PARETO AUTOMATIC MULTI-TASK GRAPH REPRESENTATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Various excellent graph representation learning models, such as graph neural networks (GNNs), can produce highly task-specific embeddings in an end-to-end manner. Due to the low transferability of learned embeddings and limited representational capabilities of handcrafted models, existing efforts cannot efficiently handle multiple downstream tasks simultaneously, especially in resource-constrained scenarios. This paper first tries to automatically search for multi-task GNN architectures to improve the generalization performance of GNNs through knowledge sharing across tasks. Because of possible task (objective) conflicts and complex dependencies of architectures and weights, the multi-task GNN architecture search is a bi-level multi-objective optimization problem (BL-MOP) to find a set of Pareto architectures and their Pareto weights, representing different trade-offs across tasks at upper and lower levels (UL and LL), respectively. The Pareto optimality of sub-problems results in each Pareto architecture corresponding to a set of Pareto weights, which is particularly challenging in deep learning with high training costs. For the first time, we propose a simple but effective differentiable multi-task GNN architecture search framework (DMTGAS) with convergence guarantees. Its search space contains aggregation and connection operations of shared layers. By introducing consistent task trade-offs for UL and LL, DMTGAS only alternately searches a single architecture and its weights to minimize multiple task losses via the gradient-based multi-objective optimizer, which neatly overcomes the above optimization difficulties. Experimental results on several tasks in three real-world graph datasets demonstrate the superiority of the GNNs obtained by our proposal compared with existing handcrafted ones.

1 INTRODUCTION

Graph neural network (GNN) Xu et al. (2019) captures the graph information by massage passing with the neighbors on graph-structured data. It has been widely used in several graph machine learning tasks, including graph classification (GC), node classification (NC), and link prediction (LP) Cai et al. (2018). A typical graph representation learning model follows an encoder-decoder neural module trained in an end-to-end fashionChami et al. (2022). Most commonly adopted GNNs can generate low-dimensional embedding vectors by aggregating node features and graph-structured information. Then embedding vectors are fed into the custom neural network to handle downstream graph machine learning tasks. The model tends to get highly task-relevant embedding vectors. However, the low transferability Buffelli & Vandin (2022) makes it unsuitable for multiple downstream tasks. Since graph tasks often do not appear alone and there is exploitable knowledge between them, multi-task graph representation learning has received much attention in practical applications in recent years, such as biomedical relation extraction Li & Ji (2019) and physico-chemical ADMET endpoints modeling Montanari et al. (2019).

In real-world scenarios, optimizing multiple tasks is difficult due to the limited representational capabilities of handcrafted models and task conflicts. For the former, existing multi-task graph neural networks (MTGNNs) adopt a fixed architecture serving a few specific tasks, which lacks flexibility Wang & Zhu (2021). With the emergence of many techniques for graph representation learning, given a few new tasks on graphs, manually identifying the optimal MTGNN model from scratch requires a lot of expertise. For the latter, the existing MTGNN training strategy combines multiple objectives (task losses) into a single objective, called linear scalarization (LS). This strategy leads

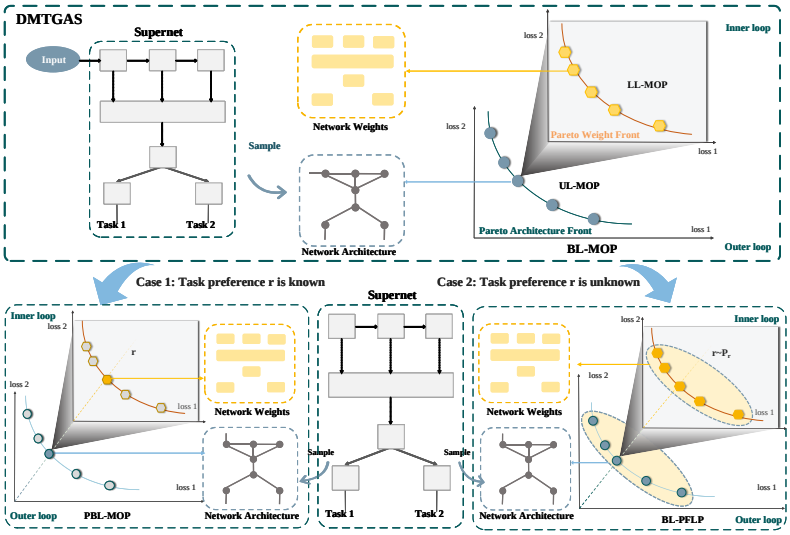


Figure 1: An illustration of our proposal for bi graph representation learning tasks.

to the inevitable lower performance when tasks conflict due to excessive parameter sharing between tasks. Since different conflicting tasks require trade-offs, multi-task graph representation learning is inherently a multi-objective optimization (MOO) problem Sener & Koltun (2018). Therefore, it is necessary to automatically search for a MTGNN model that matches various user-desired task preferences from a multi-objective perspective.

This paper first investigates automatic multi-task graph representation learning to search for user-desired MTGNN models, naturally modeled as a bi-level multi-objective optimization problem (BL-MOP), where each task loss is an objective. It aims to obtain a set of Pareto architectures and weights, representing different trade-offs across tasks at the upper level (UL) and lower level (LL), respectively. The optimization difficulty of BL-MOP is caused by the fact that each candidate architecture corresponds to a set of Pareto weights (see Figure 1). In general, a common method directly approximates the LL-MOP as a single objective problem to guarantee that each architecture corresponds to an optimal weightYe et al. (2021); Franceschi et al. (2018). But this method may lead to inconsistencies in task trade-offs at the UL and LL. Furthermore, the method's performance is severely affected by the manually tuned task loss weights.

In this paper, we propose a differentiable multi-task GNN architecture search framework (DMTGAS) with a convergence guarantee. In practice, the user may give the desired task trade-off (preference) $\boldsymbol{r}$. Two novel optimization schemes are presented to solve the BL-MOP by involving $\boldsymbol{r}$, as shown in Figure 1. 1) When $\boldsymbol{r}$ is unknown, a preference-consistent scheme is proposed to obtain a single Pareto optimal architecture and weights. It alternately optimizes a single MTGNN architecture and weights with the same task preference via preference-based MOO methods Mahapatra & Rajan (2020). 2) When $\boldsymbol{r}$ is unknown, the architecture and weights of a single model are optimized alternately via Pareto-front learning methods Ruchte & Grabocka (2021). At training time, the model couples the sampled preference to the feature space for joint training. At inference time, the model can generate the entire Pareto architecture set given different preferences. Both schemes guarantee the consistency of preference and do not need to maintain multiple weights for each candidate architecture in the optimization process, which cleverly avoids the above optimization difficulties.

In DMTGAS, a single MTGNN model is trained by the above schemes. All candidate architectures are sampled from a predefined search space (represented as a supernet). Through the continuous relaxation of the search space Huan et al. (2021), the multiple objectives are converted to differentiable. We evaluate DMTGAS and handcrafted methods on three real-world graph datasets (ENZYMES, PROTEINS, and DHFR), which are employed to perform multiple high-profile tasks, including node classification, graph classification, and link prediction. Experimental results demonstrate the superiority of the MTGNN model obtained by DMTGAS over the handcrafted ones and that DMTGAS can provide a model that matches user task preferences.

The main contributions of the work are summarized as follows: [1] 1) To improve the representation ability of handcrafted GNN models through knowledge sharing among downstream tasks, we first propose the automatic multi-task graph representation learning, which is modeled as a BL-MOP. 2) To overcome difficulties caused by each architecture corresponding to a Pareto weight set, two optimization strategies are presented by introducing consistent task preference $\boldsymbol{r}$, which only maintain one architecture and its weights instead of two solution sets. 3) We develop a differentiable multi-task GNN architecture search framework, called DMTGAS, to obtain entire Pareto front, or an architecture and its weights with a user-desired task preference. Furthermore, the convergence properties of DMTGAS are also proved.

2 Related Work

Multi-Task Graph Representation Learning Most works on multi-task graph representation learning have focused on learning an embedding model for several applications on graphs Avelar et al. (2019); Holtz et al. (2019); Li & Ji (2019); Montanari et al. (2019); Wang et al. (2020); Xie et al. (2020a;b); Buffelli & Vandin (2022); Sun et al. (2020); Wu et al. (2021). However, these application-specific handcrafted methods rely heavily on expert knowledge and ignore task conflicts. We provide more reviews in the Appendix 1.

Automatic Graph Representation Learning This section mainly focuses on the differentiable neural architecture search (DNAS), which has gained popularity in recent yearsGao et al. (2020); Wei et al. (2021). DNAS builds a single supernet that contains all operations Huan et al. (2021); Wang et al. (2021); Qin et al. (2021). A candidate operation is treated as a probability distribution over all possible operations. The objective function of DNAS is differentiable through continuous relaxation of the search space. Thus, gradient-based methods can jointly optimize the architecture and weights. Existing works mainly focus on the search space and search strategy to learn a graph representation model customized for a given task Wang & Zhu (2021). However, automatic learning methods for multiple tasks on graphs have not been noticed. We provide more reviews in the Appendix 1.

