MIM-Reasoner: Learning with Theoretical Guarantees for Multiplex Influence Maximization

Nguyen Do[1] **Tanmoy Chowdhury**[2] **Chen Ling**[2] **Liang Zhao**[2] **My T. Thai**[3,†]
[1]Posts and Telecommunications Institute of Technology, Ha Noi, Viet Nam
[2]Emory University, Atlanta, USA, [3]University of Florida, Gainesville, USA

Abstract

Multiplex influence maximization (MIM) asks us to identify a set of seed users such as to maximize the expected number of influenced users in a multiplex network. MIM has been one of central research topics, especially in nowadays social networking landscape where users participate in multiple online social networks (OSNs) and their influences can propagate among several OSNs simultaneously. Although there exist a couple combinatorial algorithms to MIM, learning-based solutions have been desired due to its generalization ability to heterogeneous networks and their diversified propagation characteristics. In this paper, we introduce MIM-Reasoner, coupling reinforcement learning with probabilistic graphical model, which effectively captures the complex propagation process within and between layers of a given multiplex network, thereby tackling the most challenging problem in MIM. We establish a theoretical guarantee for MIM-Reasoner as well as conduct extensive analyses on both synthetic and real-world datasets to validate our MIM-Reasoner's performance.

1 INTRODUCTION

Many users on Online Social Networks (OSNs), such as Facebook and Twitter, are increasingly linking their accounts across multiple platforms. The multiple linked OSNs with overlapping users is defined as a Multiplex Network. The structure of multiplex networks allows more users to post information across various OSNs simultaneously, offering significant value for marketing campaigns (Lim et al., 2015). Although this interconnectivity of the multiplex network enables seamless information flow between platforms via overlapping nodes, the underlying information propagation models on each OSN can differ. This means that the way information spreads and influences users on one OSN may not be the same as on another OSN. Given these distinct characteristics, designing a customized influence maximization strategy to wield considerable influence over various platforms becomes imperative.

As a combinatorial optimization problem, Influence Maximization (IM) aims at selecting a small subset of users to maximally spread information throughout the network. In the past decades, tremendous combinatorial optimization algorithms have been proposed on a single network (Kempe et al., 2003; Kimura and Saito, 2006; Jiang et al., 2011; Tang et al., 2015; Nguyen et al., 2016; Li et al., 2018; Tang et al., 2018; Li et al., 2019; Guo et al., 2020). They have achieved faster running time with approximation and exact solutions under certain propagation models. In parallel to traditional combinatorial optimization-based methods, machine learning based approaches (Li et al., 2022, 2023; Chen et al., 2022; Ling et al., 2023) have also been proposed in recent years and achieved success in handling massive complicated networks and generalizing well to similar problems. Both types of methods cannot be trivially adapted to the multiplex network scenario since they only consider a single model of influence propagation, whereas each network in a multiplex network can have a different propagation model.

To date, a few combinatorial optimization algorithms (Zhan et al., 2015; Zhang et al., 2016; Kuhnle et al., 2018; Singh et al., 2019; Katukuri et al., 2022) have been proposed to tackle Multiplex IM (MIM). However, despite the numerous benefits offered by machine learning-based approaches, the progress of applying such methods to MIM is still in its infancy owing to the following two fundamental challenges. *(1) Scalability.* A multiplex network is of very large size

[†]Correspondence to: *mythai@cise.ufl.edu*. Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

with heterogeneous propagation models, of which each layer has its own characteristics. Current state-of-the-art (SOTA) IM learning-based methods face computational limitations, especially in the case of simulating and estimating the propagation spread within and between layers. *(2) Generalizability For Lightweight Model.* In multiplex networks, capturing diverse propagation characteristics often requires large GNN-based models. However, using such large GNN models significantly increases the time complexity, resulting in longer training times and slower inference processes. Balancing the model size and computational efficiency is crucial when working with multiplex networks.

Our Contributions. To overcome both scalability and generalization for lightweight model issues altogether, we propose a novel framework MIM-Reasoner that decomposes multiplex networks into individual layers and leverages deep reinforcement learning to find near-optimal seed nodes for the multiplex network as a whole. Specifically, we first employ the Knapsack approach to assign appropriate budgets to each layer, while aiming to maximize the overall spread of the whole multiplex network, thereby obtaining a $(1-\epsilon)$-decomposition. With such a decomposition strategy, MIM-Reasoner learns a lightweight policy to find feasible solutions for each layer and minimize the overall computational workload and complexity of the whole network. Through the learning process, MIM-Reasoner models the interdependent relationship between the current layer and other layers, which avoids reactivating nodes and maximizes the overall spreading process. In addition, we establish guarantees for the solutions discovered by the optimal policy of MIM-Reasoner in the worst-case, best-case, and general scenarios. We empirically demonstrate the strength of MIM-Reasoner in both synthetic and real-world datasets from the perspective of influence spread and running time.

