 0.7$.

DSS and PASDM follow the proposed LIGD framework, where *gSolver* goes through t ($t = 20$) iterations. In each iteration, *gSolver* is trained on k ($k = 50$) training graphs. For fair comparisons, we let the other baselines generate $t \times k = 1,000$ training graphs to train *gSolver*. In other words, we compare the performance of *gSolver* such that it is trained on the same number of training graphs for each approach. The default parameters are: $k = 50$, $s = 50$, $b = 20$, $t = 20$, $\alpha = 0.8$, $\beta = 0.5$, and $\gamma = 1.0$.

Optimization problems. We evaluate the approaches on the following three classical problems[7]: Minimum Vertex Cover (MVC), Maximum Independent Set (MIS), and Boolean Satisfiability (SAT). In our experiments, we report the testing performance of the ML approach (*gSolver*) in terms of approximation ratio (approx. ratio). The closer the approx. ratio to 1, the better the solution quality is (the optimal solution has an approx. ratio 1).

Testing datasets. We evaluate the approaches with multiple real and synthetic datasets. We adopt the real co-author network dataset, *DBLP* with 317K nodes and 1.05M edges [31]. For SAT, we employ the standard benchmark dataset, SATLIB dataset [9]. In addition, to better illustrate the relationship between diversity and performance, we generate testing data with the following three models, which are widely adopted to evaluate the performance of ML approaches to graph optimization problems. i) Erdos-Renyi (ER) model [5] with edge probability $p = \{0.15, 0.3, 0.5\}$, ii) Barabasi-Albert (BA) model [1] with $m = \{4, 8, 13\}$, and iii) power-law cluster graph (POW) [8] with $m = \{4, 8, 13\}$ and $p = 0.25$. For each testing graph generation approach above, we generate 100 instances.

6.2 Results on Minimum Vertex Cover (MVC)

We first compare the results on MVC with different training graph generation approaches mentioned above. Here, we adopt the RL-based approach [12] to solve MVC, trained on training graphs with 50 to 300 nodes. For each approach, we train the agent for 15,000 episodes. We adopt a greedy node selection algorithm as a benchmark heuristic Z, which greedily selects the node with the greatest uncovered edges.

Comparisons with the optimal solutions. Table 2 presents the results, where the testing graph sizes are within $[50, 100]$. Since the *approximation ratios* reported here are calculated based on the *true optimal solutions*[8], we employ small testing graphs because otherwise we are unable to obtain the optimal solutions within reasonable time. Our proposed PASDM obtains near-optimal solutions in all cases (i.e., approx. ratios are close to 1). PASDM significantly outperforms all the baselines when the training graphs differ from the testing graphs, i.e., $p = 0.5$ for ER, $m = 8, 13$ for BA and POW, and achieves near-optimal performance when the training and testing graphs are similar (i.e., generated with ER with $p = 0.15$). This is

[7] Our proposed approach can be integrated with ML approaches that solve various graph optimization problems, not limited to the three problems here.
[8] Some approx. ratios presented in [12] are not calculated based on optimal solutions.

because the proposed PASDM carefully generates diversified training graphs and iteratively modifies them during the training, to optimize the generalizability of the model.

SP trains the RL agent with training data generated by ER (edge probability $p = 0.15$), and unsurprisingly, SP performs well for testing on ER with $p = 0.15$. However, when the testing graphs differ from the training ones, the performance of SP significantly deteriorates. The intuitive approach UP, which aims at diversifying the training graphs by ER with random edge probabilities, performs even worse because such generated graphs are more similar to random graphs and the agents may hardly capture the underlying patterns. For ICM, we expect the agent to explore more novel states to improve its generalizability. However, the results are quite discouraging. This illustrates that simply enhancing the RL agent cannot effectively enable the agent to generalize over different graphs. GAN is trained with ER $p = 0.15$, therefore, its performance is similar to SP in most cases. Moreover, equipped with the sampling procedure, GAN performs slightly better than SP in some cases. However, it still lacks the diversity for the agent to learn a generalized policy. The graph augmentation approaches, i.e., DE, GCA, GraphCL, and Mixup, all perform poorly because they aim to augment the original graphs to generate slightly different graphs. The RL agent can hardly gain insights from these similar training graphs.