Gradient-Based Multi-Objective Optimization for Multi-Task Learning Due to the inevitable objective conflicts when learning multiple machine learning tasks by a model simultaneously, gradient-based multi-objective optimization for multi-task learning has been widely studied. Multi-objective optimization (MOO) aims to find a set of Pareto solutions (called Pareto front in the objective space) with different objective trade-offs. Sener et al. Sener & Koltun (2018) firstly employed MGDA to train a deep multi-task neural network to generate a Pareto stationary solution. Then, a series of preference-based methods are proposed to generate a Pareto solution with a given task preference (task trade-off) Lin et al. (2019); Mahapatra & Rajan (2020). For example, an exact Pareto optimal (EPO) search can find an exact Pareto optimal solution by combining gradient descent and controlled ascent Mahapatra & Rajan (2020). However, most of these methods need to learn a model from scratch separately for each Pareto solution, which is not desirable when dealing with deep multi-task learning. Therefore, some Pareto-front learning methods Lin et al. (2020); Navon et al. (2021); Ruchte & Grabocka (2021) are presented to learn a model that at inference time can generate entire Pareto front. (called Pareto-front learning problems). One of the most common ideas is to design a hypernetwork to generate model weights with different trade-offs in an end-to-end trainingLin et al. (2020); Navon et al. (2021). Ruchte et al. Ruchte & Grabocka (2021) proposed to augment the preference to the feature space to directly condition the model to avoid the extra overhead incurred in training the hyper-network (called COSMOS). However, the gradient-based BLMOP method has not been noticed.

3 Automatic Multi-Task Graph Representation Learning

3.1 Problem Statement

In general, no one solution α^* can achieve the optimum of all objectives in a MOO problem. Thus, the MOO $min_{\alpha \in \mathcal{A}} \quad \mathcal{L}(\alpha) = (\mathcal{L}_1(\alpha), \cdots, \mathcal{L}_m(\alpha))$ aims to obtain a set of Pareto optimal solutions A^*, which provide different optimal trade-offs among m objectives. The following definitions are given.

[1] The code is available at: `https://github.com/DMTGAS/DMTGAS`.

Pareto doninance A solution $\alpha \in \mathcal{A}$ is said to dominate another solution $\alpha^{'} \in \mathcal{A}$ ($\alpha \prec \alpha^{'}$) if and only if two conditions are met: (1) $\mathcal{L}_i(\alpha) \leqslant \mathcal{L}_i(\alpha^{'}), \forall i \in \{1, \ldots, m\}$, (2) $\mathcal{L}_j(\alpha) < \mathcal{L}_j(\alpha^{'}), \exists j \in \{1, \ldots, m\}$.

Pareto optimality A solution $\alpha \in \mathcal{A}$ is said to Pareto optimal if it is not dominated by any other solution $\alpha^{'} \in \mathcal{A}$. The set of all Pareto optimal solutions is called the Pareto set A^*. The Pareto front is *M*-dimensional manifold of the Pareto set in the loss space.

Exact Pareto optimal with a preference vector *r* A Pareto optimal solution $\alpha^* \in \mathcal{A}$ is said to be exact Pareto optimal with a preference vector $\boldsymbol{r} = (r_1, r_2, \ldots, r_m)$ if $r_1\mathcal{L}_1(\alpha^*) = r_2\mathcal{L}_2(\alpha^*) = \ldots = r_m\mathcal{L}_m(\alpha^*)$. It is worth noting that for any exact Pareto optimal solution, $\mathcal{L}(\alpha^*)$ is the intersection of the preference vector *r* and the Pareto front.

To address the challenge of automatically designing MTGNN models that matches user-desired task preferences, we model automatic multi-task graph representation learning as a BL-MOP. Formally, let $\mathcal{A}$ and $\mathcal{W}$ be the search space of multi-task model architectures α and weights w. In general, given a training set $\mathcal{D}_{train} := \{(x_n, y_n)\}_{n=1}^{NT}, x_n \in \mathcal{X}, y_n \in \mathcal{Y}$ and a validation set $\mathcal{D}_{valid} := \{(x_n, y_n)\}_{n=1}^{NV}, x_n \in \mathcal{X}, y_n \in \mathcal{Y}$ drawn from a data sampling distribution P_D, a BL-MOP can be formulated as follow:

$$\begin{aligned} A^* &= argmin_{\alpha \in \mathcal{A}} \mathcal{L}(W^*(\alpha), \alpha, \mathcal{D}_{valid}) = (\mathcal{L}_1(W^*(\alpha), \alpha, \mathcal{D}_{valid}), \cdots, \mathcal{L}_m(W^*(\alpha), \alpha, \mathcal{D}_{valid})) \\ s.t. \quad & W^*(\alpha) = argmin_{w \in \mathcal{W}} \mathcal{L}(w, \alpha, \mathcal{D}_{train}) = (\mathcal{L}_1(w, \alpha, \mathcal{D}_{train}), \cdots, \mathcal{L}_m(w, \alpha, \mathcal{D}_{train})) \end{aligned}, \tag{1}$$

where function $\mathcal{L}$: $\mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^m$. p, n, m are the dimensions of architecture search space, weight search space, and loss space, respectively. The upper level and lower level MOP (UL-MOP and LL-MOP) aim to find Pareto optimal architecture set A^* and Pareto optimal weight set W^*, respectively.

Because a feasible architecture requires that its weights be Pareto optimal, the difficulty of BL-MOP lies in the customization of the interaction scheme of architecture and weights during the optimization process. For each architecture, which exact Pareto optimal weight is chosen is challenging in deep learning. In practical scenarios, users may give task preferences $\boldsymbol{r}$. So the following two strategies for different scenarios are proposed to address the issue by maintaining only a single architecture and weights during the alternate optimization process, as shown in Figure 1.

***r* is known** In the automatic multi-task graph representation learning, we expect to obtain the architecture and weights with the consistent preference $\boldsymbol{r}$, so the problem (1) is transformed into finding the exact Pareto optimal architecture α^* and weights w^* with a given preference vector $\boldsymbol{r} \in [0, 1]^m, \sum_{j=1}^m r_j = 1, r_j \geq 0$. The preference-specific BL-MOP is described as

$$\begin{aligned} \alpha^* &= argmin_{\alpha \in \mathcal{A}} \mathcal{L}\left(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r}\right) \\ s.t. \quad & w^*(\alpha) = argmin_{w \in \mathcal{W}} \mathcal{L}\left(w, \alpha, \mathcal{D}_{train}, \boldsymbol{r}\right) \end{aligned}, \tag{2}$$

where $\mathcal{L}\left(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r}\right) = \left(\mathcal{L}_1\left(w^*(\alpha), \alpha, \mathcal{D}_{valid}, r_1\right), \cdots, \mathcal{L}_m\left(w^*(\alpha), \alpha, \mathcal{D}_{valid}, r_m\right)\right)$ and $\mathcal{L}\left(w, \alpha, \mathcal{D}_{train}, \boldsymbol{r}\right) = \left(\mathcal{L}_1\left(w, \alpha, \mathcal{D}_{train}, r_1\right), \cdots, \mathcal{L}_m\left(w, \alpha, \mathcal{D}_{train}, r_m\right)\right)$. In the iterative process of optimizing the problem (2), it is only necessary to maintain one architecture and weights with the consistent preference in real time.

***r* is unknown** When the task trade-offs are unknown, it is expensive to learn the architecture and weights of a separate model for each task preference $\boldsymbol{r} \in [0, 1]^m, \sum_{j=1}^m r_j = 1, r_j \geq 0$ sampled from a probability distribution $P_{\boldsymbol{r}}$. Therefore, task preference $\boldsymbol{r}$ are incorporated into the end-to-end training of architecture α^* and weights w^* of a single model $f(x, \boldsymbol{r}; w, \alpha)$ to obtain the entire Pareto front at inference time, which is formulated as a bi-level Pareto learning problem:

$$\begin{aligned} \alpha^* &= argmin_{\alpha \in \mathcal{A}} \mathbb{E}_{\boldsymbol{r} \sim P_{\boldsymbol{r}}, \mathcal{D}_{valid} \sim P_{\mathcal{D}}} \mathcal{L}\left(y, f\left(x, \boldsymbol{r}; w^*(\alpha), \alpha\right)\right) \\ s.t. \quad & w^*(\alpha) = argmin_{w \in \mathcal{W}} \mathbb{E}_{\boldsymbol{r} \sim P_r, \mathcal{D}_{train} \sim P_{\mathcal{D}}} \mathcal{L}(y, f(x, \boldsymbol{r}; w, \alpha)) \end{aligned}, \tag{3}$$

where $\mathcal{L}\left(y, f\left(x, \boldsymbol{r}; w^*(\alpha), \alpha\right)\right) = \left(\mathcal{L}_1\left(y, f\left(x, \boldsymbol{r}; w^*(\alpha), \alpha\right)\right), \cdots, \mathcal{L}_m\left(y, f\left(x, \boldsymbol{r}; w^*(\alpha), \alpha\right)\right)\right)$ and $\mathcal{L}(y, f(x, \boldsymbol{r}; w, \alpha)) = (\mathcal{L}_1(y, f(x, \boldsymbol{r}; w, \alpha)), \cdots, \mathcal{L}_m(y, f(x, \boldsymbol{r}; w, \alpha)))$. In the scheme, the inference condition of a model is transformed into $f(x, \boldsymbol{r}; w^*(\alpha), \alpha) : \mathcal{X} \times \mathbb{R}^m \times \mathcal{A} \to \mathcal{Y}$ for the UL-MOP and $f(x, \boldsymbol{r}; w, \alpha) : \mathcal{X} \times \mathbb{R}^m \times \mathcal{W} \to \mathcal{Y}$ for the LL-MOP. Practically, the input x is concatenated with the $\boldsymbol{r}$. Architecture and weights of a model are trained on this joint feature space. Once we find the optimal solution α^* and $w^*(\alpha^*)$ for the problem (3), we can approximate the entire Pareto architecture front and Pareto weight front at inference time.