2 PROBLEM STATEMENT AND BACKGROUND

A multiplex network consisting of k layers is represented by $\mathscr{G} = \{(G_1, \sigma_1), \ldots, (G_k, \sigma_k)\}$, where each element consists of a directed graph $G_i = (V_i, E_i)$, and an influence model σ_i (i.e Independent Cascade (IC) or Linear Threshold (LT) (Kempe et al., 2003)) that describes the propagation of influence within G_i. If a node belongs to multiple layers, an interlayer edge is added between corresponding nodes to signify the node's overlapping presence. The entire set of nodes in the multiplex is denoted by V where $V = \bigcup_{i=1}^{k} V_i$. Without loss of generality, we can assume that $V_i = V$ for all i. If a vertex $v \in G_i$ does not exist in some G_η, we can simply add it to G_η as an isolated vertex. In this work, as we allow each layer of a multiplex network to have a different model of influence propagation, it becomes necessary to establish a mathematical definition for the propagation model on $\mathscr{G}$.

Definition 1 (*Influence Propagation Model)* (Kuhnle et al., 2018). A model of influence propagation, denoted as σ_i, on a graph $G_I = (V, E_i)$ is defined by a function P that assigns probabilities to the final activated sets $T \subset V$ given a seed set $S \subset V$. The probability $P(T \mid S) \in [0, 1]$ satisfies the property $\sum_{T:T\subset V} P(T \mid S) = 1$, ensuring that we have a probability distribution. The expected number of activated nodes, denoted as $\sigma_i(S)$, given a seed set S, is calculated as follows:

$$\sigma_i(S) = \sum_{T:T\subset V} P(T \mid S) \cdot |T| \tag{1}$$

The above definition covers most of the propagation models in the literature, such as IC, LT, and SIR (Kempe et al., 2003).

The influence propagation model σ on $\mathscr{G}$ is defined as follows: if an overlapping node v is activated in one layer graph G_i, then its adjacent interlayer copies in other layers also become activated in a deterministic manner, called *overlapping activation*. The propagation of influence occurs independently within each graph G_i according to its respective propagation model σ_i. Note that we count the duplicated nodes as a single instance rather than adding up all of them. We are now ready to define our MIM problem as follows:

Definition 2 (**Multiplex Influence Maximization (MIM)**). Given a multiplex graph $\mathscr{G} = \{(G_1 = (V, E_1), \sigma_1), \ldots, (G_k = (V, E_k), \sigma_k)\}$ and a budget $l \in \mathbb{N}$, the MIM problem asks us to find a seed set $S \subset V$ of size at most l so as to maximize the expected number of activated nodes in the multiplex network denoted as $\sigma(S)$. An instance of this problem is denoted as $(\mathscr{G}, k, l, \sigma)$ and finding the optimal set of seed node $\tilde{S}$ that maximizes spreading among those graphs follows the following function:

$$\tilde{S} = \arg\max_{|S|\leq l} \sigma(S) \tag{2}$$

For each layer $G_i \in \mathscr{G}$, many greedy based algorithms (Leskovec et al., 2007; Goyal et al., 2011; Tang et al., 2014, 2015) have obtained a performance guarantee bound of $(1 - 1/e)$, if σ_i is submodular and monotone increasing (Kempe et al., 2003). If all σ_i of all G_i satisfy the Generalized Deterministic Submodular (GDS) property, then σ is submodular (Kuhnle et al., 2018).

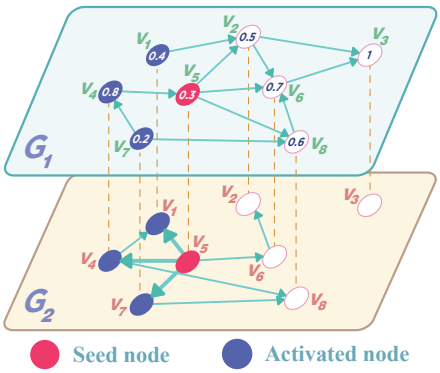

Figure 1: An example of the propagation process in a multiplex network consisting of 2 layers. Layers G_1 and G_2 operate LT and IC model, respectively. Each node in G_1 has a threshold $\zeta \in [0, 1]$. The green bold arrow in G_2 indicates high probability of activation.