For proposed baseline, DGKM, employs k-means to diversify the training graphs and shows better performance and generalibility. DGKM performs particularly well when the testing graphs are similar (edge distribution and density) to the training ones, i.e., ER ($p = 0.15$ and $p = 0.3$), BA and POW ($m = 8$). This indicates that with carefully selected diverse training graphs, the agent's performance could be significantly improved and adapt to different graph distributions. However, DGKM does not always perform well. This indicates that in addition to generating diverse graphs at the beginning, iteratively modifying the graphs during training based on *gSolver*'s performance is also critical.

The other proposed baseline, DSS, shows inferior performance compared to DGKM and PASDM. Unlike PASDM, the random modification and selection of graphs performed by DSS results in biases and thus the agent cannot adapt to the testing graphs. This also confirms that we should pay special attention when iteratively generating the training graphs in each step.

Connecting performance with training graph diversity. To validate that increasing training graph diversity results in performance improvement, we further present the *total diversity*, i.e., $\mathcal{D}_{F_d}(\cdot)$ (defined in Sec. 4), of the training graphs generated by each approach in the last column of Table 2. To better illustrate, we normalize the total diversity values by the number of training graphs (i.e., 1,000 training graphs for each approach in each setting). Our proposed PASDM achieves a diversity of 6.21, outperforming all the other baselines. This validates our claim, and our proposed PASDM effectively diversifies training graphs to boost the performance.

6.3 Results on MIS

To demonstrate that the proposed framework works for various categories of ML approaches and can significantly improve their performance even when they are trained on heavily biased training graphs, we present the results on MIS by employing a widely adopted *supervised* learning approach [16] ([16] is detailed in Appendix C).

Table 2: Testing performance (approx. ratio) and normalized total diversity ($\mathcal{D}_{F_d}(\cdot)$) for various approaches trained with graphs generated by ER ($p = 0.15$) on MVC. A smaller ratio indicates a better result

	ER (p)		BA (m)		POW (m)		
	0.15	0.5	8	13	8	13	$\mathcal{D}_{F_d}(\cdot)$
SP [12]	**1.01**	1.15	1.4	1.34	1.43	1.33	5.14
UP	1.41	1.14	1.33	1.3	1.42	1.29	4.03
ICM [21]	1.5	1.15	1.4	1.34	1.43	1.33	5.44
GAN [28]	1.04	1.14	1.19	1.31	1.2	1.28	3.75
DGKM	1.02	1.1	1.02	1.31	1.02	1.18	5.07
DSS	1.45	1.15	1.25	1.34	1.28	1.32	4.53
DE [23]	1.49	1.15	1.4	1.34	1.43	1.33	4.14
GCA [35]	1.02	1.12	1.36	1.21	1.37	1.25	5.06
GraphCL [34]	1.49	1.15	1.4	1.34	1.43	1.33	5.04
Mixup [29]	1.33	1.15	1.35	1.28	1.40	1.27	4.78
PASDM (ours)	**1.01**	**1.01**	**1.01**	**1.02**	**1.01**	**1.04**	**6.21**

Results on real dataset. We present the results on *DBLP* in Table 3. The training graphs are generated by ER with $p = 0.15$, where training graph sizes are within $[100, 500]$. The 100 testing graphs are sampled by Breadth-First Search from the DBLP dataset, where their sizes are within $[100, 1000]$. Due to the large graph size, we are unable to obtain the optimal solutions within a reasonable amount of time. Therefore, we present the *size ratio* of each approach, which is calculated as dividing the objective value of the proposed PASDM by the objective value of each approach, and a smaller ratio indicates a better solution. As shown in Table 3, the proposed PASDM outperforms the other baselines, because the testing graphs (DBLP) are much dissimilar to the training graphs (generated by ER). In this case, the proposed PASDM effectively enhances the model's performance by providing more diversified training graphs.