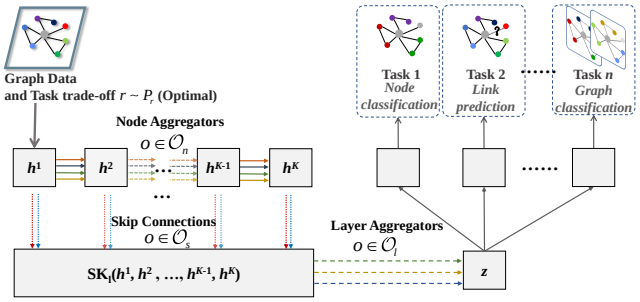


Figure 2: An illustration of a supernet of DMTGAS.

Table 1: The operations used in the search space of DMTGAS.

Name	Operations
Node Aggregators $\mathcal{O}_n$	GCN, GIN, GENIEPATH, MLP, GAT, GAT-SYM, GAT-COS, GAT-LINEAR, GAT-GEN-LINEAR, MEAN-SAGE, MAX-SAGE, SUM-SAGE
Layer Aggregators $\mathcal{O}_l$	LSTM, CONCAT, MAX, MEAN, SUM
Skip Connections $\mathcal{O}_s$	ZERO, IDENTITY

3.2 Differentiable Mult-Task GNN Architecture Search Framework

Preliminaries: GNN The message-passing framework of a typical K-layer GNN can be formulated as follows: the node representation $\mathbf{h}_i^k$ of each node i in the k-th layer is updated as:

$$\mathbf{h}_i^{(k)} = \mathrm{ACT}^k\left(\mathbf{W}^{(k)} \cdot \mathrm{AGG}^{(k)}\left(\left\{\mathbf{h}_j^{(k-1)}, \forall j \in \mathcal{N}(i)\right\}\right)\right), \tag{4}$$

where ACT^k, $\mathbf{W}^{(k)}$, and $\mathrm{AGG}^{(k)}$ are the activation function, the learnable weight matrix, and the aggregation function in the k-th layer, respectively. $\mathcal{N}(i)$ is the neighborhood of node i. Considering the residual GNN model Xu et al. (2018); Chen et al. (2020), the final node representation $\mathbf{z}_i$ of the node i is calculated by the layer aggregator function $\mathrm{AGG_l}$ and layer skip-connections $\mathrm{SK_l}$:

$$z_i = \mathrm{AGG_l}\left(\mathrm{SK_l}\left(\mathbf{h}_i^{(1)}, \cdots, \mathbf{h}_i^{(K)}\right)\right). \tag{5}$$

For the graph-level representation, the node representation information from the entire graph $G = \{V, E\}$ is aggregated by a readout function R:

$$\mathbf{h}_G = \mathrm{R}\left(\left\{\mathbf{h}_j^{(K)}, \forall j \in V\right\}\right). \tag{6}$$

Search Space The search space of DMTGAS, represented as a supernet Huan et al. (2021), is a directed acyclic graph (DAG) as shown in Figure 2. The source neural module is the input, the sink neural module is the output, and the edge (a, b) represents the operation o^{ab} on the graph data (such as a GCN layer and a GIN layer). Similar to the search space of many well-established single-task NAS Huan et al. (2021); Wei et al. (2021; 2022), we focus on three important operation components of shared layers: node aggregators $\mathcal{O}_n$, layer aggregators $\mathcal{O}_l$, and layer skip-connections $\mathcal{O}_s$ as shown in Table 1, which are based on classic model: MLP Wei et al. (2021), GIN Xu et al. (2019), GCN Kipf & Welling (2017), GAT Velickovic et al. (2017), GeniePath Liu et al. (2019), and SAGE Hamilton et al. (2017). For a three-layer shared-layer architecture (backbone), the size of the search space is $12^3 \times 2^3 \times 5 = 69120$. If the computational resources are sufficient, more operations can be expanded into the search space.

Continuous Relaxation Following the design principles of representative differentiable NAS methods, DMTGAS are employed to solve problems (2) and (3), whose search space of UL-MOPs is continuous. The discrete selection of an operation $o^{ab}\left(x^a\right)$ in the supernet is relaxed to a softmax

of all candidate operations in $\mathcal{O}$ (chosen from $\mathcal{O}_n, \mathcal{O}_l$, and $\mathcal{O}_s$) as:

$$\overline{o}^{ab}\left(x^a\right) = \sum_{o \in \mathcal{O}} \frac{\exp\left(\alpha_o^{ab}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\alpha_{o'}^{ab}\right)} o\left(x^a\right), \tag{7}$$

where α^{ab} is trainable operation mixing vectors (called architecture parameters) to control the choice of operations, and x^a is the output of node a in the supernet. In short, each operation is treated as a probability distribution over all candidate operations. After obtaining architecture parameters, the input of each node b in the supernet is calculated by aggregating all input edges as $x^b = \sum_a \overline{o}^{ab}\left(x^a\right)$. In this way, the search space of UL-MOPs is parameterized by the architecture parameters α.

Optimization Strategy The optimization strategy of DMTGAS is presented in Algorithm 1 to solve problems (2) and (3). It is difficult to directly obtain the closed solution of the LL-MOP in problems (2) and (3) due to the complicated dependency across architecture and weights. A common method Beirami et al. (2017); Ye et al. (2021); Franceschi et al. (2018) is to replace weight optimization with a dynamical system. So we consider approximate forms of these problems as:

$$\alpha^* = argmin_{\alpha \in \mathcal{A}} \quad \mathcal{L}(w_K(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r}), \tag{8}$$

$$\alpha^* = argmin_{\alpha \in \mathcal{A}} \quad \mathbb{E}_{\boldsymbol{r} \sim P_{\boldsymbol{r}}, \mathcal{D}_{valid} \sim P_{\mathcal{D}}} \mathcal{L}(y, f(x, \boldsymbol{r}; w_K(\alpha), \alpha))), \tag{9}$$

where $w_K(\alpha)$ is the iterative solution of the LL-MOP. For the LL-MOP in the formula (2), given an architecture α, a preference vector $\boldsymbol{r}$, and numbers of iterations K, a sequence $\{w_k(\alpha)\}_{k=1}^K$ can be obtained as:

$$\{w_{k+1}(\alpha) = w_k\left(\alpha\right) - \xi d_k, k = 0, \cdots, K-1\}, \tag{10}$$

where

$$d_k = \boldsymbol{\mu}^T \nabla_w \mathcal{L}(w_k(\alpha), \alpha, \mathcal{D}_{train}, \boldsymbol{r}). \tag{11}$$

η is the step size. And d_k is a valid MOO gradient direction, which is a convex combination of the vanilla gradients. Parameter $\boldsymbol{\mu} = (\mu_1, \cdots, \mu_m)$ can be calculated by the preference-based MOO method (such as LS Navon et al. (2021) (preference vector $\boldsymbol{r}$ as loss weights) and EPO Mahapatra & Rajan (2020)). For the LL-MOP in the formula (3), given an architecture α, a preference vector $\boldsymbol{r} \sim P_{\boldsymbol{r}}$, and numbers of iterations K, a sequence $\{w_k(\alpha)\}_{k=1}^K$ can be obtained as the formula (10), where

$$d_k = \nabla_w \left[\boldsymbol{r}^T \mathcal{L}(y, f(x, \boldsymbol{r}; w_k(\alpha), \alpha)) - \tau \frac{\boldsymbol{r}^T \mathcal{L}(y, f(x, \boldsymbol{r}; w_k(\alpha), \alpha))}{\|\boldsymbol{r}\| \, \|\mathcal{L}(y, f(x, \boldsymbol{r}; w_k(\alpha), \alpha))\|} \right]_{\mathcal{D}_{train} \sim P_{\mathcal{D}}}. \tag{12}$$

τ is the penalty parameter. d_k is a valid MOO gradient direction that can be obtained by the Pareto-front learning method. COSMOS Ruchte & Grabocka (2021) is adopted in the paper.