To visualize the propagation process in multiplex networks, let's consider an instance of the MIM problem denoted as $(\mathscr{G}, 2, 1, \sigma)$. Here, $\mathscr{G} = (G_1 = (V, E_1), \sigma_1), (G_2 = (V, E_2), \sigma_2)$ represents a two-layer multiplex network as shown in Figure 1 where σ_1 of G_1 is an LT model and σ_2 of G_2 follows an IC model. In this instance, node v_5 is selected as the seed node. Assuming that in layer G_2, v_5 activate nodes $[v_1, v_4, v_7]$. Those nodes in layer G_1 are also activated due to the overlapping activation property. Moreover, v_8 has a low chance to be activated by other nodes in layer G_2. However, in layer G_1, node v_8 has a threshold $\zeta = 0.6$, meaning it requires at least two activated neighboring nodes to be activated itself. In this case, $[v_5, v_7]$ are activated, leading to the activation of v_8 in both layers. Similarly, the activation process continues to propagate to other nodes, such as $[v_2, v_6, v_3]$ in layer G_1, hence, $[v_2, v_6, v_3]$ in layer G_2 are also activated in the deterministic manner. As a result, by properly selecting seed nodes, we only need one budget to activate all nodes in $\mathscr{G}$.

3 RELATED WORK

Combinatorial optimization for IM. IM, which was firstly introduced by Kempe et al. (2003), is a well-studied problem in network analysis. Traditional approaches include simulation-based (Leskovec et al., 2007; Goyal et al., 2011; Zhou et al., 2015), proxy-based (Kimura and Saito, 2006; Chen et al., 2010a,b), and approximation-based methods (Li et al., 2018; Jiang et al., 2011; Tang et al., 2015; Nguyen et al., 2016; Tang et al., 2018; Guo et al., 2020). Most of these works obtained a $(1 - 1/e)$-approximation ratio due to the submodularity of propagation models. Recently, Tiptop (Li et al., 2019) has been introduced as an almost exact solution to IM. For more detailed reviews of traditional methods, we refer readers to a recent survey by Banerjee et al. (2020). In the context of MIM, the notable *approximation* algorithms with theoretical guarantees using combinatorial methods are (Zhang et al., 2016; Nguyen et al., 2013; Kuhnle et al., 2018). In (Zhang et al., 2016; Nguyen et al., 2013), the authors addressed the Least Cost Influence problem by mapping layers into a single layer using lossless and lossy coupling schemes. These works assume that all layers have the same propagation model. Later, (Kuhnle et al., 2018) considered the heterogeneous propagation models and presented Knapsack Seeding of Networks (KSN), which serves as a starting point for our learning model.

ML-based for IM. The rise of learning-based approaches that utilize deep learning techniques to enhance generalization capabilities has been applied to IM. Reinforcement learning (RL) integrated with IM has shown promise (Lin et al., 2015; Ali et al., 2018), with recent state-of-the-art solutions focusing on learning latent embeddings of nodes or networks for seed node selection (Manchanda et al., 2020; Chen et al., 2022; Li et al., 2022). Graph neural networks (GNNs) have also been explored in IM to encode social influence and guide node selection (Ling et al., 2022b,a). Recent methods (Ling et al., 2023; Chowdhury et al., 2024) have proposed DeepIM as a generative approach to solving IM and achieved state-of-the-art performance. However, these methods only focused on the IM problem, and could not readily be extended to our MIM problem due to the scalability issues and the ability to capture the inter- and intra-propagation relations in a multiplex network.

4 MIM-REASONER

In this section, we introduce our MIM-Reasoner, a deep reinforcement learning (RL) approach coupled with a Probabilistic Graphical Model (PGM) designed to maximize influence in a multiplex network under a budget constraint (Figure 2). In a nutshell, to tackle the scalability issue, MIM-Reasoner, decomposes $\mathscr{G}$ into individual layers, enabling it to find solutions for each individual layer in parallel in the first phase. It utilizes a knapsack-based approach (Chandra et al., 1976) in order to effectively allocate the budget for each layer. Furthermore, to capture the inter-activation via overlapping users, in the second phase, we train a PGM for each layer. Based on PGMs, MIM-Reasoner then learns a policy π that operates sequentially from one layer to another. The PGMs themselves suggest the rewards that can aid policy π further tuning the solutions found in each layer to ob-