Table 3: Testing performance for MIS on *DBLP*. The sizes of testing graph range from 100 to 1000. A smaller value indicates a better result

SP	UP	GAN	DGKM	DSS	DE	GCA	GraphCL	Mixup	**PASDM**
1.12	1.06	1.07	1.06	1.06	1.17	1.14	1.19	1.08	**1.0**

Training with heavily biased data on MIS. To demonstrate the effectiveness of PASDM, we consider an extreme case that the training graphs are heavily biased. Here, we test PASDM by training the supervised ML approach for MIS [16] on a special graph class, 1-regular graph, and testing on various graphs generated with ER, BA, and POW models. In an 1-regular graph, each node has degree exactly 1. It is difficult for an ML approach to learn a good heuristic from such training graphs because an 1-regular graph includes a large number of components, each consisted of two nodes and one edge connecting them. ML approaches trained on such graphs can hardly generalize its insights to the general testing graphs.

We generate the initial training graphs as a set of 1-regular graphs, with sizes in $[50, 100]$. Here, we would like to understand whether PASDM can quickly adapt to this kind of heavily-biased graphs and improve the performance of the ML model [16]. The results (in terms of approximation ratio) of testing on various general graphs generated by ER, BA, and POW are listed in Table 4, where each result is averaged over 50 test instances with sizes in $[50, 100]$, to allow us computing the optimal solutions as the benchmark. Please note

that SP and UP are not compared here because they cannot generate 1-regular graphs. Equipped with PASDM, the proposed PASDM significantly outperforms the graph augmentation approaches, i.e., DE, GCA, GraphCL, Mixup, and its counterpart without PASDM, i.e., applying [16] directly. This manifests that PASDM is also very effective even on the heavily-biased training graphs.

Table 4: Testing performance when trained with 1-regular graphs (heavily biased graphs) for MIS (approx. ratio)

	ER (p)			BA (m)			POW (m)		
	0.15	0.3	0.5	4	8	13	4	8	13
Directly apply [16]	1.88	2.01	2.00	2.11	2.32	3.20	2.13	2.48	3.21
DE	1.87	2.03	2.00	2.11	2.33	3.20	2.15	2.50	3.21
GCA	1.04	1.07	1.08	1.04	1.06	1.06	1.04	1.07	1.13
GraphCL	1.75	1.43	1.33	2.19	2.17	2.18	2.27	2.54	1.16
Mixup	1.81	1.57	1.32	1.98	2.04	2.27	2.41	2.24	1.62
PASDM	**1.02**	**1.04**	**1.04**	**1.02**	**1.03**	**1.02**	**1.01**	**1.02**	**1.04**

Additional experiments. We further demonstrate that PASDM outperforms the other baselines for MIS in Appendix D. We also present the results on SAT and the sensitivity tests on MVC in Appendices E and F, respectively.

7 Conclusion

In this paper, we address the important problem that when the training and testing graphs are dissimilar, ML model performance significantly deteriorates. We formulate a new problem, DGMP. We first propose two baseline approaches to DGMP. Then, a more advanced approach, PASDM, is proposed based on the insights of the baseline approaches. Experimental results confirm that our proposed PASDM significantly boosts the performance of various ML approaches to graph optimization problems.

Acknowledgment

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant MOST 111-2636-E-007-019-.