In the same way, the UL-MOP in problems (8) and (9) can also be directly solved by the same MOO method. α can be updated by:

$$\{\alpha_{t+1} = \alpha_t - \eta d_t, t = 0, \cdots, T-1\}. \tag{13}$$

For the UL-MOP in the problem (8),

$$d_t = \boldsymbol{\nu}^T \nabla_\alpha \mathcal{L}(w_K(\alpha_t), \alpha_t, \mathcal{D}_{valid}, \boldsymbol{r}), \tag{14}$$

where parameter $\boldsymbol{\nu} = (\nu_1, \cdots, \nu_m)$ can be calculated by the same preference-based MOO method. For the UL-MOP in the problem (9),

$$d_t = \nabla_w \left[\boldsymbol{r}^T \mathcal{L}(y, f(x, \boldsymbol{r}; w_K(\alpha_t), \alpha_t)) - \sigma \frac{\boldsymbol{r}^T \mathcal{L}(y, f(x, \boldsymbol{r}; w_K(\alpha_t), \alpha_t))}{\|\boldsymbol{r}\| \, \|\mathcal{L}(y, f(x, \boldsymbol{r}; w_K(\alpha_t), \alpha_t))\|} \right]_{\mathcal{D}_{valid} \sim P_{\mathcal{D}}}, \tag{15}$$

where σ is the penalty parameter. After the optimization, a operation o^{ab} of the largest weight in the supernet is selected as the final discrete architecture α, i.e., $o^{ab} = argmax_{o \in \mathcal{O}} \alpha_o^{ab}$. Finally, discrete architecture α is retrained by the same MOO method to obtain weights with the high performance.

Algorithm 1 The Optimization Strategy of DMTGAS

Require: T, K: Numbers of iterations, ξ, η: Step size, $\boldsymbol{r}$: Preference vector ($\boldsymbol{r}$ is known), τ, σ: Penalty parameter ($\boldsymbol{r}$ is unknown).
Ensure: α, w: The obtained architecture and weights.

```
Randomly initialize α_0;
for each t ∈ [0, T − 1] do
    if r is unknown then
        Sample a preference vector r ∼ P_r;
    end if
    Randomly initialize w_0^t (α_t);
    for each k ∈ [0, K − 1] do
        Update w_{k+1}^t (α_t) by the formula (10);
    end for
    Update α_{t+1} by the formula (13);
end for
Derive the final discrete architecture α based on α_T;
Retrain on the architecture α by the same MOO method to obtain its weights w.
```

3.3 Theoretical Analysis

In this section, the convergence of Algorithm 1 for the problem (2) is presented (The proof for the problem (3) is similar). The following assumptions are given first.

Assumption 3.1. *It is assumed that: 1) $\mathcal{A}$ is a compact set; 2) $argmin_{w \in \mathcal{W}} \quad \mathcal{L}(w, \alpha, \mathcal{D}_{train}, \boldsymbol{r})$ is a singleton for every $\alpha \in \mathcal{A}$; 3) $\{w_k(\alpha)\}_{k=1}^K$ is uniformly bounded on $\mathcal{A}$ 4) $\mathcal{L}(w(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})$ is jointly continuous.*

Assumption 3.2. *It is assumed that: 1) $\mathcal{L}(*, \alpha, \mathcal{D}_{valid}, \boldsymbol{r})$ is uniformly Lipschitz continuous; 2) $\{w_k(\alpha)\}_{k=1}^K$ converges uniformly to $w^*(\alpha)$ on $\mathcal{A}$ as $K \to +\infty$.*

Assumption 3.1 *2)* shows that for each architecture α^*, the problem (2) (or (3)) only admits a unique optimal weight w^*. In addition, **Assumption (3.1)** and **Assumption (3.2)** are natural and widely used to analyzed bi-level single-objective optimization problems (BLSOPs) Franceschi et al. (2018); Grazzi et al. (2020); Ye et al. (2021). Under these assumptions, we have:

Theorem 3.1. *Assuming **Assumption (3.1)** is satisfied, then $\mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r}) \to \mathcal{L}(w^*(\overline{\alpha}), \overline{\alpha}, \mathcal{D}_{valid}, \boldsymbol{r})$ for any convergent sequence $\alpha \to \overline{\alpha}$.*

Proof. See the Appendix 2. Theorem 3.1 shows the existence of solutions and the convergence of the LL-MOP. It is similar to that of the BL-SOPs Franceschi et al. (2018); Grazzi et al. (2020); Ye et al. (2021). □

Theorem 3.2. *Assuming **Assumption (3.1)** and **Assumption (3.2)** are satisfied, then $\mathcal{L}(w_K(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})$ continuously converges to $\mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})$.*

Proof. See the Appendix 2. Theorem 3.2 shows the convergence of Algorithm 1. □

4 Experiments

Experimental Setting We evaluate DMTGAS with LS, EPO, and COSMOS (called DMTGAS-LS, DMTGAS-EPO, and DMTGAS-COSMOS) on three real-world graph datasets (ENZYMES, PROTEINS, and DHFR Morris et al. (2020)) that allow multi-task settings (graph classification (GC), node classification (NC), and link prediction (LP)). We consider the baselines: 1) Single Task: a single task baseline for the handcrafted model. It trains each task using the different backbone (GCN, SAGE, GAT, and GIN) and a task-specific layer; 2) Random Search: a simple baseline in NAS. It randomly samples architectures from the discrete search space; 3) ST-SANE: an advanced single task baseline in Graph NAS; 4) Handcrafted methods with MGDA, LS, EPO, and COSMOS: the state-of-the-art handcrafted MTGNN Buffelli & Vandin (2022) with different MOO strategies. We use the same architecture of task-specific layers for all multi-task baselines to ensure a fair

Table 2: Results compared to all baselines. For LS and EPO, a uniform preference vector $\boldsymbol{r} = (0.333, 0.333, 0.333)$ is given. For COSMOS, the model is inferred 25 times with 25 equally distributed $\boldsymbol{r}$ to obtain the optimal performance on each task at inference time.

Methods	ENZYMES (Mean(Std))			PROTEINS (Mean(Std))			DHFR (Mean(Std))		
	GC	NC	LP	GC	NC	LP	GC	NC	LP
ST-GCN	51.6 (0.047)	87.5 (0.009)	75.5 (0.020)	73.3 (0.034)	72.3 (0.008)	85.6 (0.007)	71.5 (0.019)	97.3 (0.003)	98.8 (0.005)
ST-SAGE	53.3 (0.042)	93.6 (0.008)	84.7 (0.030)	67.7 (0.022)	90.9 (0.008)	89.1 (0.002)	64.5 (0.034)	95.3 (0.002)	99.4 (0.001)
ST-GAT	53.7 (0.010)	89.5 (0.005)	76.5 (0.012)	65.9 (0.019)	75.7 (0.014)	77.0 (0.014)	67.8 (0.026)	43.9 (0.028)	95.9 (0.005)
ST-GIN	45.8 (0.031)	86.3 (0.015)	77.2 (0.025)	69.5 (0.019)	80.2 (0.010)	82.9(0.010)	67.8 (0.031)	95.8 (0.003)	97.7 (0.007)
Random Search	53.3 (0.016)	89.0 (0.122)	79.8 (0.026)	70.0 (0.064)	82.7 (0.095)	86.8 (0.042)	72.4 (0.023)	85.8 (0.147)	98.6 (0.092)
ST-SANE	55.7 (0.018)	90.1 (0.010)	89.4 (0.002)	81.5 (0.016)	82.9 (0.002)	92.3 (0.010)	75.0 (0.030)	97.4 (0.053)	98.3 (0.015)
Handcrafted-MGDA	51.7 (0.035)	88.6 (0.008)	70.9 (0.019)	68.0 (0.016)	82.3 (0.006)	75.5 (0.018)	69.7 (0.030)	85.7 (0.003)	97.1 (0.006)
Handcrafted-LS	41.3 (0.542)	**88.8** (0.906)	76.9 (0.833)	69.6 (0.030)	**79.9** (0.022)	78.5 (0.097)	62.4 (0.038)	**84.6** (0.008)	50.0 (0.000)
DMTGAS-LS	**55.3** (0.021)	85.4 (0.016)	**84.0** (0.008)	**83.0** (0.009)	76.5 (0.011)	**81.9** (0.066)	**76.9** (0.044)	79.7 (0.014)	**95.6** (0.010)
Handcrafted-EPO	56.0 (0.036)	88.1 (0.009)	86.3 (0.020)	71.0 (0.016)	73.5 (0.007)	88.1 (0.019)	72.8 (0.030)	85.9 (0.004)	97.7 (0.007)
DMTGAS-EPO (Ours)	**56.7** (0.049)	**89.1** (0.012)	**95.8** (0.022)	**87.0** (0.004)	**82.7** (0.005)	**96.8** (0.043)	**83.3** (0.023)	**86.2** (0.004)	**99.9** (0.006)
Handcrafted-COSMOS	50.4 (0.132)	88.9 (0.005)	81.0 (0.152)	78.0 (0.029)	79.5 (0.043)	67.1 (0.010)	70.3 (0.023)	81.4 (0.017)	94.9 (0.014)
DMTGAS-COSMOS (Ours)	**52.7** (0.040)	**89.2** (0.006)	**94.3** (0.006)	**88.9** (0.024)	**81.7** (0.013)	**96.2** (0.011)	**79.2** (0.027)	**94.5** (0.014)	**98.7** (0.009)

Table 3: The HV values obtained by Handcrafted-COSMOS and DMTGAS-COSMOS.