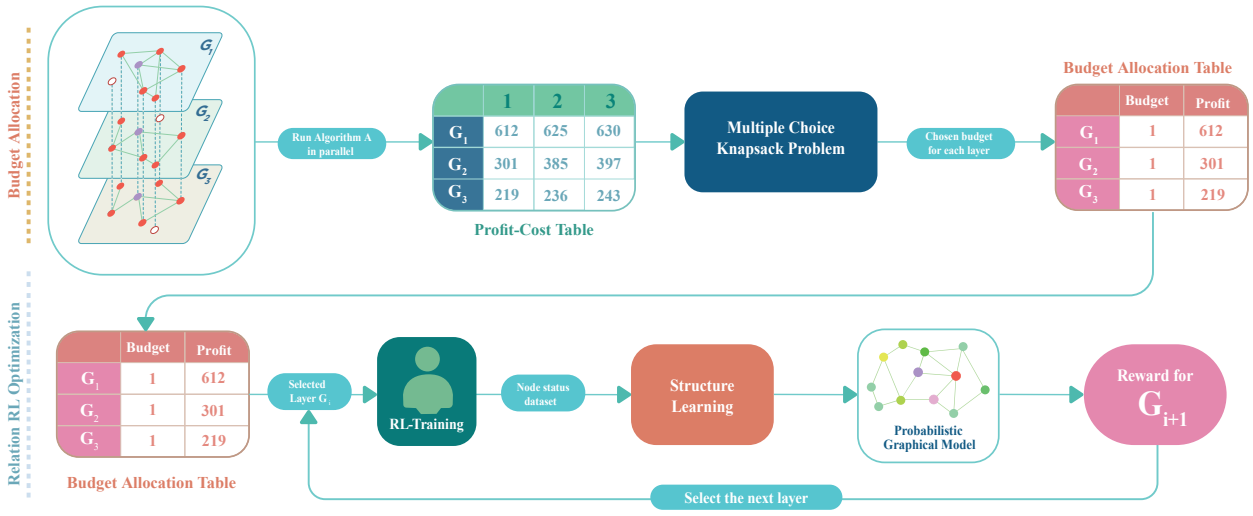

Figure 2: MIM-Reasoner consists of two phases: Budget Allocation and Relation RL Optimization. In Phase 1, algorithm $\mathcal{A}$ calculates the profit and cost for each layer in parallel. The 'Multiple Choice Knapsack Problem' is then solved to determine the allocated budget for each layer, presented in the 'Budget Allocation Table'. In Phase 2, an RL-Agent is trained to sequentially find solutions for each layer using the allocated budget. Simultaneously, the status dataset is processed through 'Structure Learning' to create a 'Probabilistic Graphical Model' which reveals layer relationships and helps the RL-Agent to avoid reactivating nodes already activated by other layers.

tain our final solution within a bound of $\frac{(1-\epsilon)(1-1/e)}{(o+1)k}$.

4.1 Phase 1. Budget Allocation

This phase aims to find an initial candidate seed set solution serving as a starting point for our deep RL agent in Phase 2. In particular, the key point is how to allocate the budget for each layer so that it can maximize the overall expected number of activated nodes while solving each layer individually. Firstly, this phase involves running any algorithm $\mathcal{A}$ that solves the IM problem (e.g., a Greedy-based algorithm) with budget l for each layer G_i to measure $\sigma(S_{ij})$ where $j \in [0, ..., l]$. Note that G_i can be in billion-scale. Thus, Let's denote $\mathcal{E}_i = [\hat{G}_{i,1}, ..., \hat{G}_{i,h}]$ as a set of subgraphs sampled from i^{th} layer of $\mathscr{G}$; i.e. G_i. Here, h represents the total number of sampled subgraphs of i^{th} layer. We adopt a Graph Attention Network (GAT), denoted as $\mathcal{I}(.)$ on a set of subgraphs $\mathcal{E} = [\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_k]$ by following a probabilistic greedy concept (Manchanda et al. (2020)) (details in Appendix **A.1**). This trained GAT helps identify a good candidate set $V_i^g \subset V$ before running $\mathcal{A}$, thereby further reducing the search space and improving efficiency.

Given a feasible seed node set S_{ij} found by $\mathcal{A}$ for each G_i, MIM-Reasoner constructs a Profit-Cost table $\mathcal{H}$ that records the profit $p(S_{ij}) = \sigma(S_{ij})$ associated with the overall spreading achieved in $\mathscr{G}$, while the cost $c(S_{ij}) = |S_{ij}|$ indicates the corresponding budget j spent to achieve that spreading. We next solve the Multiple Choice Knapsack Problem (MCKP) to allocate budgets effectively for each layer. In the MCKP, we aim to maximize the total profit $\sum_{i=1}^{k} p(S_{ij})$ while satisfying the constraint $\sum_{i=1}^{k} c(S_{ij}) \leq l$. For $\epsilon > 0$, MCKP solver (Chandra et al., 1976) has a $(1-\epsilon)$ approximation for arbitrary $\epsilon > 0$. The output of MCKP is the budget allocation table $\mathcal{U}^{k\times 2}$, where each row corresponds to a layer G_i, and the two columns represent the spent budget and profit, respectively.