References

[1] Rka Albert and Albert lszl Barabsi. 2002. Statistical mechanics of complex networks. *Rev. Mod. Phys* (2002), 47–97.
[2] Aditya Bhaskara, Moses Charikar, Eden Chlamtac, Uriel Feige, and Aravindan Vijayaraghavan. 2010. Detecting high log-densities: an O (n 1/4) approximation for densest k-subgraph. In *Proceedings of the forty-second ACM symposium on Theory of computing*. 201–210.
[3] Chih-Chieh Chang, Ming-Yi Chang, Jhao-Yin Jhang, Lo-Yao Yeh, and Chih-Ya Shen. 2022. Learning to Extract Expert Teams in Social Networks. *IEEE Transactions on Computational Social Systems* (2022).
[4] Yi-Ling Chen, De-Nian Yang, Chih-Ya Shen, Wang-Chien Lee, and Ming-Syan Chen. 2018. On efficient processing of group and subsequent queries for social activity planning. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2018), 2364–2378.
[5] P. Erdos and A Renyi. 1960. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Academy of Sciences*. 17–61.
[6] U. Feige, D. Peleg, and G. Kortsarz. 2001. The dense k-subgraph problem. *Algorithmica* 29 (2001), 410–421.
[7] Wenzheng Feng, Jie Zhang, Yuxiao Dong, Yu Han, Huanbo Luan, Qian Xu, Qiang Yang, Evgeny Kharlamov, and Jie Tang. 2020. Graph Random Neural Network for Semi-Supervised Learning on Graphs. *Advances in Neural Information Processing Systems* (2020).
[8] Petter Holme and Beom Jun Kim. 2002. Growing scale-free networks with tunable clustering. *Phys. Rev. E* 65 (Jan 2002), 026107. Issue 2.
[9] Holger H. Hoos and Thomas Sttzle. 2000. SATLIB: an online resource for research on SAT. IOS Press, 283–292.
[10] Bay-Yuan Hsu, Yi-Feng Lan, and Chih-Ya Shen. 2018. On automatic formation of effective therapy groups in social networks. *IEEE Transactions on Computational Social Systems* 5, 3 (2018), 713–726.
[11] Richard M. Karp. 1980. An algorithm to solve the m × n assignment problem in expected time O(mn log n). *Networks* 10 (1980), 143–152.
[12] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 6348–6358.
[13] Johannes Klicpera, Stefan Weißenberger, and Stephan Günnemann. 2019. Diffusion improves graph learning. *Advances in Neural Information Processing Systems* 32 (2019), 13354–13366.
[14] Wouter Kool, Herke van Hoof, and Max Welling. 2019. Attention, Learn to Solve Routing Problems!. In *International Conference on Learning Representations*.
[15] Juho Lauri and Sourav Dutta. 2019. Fine-grained search space classification for hard enumeration variants of subset problems. In *The Thirty-Third AAAI Conference on Artificial Intelligence*. 2314–2321.
[16] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. 2018. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 539–548.
[17] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. 2019. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems 32*. 4255–4265.
[18] Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. 2020. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. In *AAAI Workshop on Deep Learning on Graphs: Methodologies and Applications*.
[19] MohammadReza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takac. 2018. Reinforcement Learning for Solving the Vehicle Routing Problem. In *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 9839–9849.
[20] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takáč. 2018. Reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:1802.04240* (2018).
[21] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. 2017. Curiosity-driven exploration by self-supervised prediction. In *ICML*.
[22] Marcelo Prates, Pedro HC Avelar, Henrique Lemos, Luis C Lamb, and Moshe Y Vardi. 2019. Learning to solve np-complete problems: A graph neural network for decision tsp. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 4731–4738.
[23] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2019. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *International Conference on Learning Representations*.
[24] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. 2019. Learning a SAT solver from single-bit supervision. In *International Conference on Learning Representations*.
[25] Chih-Ya Shen, CP Kankeu Fotsing, De-Nian Yang, Yi-Shin Chen, and Wang-Chien Lee. 2018. On organizing online soirees with live multi-streaming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.
[26] Chih-Ya Shen, De-Nian Yang, Wang-Chien Lee, and Ming-Syan Chen. 2016. Spatial-proximity optimization for rapid task group deployment. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10, 4 (2016), 1–36.
[27] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2692–2700.
[28] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. GraphGAN: graph representation learning with generative adversarial nets. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18)*. 2508–2515.
[29] Yiwei Wang, Wei Wang, Yuxuan Liang, Yujun Cai, and Bryan Hooi. 2021. Mixup for node and graph classification. In *Proceedings of the Web Conference 2021*. 3663–3674.
[30] Chen-Hsu Yang, Hong-Han Shuai, Chih-Ya Shen, and Ming-Syan Chen. 2021. Learning to Solve Task-Optimized Group Search for Social Internet of Things. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[31] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.
[32] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. 2018. GraphRNN: generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*. PMLR, 5708–5717.
[33] Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. 2021. Graph contrastive learning automated. *International Conference on Machine Learning* (2021).
[34] Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. 2020. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems* 33 (2020), 5812–5823.
[35] Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. 2021. Graph contrastive learning with adaptive augmentation. In *Proceedings of the Web Conference 2021*. 2069–2080.