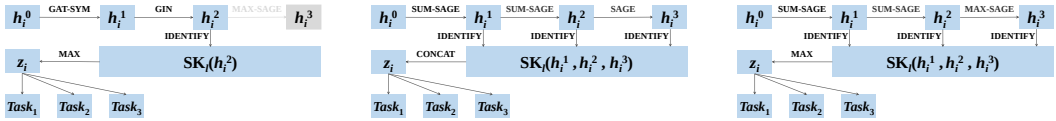
Methods	ENZYMES		PROTEINS		DHFR	
	Test	Valid	Test	Valid	Test	Valid
Handcrafted-COSMOS	0.151	0.187	0.415	0.398	0.369	0.354
DMTGAS-COSMOS	**0.188**	**0.245**	**0.472**	**0.468**	**0.475**	**0.493**

comparison. Complete experimental settings and more experimental results are presented in the Appendix 3 and Appendix 4.

Comparison with Baselines We demonstrate the overall multi-task performance of all the comparison methods in terms of accuracy (%) for NC and GC and AUC (%) for LP over five independent runs as shown in Table 2. The following five observations are presented: 1) DMTGAS-EPO and DMTGAS-COSMOS obtain higher values than the Handcrafted-EPO and Handcrafted-COSMOS, respectively, which illustrates our proposed DMTGAS can obtain a MTGNN model with stronger graph representation ability. 2) Since tasks conflict with each other, none of the handcrafted MTGNN methods with different optimization strategies achieves vast superiority compared to the single-task baseline. However, the accuracy of models obtained by DMTGAS-EPO and DMTGAS-COSMOS exceeds that of the single-task baseline in most cases, thanks to the real-time trade-off among tasks during the alternate optimization process and knowledge sharing. Furthermore, it is worth noting that DMTGAS solves all tasks at once, whereas the single-task baseline is individually tailored for each model, requiring more computational cost (see the Appendix 4 for a more detailed explanation). 3) Compared to LS (simply weighting for all task losses), EPO and COSMOS (both handcrafted methods and DMTGAS) achieve better performance on all tasks by finding the search direction in which all task losses decrease, indicating that our proposal can better handle conflict among tasks. 4) Since LS cannot guarantee to find a Pareto optimal solution with a given $\boldsymbol{r}$ Ruchte & Grabocka (2021), in the iterative optimization of DMTGAS-LS, it isn't easy to maintain the architecture and parameters with consistent preferences. This phenomenon makes DMTGAS-LS unable to outperform Handcrafted-LS comprehensively. 5) Our method outperforms random search on all tasks, which indicates that differentiable search methods in the continuous search space are more likely to find excellent solutions.

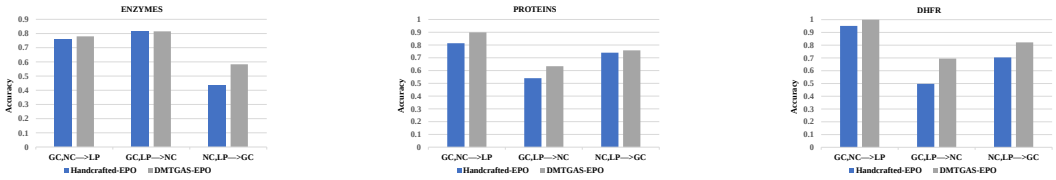
In addition, Table 3 reports the **Hypervolume** (HV) values Tian et al. (2017) obtained by Handcrafted-COSMOS and DMTGAS-COSMOS to solve GC, NC, and LP over 5 independent runs. The HV is calculated with reference point (2.5, 2.5, 2.5) and 25 equally distributed $\boldsymbol{r}$. Experimental results show that DMTGAS-COSMOS outperforms Handcrafted-COSMOS in terms of HV, i.e. our proposal can generate a higher quality Pareto front. The convergence curves of HV on all datasets are further reported and analysed in the Appendix 4.

Searched Architectures We visualize the architectures searched by DMTGAS-EPO and DMTGAS-COSMOS on different datasets as shown in Figure 3 and the Appendix 4. It can be seen that these architectures are data-dependent, whose node aggregation, skip connections, and layer aggregation are significantly different. This phenomenon further shows the importance of automatically searching for data-dependent model architectures.



(a) ENZYMES (b) PROTEINS (c) DHFR

Figure 3: Visualization of architectures searched by DMTGAS-COSMOS on all datasets, where skip connection is removed.



(a) ENZYMES (b) PROTEINS (c) DHFR

Figure 4: Results of training a task-specific model on unseen tasks using node embeddings obtained by DMTGAS-EPO and Handcrafted-EPO, where "$a, b \rightarrow c$" means that a and b are the tasks for training the multi-task model, and c is the unseen task. The preference $\boldsymbol{r}$ is set to $(0.5, 0.5)$.

Transferability We employ the node embeddings obtained by DMTGAS-EPO and Handcrafted-EPO on any two tasks to train a task-specific network on unseen tasks to demonstrate the transferability of our method. Figure 4 shows the obtained accuracy on all datasets. DMTGAS-EPO achieves the highest accuracy on all unseen tasks, which shows that our proposal can get higher quality node embedding information than handcrafted methods.

Ablation Study DMTGAS need to maintain preference-consistency of architecture and weights. In this section, we construct a baseline with inconsistent trade-offs during optimization process, called DMTGASIC. In each iteration, for EPO, a fixed preference $\boldsymbol{r} = (0.333, 0.333, 0.333)$ is provided to the outer optimization, and a randomly sampled preference vectors $\boldsymbol{r} \sim P_{\boldsymbol{r}}$ is provided to the inner optimization. For COSMOS, two randomly sampled preference vectors $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ from the distribution $P_{\boldsymbol{r}}$ are provided to inner and outer optimizations, respectively. The experimental results on the ENZYMES dataset are shown in Table 4. It can be seen that DMTGAS outperforms DMTGASIC on all tasks, which shows the importance of maintaining preference-consistency of UL-MOP and LL-MOP. We also observed this phenomenon on other datasets (see the Appendix 4 in detail).

Table 4: The experimental results obtained by DMTGASIC and DMTGAS on the ENZYMES dataset in terms of accuracy (%) for NC and GC and AUC (%) for LP.

METHODS	ENZYMES(MEAN(STD))		
	GC	NC	LP
DMTGASIC-EPO	39.3 (0.017)	85.4 (0.015)	93.3 (0.006)
DMTGAS-EPO	**56.7** (0.049)	**89.1** (0.012)	**95.8** (0.022)
DMTGASIC-COSMOS	30.0 (0.035)	80.6 (0.012)	93.2 (0.015)
DMTGAS-COSMOS	**52.7** (0.040)	**89.2** (0.006)	**94.3** (0.006)

5 CONCLUSION

We firstly introduce a differentiable multi-task GNN architecture search framework based on BL-MOP for automatic multi-task graph representation learning, called DMTGAS. DMTGAS can provide a multi-task architecture and its weights, which match user-defined task trade-offs. Experimental results on real-world datasets confirm that our proposal can generate novel multi-task GNN models that outperform current hand-designed models. In the future, we will further analyze relationships across downstream tasks and apply DMTGAS to more graph learning problems, such as self-supervised graph learningJin et al. (2022); Ju et al. (2022). A non-asymptotic convergence rate of DMTGAS is also worth exploring, which is related to the convergence rate of the used preference-based MOO methods and Pareto-front learning methods.

REFERENCES

Pedro Avelar, Henrique Lemos, Marcelo Prates, and Luis Lamb. Multitask learning on graph neural networks: Learning multiple graph centrality measures with a unified network. In *International Conference on Artificial Neural Networks*, pp. 701–715. Springer, 2019.

Ahmad Beirami, Meisam Razaviyayn, Shahin Shahrampour, and Vahid Tarokh. On optimal generalizability in parametric learning. *Advances in Neural Information Processing Systems*, 30, 2017.

Davide Buffelli and Fabio Vandin. Graph representation learning for multi-task settings: a meta-learning approach. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2022. doi: 10.1109/IJCNN55064.2022.9892010.

Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

Xinye Cai, Qi Sun, Zhenhua Li, Yushun Xiao, Yi Mei, Qingfu Zhang, and Xiaoping Li. Cooperative coevolution with knowledge-based dynamic variable decomposition for bilevel multi-objective optimization. *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2022. doi: 10.1109/TEVC.2022.3154057.

Ines Chami, Sami Abu-El-Haija, Bryan Perozzi, Christopher Ré, and Kevin Murphy. Machine learning on graphs: A model and comprehensive taxonomy. *Journal of Machine Learning Research*, 23(89):1–64, 2022. URL `http://jmlr.org/papers/v23/20-852.html`.

Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pp. 1725–1735. PMLR, 2020.

Kalyanmoy Deb and Ankur Sinha. An efficient and accurate solution methodology for bilevel multi-objective programming problems using a hybrid evolutionary-local-search algorithm. *Evolutionary Computation*, 18(3):403–449, 2010. doi: 10.1162/EVCO_a_00015.

Kalyanmoy Deb, Zhichao Lu, Ian Kropp, J. Sebastian Hernandez-Suarez, Rayan Hussein, Steven Miller, and A. Pouyan Nejadhashemi. Minimizing expected deviation in upper-level outcomes due to lower-level decision-making in hierarchical multi-objective problems. *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2022. doi: 10.1109/TEVC.2022.3172302.