Supplemental Materials: Technical Appendix

A Proof of NP-hardness of DGMP

Theorem A.1. *DGMP is NP-hard.*

Proof. We prove that DGMP is NP-hard with the reduction from *Graph Isomorphism (GI)*. Given two graphs g_i and g_j, they are isomorphic if the number of components are the same (i.e., the same node and edge numbers), and the edge connectivity is retained (i.e., the same degree sequences). The decision problem of GI determines whether g_i and g_j are isomorphic. We transform an instance of the GI problem to an instance of the proposed DGMP as follows. Given g_i and g_j, each having s nodes, we create two initial graphs $G = \{\widehat{g}_i, \widehat{g}_j\}$ each having s nodes, and a modification budget $b = 0$. We set the dissimilarity function F_d such that that $F_d(\widehat{g}_i, \widehat{g}_j) = 0$ if and only if there exists a permutation of the node degree sequence of $\widehat{g}_1$ that is equal to the node sequence of $\widehat{g}_2$; Otherwise $F_d(\widehat{g}_i, \widehat{g}_j) = 1$.

We first prove the sufficient condition. If g_i and g_j are isomorphic, the corresponding instance in DGMP must have $F_d(\widehat{g}_i, \widehat{g}_j) = 1$. For the necessary condition, if $\widehat{g}_i$ and $\widehat{g}_j$ maximizes the objective value of DGMP i.e., $F_d(\widehat{g}_i, \widehat{g}_j) = 1$ in this case, GI must determine g_i and g_j isomorphic due to the same node numbers and the same degree sequences. This proves DGMP NP-hard. The theorem follows. □

B Pseudo codes

The pseudo code of DGKM is presented in Algo. 1, the pseudo code of PASDM is presented in Algo. 2. Algo. 3 and Algo. 4 present the two important steps of PASDM i.e., *Node-Match* and *Modify*.

Algorithm 1: Dense Subgraph Selection (baseline)

Input: Node number of training graphs s, number of output graphs k, number of iterations t, performance estimator Z, dissimilarity function F_d, modification budget b, mutation number α, retaining ratio β

Generate k initial training graphs $G = (g_1, g_2, ..., g_k)$ with Diverse Graph k-Means
for $i = 1$ *to* t **do**
  for *each graph* $g_i \in G$ **do**
    Train the model (*gSolver*) with g_i
  $G' \leftarrow \emptyset$
  for *each graph* $g_i \in G$ **do**
    Generate α graphs from g_i by randomly modifying g_i with b edges. Add those graphs to G'
  for *each graph* $g'_i \in G'$ **do**
    $o_i \leftarrow$ testing objective value of *gSolver* on graph g'_i
    $r_i \leftarrow \frac{o_i}{Z(g'_i)}$
  $G'' \leftarrow \beta$ (β is a ratio) out of the set of graphs in G' that *gSolver* performs the worst in respect to r_i
  Construct selection graph $g_x = \{V_x, E_x\}$ based on F_d, and find *Weighted Densest k-Subgraph (DkS)* $\widehat{g}$ with k nodes on g_x.
  Identify the k corresponding graphs of DkS $\widehat{g}$ in G'' to form the set of training graphs G for the next iteration

Algorithm 2: Performance-Aware Structure Diversifying Modification (PASDM)

Input: Node number of training graph s, number of output graphs k, number of training iterations t, performance estimator Z, dissimilarity function F_d, modification budget b, performance threshold γ

Generate k initial training graphs $G = (g_1, g_2, ..., g_k)$ with Diverse Graph k-Means
for $i = 1$ *to* t **do**
  for *each graph* $g_i \in G$ **do**
    Train the model (*gSolver*) with g_i
  for *each graph* $g_i \in G$ **do**
    $o_i \leftarrow$ testing objective value of *gSolver* on graph g_i
    $r_i \leftarrow \frac{o_i}{Z(g_i)}$
    if $r_i \geq \gamma$ *(for* minimization *problems;* $r_i \leq \gamma$ *for* maximization *problems)* **then**
      $\widetilde{g}_i \leftarrow g_i$
    else
      $g_j \leftarrow \arg\min_{g \in G} F_d(g, g_i)$
      $\widetilde{g}_i \leftarrow$ **Modify**(g_i, g_j, b)
  for $i = 1$ *to* k **do**
    $g_i \leftarrow \widetilde{g}_i$

Algorithm 3: Node-Match

Input: Two graphs g and g'
Output: Matched pairs $\mathbb{M}$, node dissimilarity list U

$D_g \leftarrow$ feature vector for each node in g
$D_{g'} \leftarrow$ feature vector for each node in g'
Construct complete bipartite graph $g_p = (V_1, V_2, E_p)$ such that $V_1 = V(g)$, $V_2 = V(g')$ and edge weight of edge (v_i, v_j), $v_i \in V_1, v_j \in V_2$ equals to the distance between the feature vectors of v_i and v_j
$\mathbb{M} \leftarrow$ minimum weight bipartite matching on g_p
$U \leftarrow$ distances of the matched node pairs
return $\mathbb{M}, U$

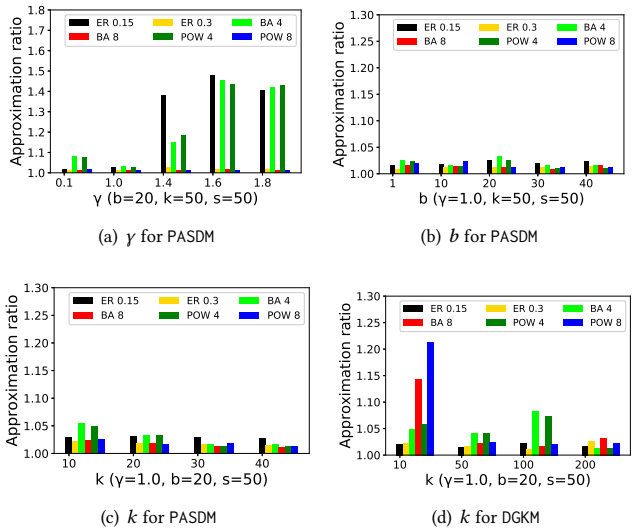
Algorithm 4: Modify

Input: Graph to be modified g, reference graph g', modification budget B
Output: modified graph $\widetilde{g}$

$\mathbb{M}, U \leftarrow$ **Node-Match**(g, g')
for $i = 1$ *to* b **do**
  Extract the un-selected pair of nodes $v_i, v_j \in g$ with the minimum $\Delta(v_i, v_j) = U(v_i, v'_i) + U(v_j, v'_j)$
  if *edge* $(v_i, v_j) \in g$ **then**
    Delete edge (v_i, v_j) from g
  else
    Add edge (v_i, v_j) to g
return $\widetilde{g} \leftarrow g$

C Description of Supervised Learning Approach for MIS

The supervised learning approach for MIS [16] is described as follows. For each training graph, we assign label **1** to a node if it is included in its corresponding optimal solution, and assign label **0** to it otherwise. A two-layer GCN is adopted to predict the probability if a node is in the optimal solution. Afterward, a greedy approach



(a) γ for PASDM (b) b for PASDM

(c) k for PASDM (d) k for DGKM

Figure 4: Sensitivity tests for MVC

iteratively selects the nodes to form the independent set based on the predicted values. In each iteration, one node incurring the highest predicted probability is selected into the set. Once a node is selected, its neighbor nodes are removed from the graph. The remaining graph is fed into the above GCN to obtain the predicted probabilities again. The above procedure repeats until the remaining graph becomes empty. The selected nodes form the resulting independent set.