Gabriele Eichfelder. Multiobjective bilevel optimization. *Mathematical Programming*, 123(2):419–449, 2010.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimiliano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. In *International Conference on Machine Learning*, pp. 1568–1577. PMLR, 2018.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In Christian Bessiere (ed.), *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pp. 1403–1409. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/195. URL `https://doi.org/10.24963/ijcai.2020/195`. Main track.

Riccardo Grazzi, Luca Franceschi, Massimiliano Pontil, and Saverio Salzo. On the iteration complexity of hypergradient computation. In *International Conference on Machine Learning*, pp. 3748–3758. PMLR, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Chester Holtz, Onur Atan, Ryan Carey, and Tushit Jain. Multi-task learning on graphs with node and graph level labels. In *NeurIPS Workshop on Graph Representation Learning*, 2019.

ZHAO Huan, YAO Quanming, and TU Weiwei. Search to aggregate neighborhood for graph neural network. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 552–563. IEEE, 2021.

Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. Automated self-supervised learning for graphs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=rFbR4Fv-D6-.

Mingxuan Ju, Tong Zhao, Qianlong Wen, Wenhao Yu, Neil Shah, Yanfang Ye, and Chuxu Zhang. Multi-task self-supervised graph neural networks enable stronger task generalization, 2022. URL https://arxiv.org/abs/2210.02016.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/pdf?id=SJU4ayYgl.

Diya Li and Heng Ji. Syntax-aware multi-task graph convolutional networks for biomedical relation extraction. In *Proceedings of the Tenth International Workshop on Health Text Mining and Information Analysis (LOUHI 2019)*, pp. 28–33, 2019.

Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qing-Fu Zhang, and Sam Kwong. Pareto multi-task learning. *Advances in neural information processing systems*, 32, 2019.

Xi Lin, Zhiyuan Yang, Qingfu Zhang, and Sam Kwong. Controllable pareto multi-task learning, 2020. URL https://arxiv.org/abs/2010.06313.

Ziqi Liu, Chaochao Chen, Longfei Li, Jun Zhou, Xiaolong Li, Le Song, and Yuan Qi. Geniepath: Graph neural networks with adaptive receptive paths. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4424–4431, Jul. 2019. doi: 10.1609/aaai.v33i01.33014424. URL https://ojs.aaai.org/index.php/AAAI/article/view/4354.

Debabrata Mahapatra and Vaibhav Rajan. Multi-task learning with user preferences: Gradient descent with controlled ascent in pareto optimization. In *International Conference on Machine Learning*, pp. 6597–6607. PMLR, 2020.

Floriane Montanari, Lara Kuhnke, Antonius Ter Laak, and Djork-Arné Clevert. Modeling physico-chemical admet endpoints with multitask graph convolutional networks. *Molecules*, 25(1):44, 2019.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. URL www.graphlearning.io.

Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=NjF772F4ZZR.

Yijian Qin, Xin Wang, Zeyang Zhang, and Wenwu Zhu. Graph differentiable architecture search with structure learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Michael Ruchte and Josif Grabocka. Scalable pareto front approximation for deep multi-objective learning. In *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 1306–1311. IEEE, 2021.

Sauli Ruuska and Kaisa Miettinen. Constructing evolutionary algorithms for bilevel multiobjective optimization. In *2012 IEEE congress on evolutionary computation*, pp. 1–7. IEEE, 2012.

Rihab Said, Slim Bechikh, Ali Louati, Abdulaziz Aldaej, and Lamjed Ben Said. Solving combinatorial multi-objective bi-level optimization problems using multiple populations and migration schemes. *IEEE Access*, 8:141674–141695, 2020. doi: 10.1109/ACCESS.2020.3013568.

Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018.

Xinping Shi and Hong Sheng Xia. Model and interactive algorithm of bi-level multi-objective decision-making with multiple interconnected decision makers. *Journal of Multi-Criteria Decision Analysis*, 10(1):27–34, 2001.

Ankur Sinha. Bilevel multi-objective optimization problem solving using progressively interactive emo. In *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 269–284. Springer, 2011.

Ankur Sinha, Pekka Malo, and Kalyanmoy Deb. A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295, 2018. doi: 10.1109/TEVC.2017.2712906.

Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33: 8728–8740, 2020.

Ye Tian, Ran Cheng, Xingyi Zhang, and Yaochu Jin. Platemo: A matlab platform for evolutionary multi-objective optimization [educational forum]. *IEEE Computational Intelligence Magazine*, 12(4):73–87, 2017. doi: 10.1109/MCI.2017.2742868.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

Shanfeng Wang, Qixiang Wang, and Maoguo Gong. Multi-task learning based network embedding. *Frontiers in Neuroscience*, pp. 1387, 2020.

Weizhong Wang, Hai-Lin Liu, and Hongjian Shi. A multi-objective bilevel optimisation evolutionary algorithm with dual populations lower-level search. *Connection Science*, 34(1):1556–1581, 2022. doi: 10.1080/09540091.2022.2077312.

Xin Wang and Wenwu Zhu. Automated machine learning on graph. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 4082–4083, 2021.

Zhili Wang, Shimin Di, and Lei Chen. Autogel: An automated graph neural network with explicit link information. *Advances in Neural Information Processing Systems*, 34, 2021.

Lanning Wei, Huan Zhao, Quanming Yao, and Zhiqiang He. Pooling architecture search for graph classification. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 2091–2100, 2021.

Lanning Wei, Huan Zhao, and Zhiqiang He. Designing the topology of graph neural networks: A novel feature fusion perspective. In *Proceedings of the ACM Web Conference 2022*, pp. 1381–1391, 2022.

Bichen Wu, Chaojian Li, Hang Zhang, Xiaoliang Dai, Peizhao Zhang, Matthew Yu, Jialiang Wang, Yingyan Lin, and Peter Vajda. Fbnetv5: Neural architecture search for multiple tasks in one run. *arXiv preprint arXiv:2111.10007*, 2021.

Yu Xie, Maoguo Gong, Yuan Gao, AK Qin, and Xiaolong Fan. A multi-task representation learning architecture for enhanced graph classification. *Frontiers in Neuroscience*, pp. 1395, 2020a.

Yu Xie, Peixuan Jin, Maoguo Gong, Chen Zhang, and Bin Yu. Multi-task network representation learning. *Frontiers in Neuroscience*, pp. 1, 2020b.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *International Conference on Machine Learning*, pp. 5453–5462. PMLR, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=ryGs6iA5Km`.

Feiyang Ye, Baijiong Lin, Zhixiong Yue, Pengxin Guo, Qiao Xiao, and Yu Zhang. Multi-objective meta learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, 1999. doi: 10.1109/4235.797969.

A APPENDIX

A.1 DETAILED RELATED WORK

Multi-Task Graph Representation Learning Most works on multi-task graph representation learning have focused on learning an embedding model for several applications on graphs. Avelar et al. Avelar et al. (2019) proposed a unified multi-task graph neural network that learns multiple graph centrality measures. Holtz et al. Holtz et al. (2019) developed a supervised learning technique for two-level labels using a multi-task architecture and validated the performance of their framework on PROTEIN and ENZYME datasets. Li et al. Li & Ji (2019) proposed a syntax-aware multi-task graph convolutional network for biomedical relation extraction, which simultaneously performs relation recognition and classification tasks, achieving state-of-the-art performance on the drug-drug interaction extraction task. Montanari et al. Montanari et al. (2019) employed a multi-task graph convolutional network to model physical-chemical ADMET endpoints. Wang et al. Wang et al. (2020) presented a multi-task network embedding method, where one of the tasks aims to maintain high-order proximity between paired nodes throughout the network. Another task is to maintain low-order proximity in the one-hop region of each node. Xie et al. Xie et al. (2020a) developed a multi-task representation learning architecture for enhanced graph classification, which leverages the knowledge of the node classification task to better enhance the performance of the graph classification task. They further proposed multi-task network representation learning for handling node classification and link prediction tasks Xie et al. (2020b). Buffell et al. Buffelli & Vandin (2022) designed a meta-learning method for multi-task graph representation learning. However, these application-specific handcrafted methods rely heavily on expert knowledge and ignore task conflicts due to excessive parameter sharing between tasks.

Automatic Graph Representation Learning This section mainly focuses on the neural architecture search (NAS) for automatic graph representation learning. The challenges of NAS generally include the design of search space and search strategy. According to the construction principles of graph neural modules, the search space can be divided into four categories: micro search space, macro search space, pooling methods, and hyperparameters. Search strategies include reinforcement learning, differentiable methods, evolutionary algorithms, and hybrid methods Wang & Zhu (2021). Differentiable methods for GNN have gained popularity in recent years, which builds a single supernet that contains all operations Huan et al. (2021); Wang et al. (2021); Qin et al. (2021). Each candidate operation is treated as a probability distribution over all possible operations. The objective function of architecture search is differentiable through continuous relaxation of the search space. Thus, gradient-based methods can jointly optimize the architecture and weights. Existing works mainly focus on the search space and search strategy to learn a graph representation model customized for a given task Wang & Zhu (2021). However, automatic learning methods for multiple tasks on graphs have not been noticed.