D Additional Experimental Results on MIS

Table 5 lists the results. Here, we generate $k = 50$ graphs each having $s = 50$ vertices, along with an optimal solution for assigning labels. Since the training instance are small enough, we employ the exact algorithm as the performance estimator Z that generates the optimal solutions. As shown in Table 5, our proposed PASDM outperforms the baselines SP and UP in all cases for MIS because PASDM effectively diversifies the training graphs and iteratively refines them to optimize *gSolver*. In contrast, SP, although widely adopted as the standard training data generation approach for ML approaches to graph optimization problems, performs poorly compared to PASDM.

Table 5: Testing performance (approx. ratio) for MIS. A smaller ratio indicates a better result

	ER (p)			BA (m)			POW (m)		
	0.15	0.3	0.5	4	8	13	4	8	13
SP	1.05	1.08	1.09	1.02	1.05	1.03	1.02	1.04	1.13
UP	1.05	1.08	1.09	1.02	1.05	1.03	1.01	1.03	1.13
PASDM (ours)	**1.03**	**1.06**	**1.06**	**1.01**	**1.03**	**1.02**	**1.00**	**1.01**	**1.09**

E Results on SAT

For SAT, we adopt the supervised learning approach [16]. We employ the standard benchmark dataset, SATLIB dataset [9], for performance evaluation. We report the average satisfiability ratio of 1,000 instances, where each instance contains 20 variables and 91 clauses. All the instances are satisfiable. Here, we do not employ an additional heuristic or exact algorithm as the performance estimator Z. Instead, at iteration $(t+1)$, we employ *gSolver* trained at iteration t as Z. For the results, a larger satisfiability ratio indicates that the algorithm performs better. As shown in Table 6, our proposed PASDM significantly outperforms the other baselines, including the graph augmentation approaches (DE, GCA, GraphCL, Mixup) and deep learning-based graph generation approach (GAN). This is because PASDM effectively produces diversified training data and carefully adjusts them during training. In contrast, the other baselines perform poorly due to the ignorance of the diversity factor in the training graphs.

Table 6: Testing performance (satisfiability ratio) for SAT. A larger satisfiability ratio indicates a better result

SP	UP	GAN	DGKM	DSS	DE	GCA	GraphCL	Mixup	**PASDM**
0.19	0.26	0.27	0.26	0.0	0.19	0.2	0.24	0.21	**0.36**

F Parameter Sensitivity on MVC

Fig. 4 presents the sensitivity tests of PASDM on MVC solved by the RL-based approach [12]. Fig. 4(a) manifests that when γ is set too small (i.e., $\gamma = 0.1$) for this minimization problem, PASDM rarely modifies the training graph, and *gSolver* is trained on almost the same set of training graphs repeatedly. This results in poor generalization ability of *gSolver* on the testing graphs, leading to inferior performance compared to $\gamma = 1.0$. In contrast, setting γ too larger (i.e., $\gamma \geq 1.4$) results in over-frequent modifications of the training graphs, and the performance deteriorates as well because rapidly modifying graphs makes the training process unstable. Therefore, in our experiments, we set $\gamma = 1.0$ for PASDM, which strikes a good balance. Fig. 4(b) compares different modification budgets of PASDM. With different b values, PASDM achieves stable performance, because PASDM well leverages the modification budgets to diversify the training graphs and thus *gSolver* achieves good generalizability.

Figs. 4(c) and 4(d) present the approximation ratios of PASDM and DGKM for different k, i.e., the number of training graphs for each iteration. PADSM is able to achieve good approximation ratios even with small k, indicating the effectiveness of training graph diversification carried out by PASDM. When k becomes larger, the performance of *gSolver* in PASDM and DGKM gradually improve due to the larger number of training graphs.