Bi-level Multi-objective Optimization (BL-MOP) Due to the complex interaction mechanism between the upper level (UL) and lower level (LL) and the high computational cost, there are limited works on the BL-MOPsSinha et al. (2018). Existing efforts are divided into three categories: classicalShi & Xia (2001); Eichfelder (2010), evolution-basedDeb & Sinha (2010); Sinha (2011); Ruuska & Miettinen (2012); Said et al. (2020); Cai et al. (2022); Wang et al. (2022); Deb et al. (2022) and gradient-based methodsYe et al. (2021). Classical methods require some strong mathematical assumptions, which make them incapable of being directly applicable to many practical scenarios. Despite good performance on traditional NP-hard problems, evolution-based methods are not directly applicable to high-dimensional BL-MOPs in deep learning due to the expensive training cost. In addition, the high computational cost of maintaining two solution sets is unacceptable in practical resource-constrained environments. The gradient-based method directly approximates the LL-MOP as a single objective problem to guarantee that each architecture corresponds to an optimal weight. But this method may lead to inconsistencies in task trade-offs at the upper and lower levels. Furthermore, the method's performance is severely affected by the manually tuned task loss weights. However, our proposed schemes guarantee the consistency of task preference at the upper and lower levels and do not need to maintain multiple weights for each candidate architecture in the optimization process, which cleverly avoids the above optimization difficulties.

A.2 PROOFS OF THEOREMS IN SECTION 3.3

Proof of Theorem 3.1 Let $\overline{\alpha} \in \mathcal{A}$ and $\{\alpha^t\}_{t=1}^T$ is a sequence in $\mathcal{A}$ satisfying $\alpha^t \to \overline{\alpha}$, we should prove that $\mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r}) \to \mathcal{L}(w^*(\overline{\alpha}), \overline{\alpha}, \mathcal{D}_{valid}, \boldsymbol{r})$. Because $\{w^*(\alpha)\}$ is uniformly bounded on $\mathcal{A}$ (**Assumption (3.1) *3)***), for some $\overline{w}$, there is a sub-sequence $\{w^*(\alpha_k^t)\} \to \overline{w}$. As $\alpha_k^t \to \overline{\alpha}$, $w^*(\overline{\alpha}) = argmin_{w \in \mathcal{W}} \mathcal{L}(w, \overline{\alpha}, \mathcal{D}_{train}, \boldsymbol{r})$ (**Assumption (3.1) *2)***). Thus, $w^*(\overline{\alpha}) = \overline{w}$. That is, $\{w^*(\alpha_k^t)\}$ is convergence to a unique cluster point $w^*(\overline{\alpha})$. Therefore, as $\alpha^t \to \overline{\alpha}$, $w^*(\alpha^t) \to w^*(\overline{\alpha})$. Since $\mathcal{L}(w(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})$ is jointly continuous (**Assumption (3.1) *4)***), $\mathcal{L}(w^*(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) \to \mathcal{L}(w^*(\overline{\alpha}), \overline{\alpha}, \mathcal{D}_{valid}, \boldsymbol{r})$ as $\alpha^t \to \overline{\alpha}$. In addition, because $\mathcal{A}$ is a compact set (**Assumption (3.1) *1)***), the existence of the solution is proved according to the Weierstrass theorem.

Proof of Theorem 3.2 Let $\{\alpha^t\}_{t=1}^T$ is a sequence in $\mathcal{A}$ satisfying $\alpha^t \to \alpha$. For every sequence $\{\alpha^t\}_{t=1}^T$,

$$\begin{aligned} &\|\mathcal{L}(w_K(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) - \mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})\| \\ \leq &\|\mathcal{L}(w_K(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) - \mathcal{L}(w^*(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r})\|. \\ &+ \|\mathcal{L}(w^*(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) - \mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})\| \end{aligned} \tag{16}$$

According to Theorem 3.1, as $\alpha^t \to \alpha$,

$$\left\|\mathcal{L}(w^*(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) - \mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})\right\| \to 0. \tag{17}$$

Therefore,

$$\begin{aligned} &\|\mathcal{L}(w_K(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) - \mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})\| \\ \leq &\|\mathcal{L}(w_K(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) - \mathcal{L}(w^*(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r})\|. \end{aligned} \tag{18}$$

In addition, since $\mathcal{L}(*, \alpha, \mathcal{D}_{valid}, \boldsymbol{r})$ is uniformly Lipschitz continuous (**Assumption (3.2) *1)***), we have

$$\begin{aligned} &\|\mathcal{L}(w_K(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) - \mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})\| \\ \leq &\|\mathcal{L}(w_K(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r}) - \mathcal{L}(w^*(\alpha^t), \alpha^t, \mathcal{D}_{valid}, \boldsymbol{r})\| \\ \leq &\beta \left\|w_K(\alpha^t) - w^*(\alpha^t)\right\|. \end{aligned} \tag{19}$$

Because $\{w_k(\alpha)\}_{k=1}^K$ converges uniformly to $w^*(\alpha)$ on $\mathcal{A}$ as $K \to +\infty$ (**Assumption (3.2) *2)***), $\mathcal{L}(w_K(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})$ continuously converges to $\mathcal{L}(w^*(\alpha), \alpha, \mathcal{D}_{valid}, \boldsymbol{r})$.

A.3 MORE EXPERIMENTAL DETAILS

In this section, we provide more experimental details and results in our experimental section. Our experiments are running with PyTorch (1.10.1) and PyTorch Geometric (2.0.3) on a GPU 2080Ti (Memory: 12GB, Cuda version:11.3). We implement our proposed DMTGAS [2] based on the source code provided by SANE [3]. We independently run all automated search methods for 5 times with different random seeds to obtain the best architecture on the validation dataset. Then the best architecture is retrained over 5 independent runs. The final performance on the test set is reported. In each run, we split the nodes in all graphs into 70%, 10%, 20% for training, validation, and test.

Datasets We employ three popular read-world datasets (ENZYMES, PROTEINS, and DHFR) with graph labels, node attributes, and node labels from the TUDataset library Morris et al. (2020). These datasets satisfy multi-task conditions (node classification, graph classification, and link prediction) execution. For the node classification, only a small fraction of nodes are labeled. For the link prediction, in each graph, some randomly removed edges and unremoved edges are treated as positive examples. Couples of non-adjacent nodes are randomly sampled as negative examples. More implementation details can be found in Buffelli & Vandin (2022).

Task-Specific Layer We employ the same task-specific layers as state-of-the-art hand-designed multi-task graph models Buffelli & Vandin (2022). The task-specific layer is composed of three heads. For node classification head, a single-layer neural network with a Softmax activation is employed, which maps embeddings to class predictions. For graph classification head, two-layer neural network is used to map embeddings to class predictions. The first one is a linear transformation with

[2] https://github.com/DMTGAS/DMTGAS.

[3] https://github.com/AutoML-4Paradigm/SANE.

a ReLU activation. And then its output is averaged. The second one is a simple neural network layer with a Softmax activation. For link prediction head, two-layer neural network is used to map embeddings to the probability of a link between nodes. The first one is a linear transformation with a ReLU activation. The second one is a single-layer neural network whose input is the concatenation of two node embeddings and whose output is the probability of the link between them.

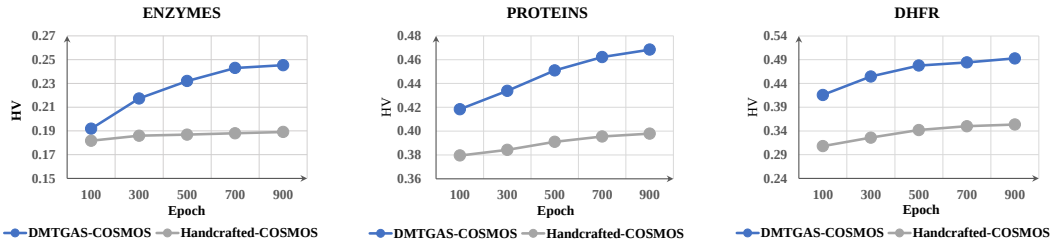
Baseline To verify the efficacy of DMTGAS, we provide four types of baselines for experiments with respect to two levels - MTL and Graph NAS: single-task handcrafted baselines (single-level single-objective optimization), single-task Graph NAS baselines (bi-level single-objective optimization), multi-task handcrafted baselines (single-level multi-objective optimization), and multi-task Graph NAS baselines (bi-level multi-objective optimization).

1) **Single-task Handcrafted Baselines**. Single task handcrafted baselines (ST-GCN, ST-SAGE, ST-GAT, and ST-GIN) train each task independently by the advanced backbone (GCN, SAGE, GAT, and GIN) and a task-specific layer.
2) **Single-task Graph NAS Baselines**. An advanced Graph NAS method (ST-SANE) with a task-specific layer is employed as a single task Graph NAS baseline.
3) **Multi-task Handcrafted Baselines**. We adopt the state-of-the-art handcrafted MTGNN architecture Buffelli & Vandin (2022), and optimize it with four MOO strategies, including linear scalarization (LS), multiple-gradient descent algorithm (MGDA), exact Pareto optimal search (EPO), and conditioned one-shot multi-objective search (COSMOS). These multi-task handcrafted baselines are called Handcrafted LS, Handcrafted MGDA, Handcrafted EPO, and Handcrafted COSMOS. The handcrafted architecture is an encoder-decoder model. The encoder is the same as the single task GCN baseline, and the decoder is as described in the previous section.
4) **Multi-task Graph NAS Baselines**. Random search is a simple baseline in Graph NAS, which randomly samples architectures from the discrete search space. And LS is used as the optimization strategy for each sampled architecture. In addition, LS is also embedded in DMTGAS to obtain DMTGAS-LS as a multi-task graph NAS baseline.

Implementation Details In this part, we give more implementation details of all compared methods.

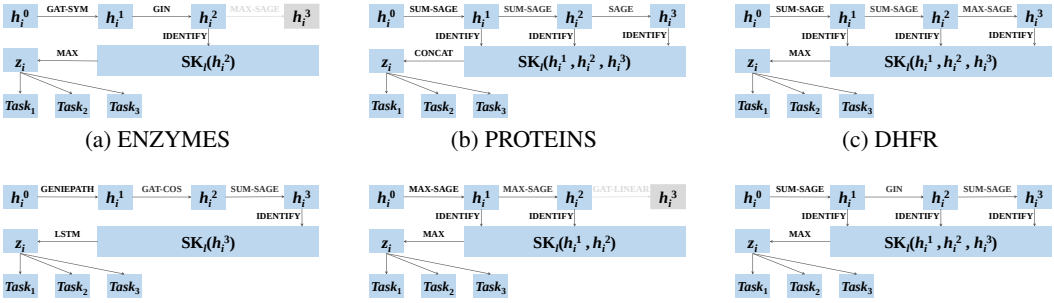
- For all baselines, we set the batch size to 256 and train all models for 1000 epochs using Adam with a learning rate of 0.001.
- For the single task baseline and human-designed MTGNN, node embeddings are normalized to the unit norm between GCN layers for better performance.
- For the random search, the number of samples is set to 200. And we employ grid search to tune the hyper-parameters.
- For methods with EPO and LS, we set a uniform preference vector $\boldsymbol{r} = (0.33, 0.33, 0.33)$.
- For methods with COSMOS, the preference vector $\boldsymbol{r}$ is sampled from a Dirichlet distribution controlled with α. We use grid search to tune the penalty parameter τ (σ) and the Dirichlet sampling parameter α with the search space $[0.001, 0.01, 2, 3, 8]$ and $[0.1, 0.5, 1, 5, 10]$, respectively. We run the model over 25 uniformly distributed preference vectors to obtain a Pareto front in the inference phase.
- For the DMTGAS, both the architecture optimizer and the weight optimizer are Adam with a learning rate of 0.001. In this search phase, we set the hidden layer to 64 for the sake of computational resources.

Metric Hypervolume (HV) is a metric commonly used as a performance indicator of multi-objective algorithms and is calculated as the size of the dominated spaceZitzler & Thiele (1999); Tian et al. (2017). It represents the volume of the hypercube enclosed by the individuals in the solution set and the reference points in the objective space. Let $S = \{q_1, q_2, ..., q_n\}$ be the non-dominated set, and $Z = \{z_1, z_2, ..., z_n\}$ as the reference point, the HV is expressed as:



(a) ENZYMES (b) PROTEINS (c) DHFR

Figure 5: Convergence curves of HV obtained by Handcrafted-COSMOS and DMTGAS-COSMOS on the valid dataset.



(a) ENZYMES (b) PROTEINS (c) DHFR

(d) ENZYMES (e) PROTEINS (f) DHFR

Figure 6: Visualization of architectures searched by DMTGAS-COSMOS ((a)-(c)) and DMTGAS-EPO ((d)-(f)) on all datasets, where skip connection is removed.

$$HV(S) = VOL\left(\bigcup_{\substack{q_i \in S \\ q_i \preceq Z}} \prod_{i=1}^{n} [q_i, z_i]\right), \tag{20}$$

where $\prod_{i=1}^{n} [q_i, z_i]$ denotes the region of the non-dominated points bounded by Z, and VOL is the Euclidean volume. The accuracy of the HV metric depends on the choice of reference points, and different choices of reference points will yield different results when evaluating the same solution set. In general, a solution with a larger HV is considered to be of better quality.

A.4 More Experimental Results

Figure 5 reports convergence curves of HV values obtained by Handcrafted-COSMOS and DMTGAS-COSMOS on all datasets over five independent runs, respectively. As can be seen from Figure 5, our searched architectures converge more easily than manually designed ones, which further illustrates the advantages of automatically designing architectures.

Figure 6 shows the visualization of the architectures searched by DMTGAS-EPO and DMTGAS-COSMOS on all datasets.

Table 5 lists the experimental results obtained by DMTGASIC and DMTGAS on all datasets in terms of accuracy (%) for NC and GC and AUC (%) for LP.

Table 6 lists the search times (clock time in seconds) of all NAS baselines. The search times of our proposed method are shorter than those of NAS baselines, which further demonstrates the superiority of DMTGAS in terms of search efficiency. Compared with linear scalarization, the MOO method searches for better-performing architectures (see Table 2) in a similar search time, which shows the efficiency of applying MOO. Both the upper and lower levels only need to maintain one solution

Table 5: The experimental results obtained by DMTGASIC and DMTGAS on all datasets in terms of accuracy (%) for NC and GC and AUC (%) for LP.

Methods	ENZYMES (Mean(Std))			PROTEINS (Mean(Std))			DHFR (Mean(Std))		
	GC	NC	LP	GC	NC	LP	GC	NC	LP
DMTGASIC-EPO	39.3 (0.017)	85.4 (0.015)	93.3 (0.006)	83.5 (0.022)	72.6 (0.068)	96.1 (0.038)	74.7 (0.014)	63.8 (0.018)	99.2 (0.001)
DMTGAS-EPO	**56.7** (0.049)	**89.1** (0.012)	**95.8** (0.022)	**87.0** (0.004)	**82.7** (0.005)	**96.8** (0.043)	**83.3** (0.023)	**86.2** (0.004)	**99.9** (0.006)
DMTGASIC-COSMOS	30.0 (0.035)	80.6 (0.012)	93.2 (0.015)	76.3 (0.057)	72.6 (0.000)	87.9 (0.066)	62.2 (0.084)	82.0 (0.042)	68.0 (0.009)
DMTGAS-COSMOS	**52.7** (0.040)	**89.2** (0.006)	**94.3** (0.006)	**88.9** (0.024)	**81.7** (0.013)	**96.2** (0.011)	**79.2** (0.027)	**94.5** (0.014)	**98.7** (0.009)

instead of Pareto sets in our proposed optimization strategy. Furthermore, our proposed method obtains one model that handles multiple tasks in a single run, whereas ST-SANE requires a model that is individually tailored for each task. Therefore, the search times of our method are much shorter than those of ST-SANE, indicating the necessity of multi-task graph representation learning.

Table 6: The search times (clock time in seconds) of all NAS baselines.

	ENZYMES	PROTEINS	DHFR
Random Search	162063	346275	289545
ST-SANE	3,762	5,621	6,556
DMTGAS-LS	1,373	3,006	2,862
DMTGAS-EPO	2,023	2,829	2,975
DMTGAS-COSMOS	1,832	3001	2888

Table 7 lists experimental results of DMTGAS-EPO with different non-uniform preferences $\boldsymbol{r}$ including (0.80,0.10,0.10), (0.10,0.80,0.10), (0.10,0.10,0.80), and (0.33,0.33,0.33) in terms of accuracy (%) for NC and GC and AUC (%) for LP. By introducing different preferences $\boldsymbol{r}$, DMTGAS-EPO can achieve excellent performance on a certain task that cannot be achieved by single-task baselines (see Table 2 and Table 7). This further suggests that the MTGNN may have stronger representational capabilities than the single-task GNN through knowledge transfer.

Table 7: The experimental results of DMTGAS-EPO with different non-uniform preferences $\boldsymbol{r}$ in terms of accuracy (%) for NC and GC and AUC (%) for LP.

Methods	ENZYMES (Mean(Std))			PROTEINS (Mean(Std))			DHFR (Mean(Std))		
	GC	NC	LP	GC	NC	LP	GC	NC	LP
DMTGAS-EPO-(0.80,0.10,0.10)	**63.9** (0.027)	88.3 (0.023)	81.3 (0.015)	**89.8** (0.014)	78.8 (0.010)	84.1 (0.014)	80.6 (0.049)	84.0 (0.005)	97.1 (0.012)
DMTGAS-EPO-(0.10,0.80,0.10)	56.7 (0.014)	**93.1** (0.013)	82.6 (0.019)	73.9 (0.020)	**87.3** (0.009)	90.0 (0.015)	76.7 (0.029)	**97.0** (0.007)	99.5 (0.003)
DMTGAS-EPO-(0.10,0.10,0.80)	55.7 (0.059)	87.6 (0.022)	**95.8** (0.011)	73.0 (0.018)	85.9 (0.012)	96.2 (0.014)	78.7 (0.042)	86.9 (0.009)	**99.9** (0.005)
DMTGAS-EPO-(0.33,0.33,0.33)	56.7 (0.049)	89.1 (0.012)	**95.8** (0.022)	87.0 (0.004)	82.7 (0.005)	**96.8** (0.043)	**83.3** (0.023)	86.2 (0.004)	**99.9** (0.006)