4.2 Phase 2. Relation RL Optimization

In this core phase, MIM-Reasoner's goal is to train the policy π to optimize our obtained solution, given the budget allocation table $\mathcal{U}$, achieved in Phase 1. Phase 2 has a key component which is a PGM (Koller and Friedman (2009)) denoted as φ which is specifically a Bayesian Network trained to capture the complex propagation process within and between layers to provide a novel reward signal to the RL agent. This reward signal assists the agent in finding a nearly optimal seed node set for the current layer while avoiding the reactivation of nodes already activated by previously selected layers.

To begin, let's define $\mathcal{G}^{prev} = [\varphi_1, ..., \varphi_{k-1}]$ as a set that contains the trained PGMs for each layer in a multiplex network. Initially, $\mathcal{G}^{prev} = \emptyset$ since no layer has been selected to learn yet. At each step, a layer G_i is selected to train the policy π based on the order

provided by $\mathcal{U}$ as follows:

$$G_i = \underset{i \in 1,\ldots,k}{\operatorname{argmin}} \mathcal{U}_{i,2} \tag{3}$$

Given an allocated budget $j \leq l$, MIM-Reasoner starts with the layer having the lowest profit, say $G_{\hat{i}}$. After selecting the starting layer $G_{\hat{i}}$ for the second phase, it removes row $\hat{i}$ from $\mathcal{U}$ to ensure that $G_{\hat{i}}$ will not be selected in the future. Given layer $G_{\hat{i}}$, it trains the policy π to improve the seed node set $S_{\hat{i}j}$, where $j = \mathcal{U}_{\hat{i},1}$, representing the allocated budget. During the policy training process, a status dataset $\mathcal{D} = [0,1]^{m \times |V|}$, where m represents the number of Monte Carlo simulation steps, is also collected. This dataset records the activation status of nodes in the currently selected layers and also indirectly records the activation of nodes in other layers due to the overlapping activation.

Next, MIM-Reasoner trains a PGM $\varphi_{\hat{i}}$ for the selected layer $G_{\hat{i}}$ using a step called Structure Learning, where the learned structure is a directed graph modeling the causal relationship between trained nodes. It takes $\mathcal{D}$ as input and returns a learned structure as output for $\varphi_{\hat{i}}$ (details in Appendix **A.3**). Note that, during the Structure Learning of $\varphi_{\hat{i}}$, only a subset of representative nodes are trained to reduce the complexity, other nodes that have high correlation with each other are grouped together but they can be inferred based on their Pearson Correlation (Cohen et al., 2009) with trained representative nodes. Details for this Variable Grouping are presented in Appendix **A.2**.

After training $\varphi_{\hat{i}}$, it is added to $\mathcal{G}^{prev}$. Since $\varphi_{\hat{i}}$ is trained on $\mathcal{D}$, which records the activation of nodes in other layers, it can predict the relationships between nodes in other layers. Suppose in the next step, the layer with the lowest influence G_η ($\eta \neq \hat{i}$) is chosen using Equation 3. This time, MIM-Reasoner utilizes $\mathcal{G}^{prev}$ to generate rewards that assist the policy π in selecting a seed set which ensures that nodes activated by the previously selected layers (i.e., $G_{\hat{i}}$) are not reactivated. These reward functions are discussed next in the (RL) framework.

4.3 Reinforcement Learning Framework

This section presents our RL framework, which includes state representation, action space, and reward function. Moreover, we also explain how PGM φ helps the RL agent to effectively maximize the overall spread within the allocated budget for each layer of $\mathscr{G}$.

State Representation: We leverage $\mathcal{I}(.)$ trained in Phase 1, to predict a set of candidate nodes $V^g \subset V$ for an unseen graph $\mathscr{G}$ and even further reduce the complexity of the state space and action space. Note that V_i^g represents the set of candidates that potentially contain optimal or near-optimal seed nodes in G_i. The state space, $C_{i,t} = (V_i^g, X_i, S_{i,t}, t)$, describes the state of G_i at time step t while X_i is the feature vector of layer G_i extracted by Structure2vec (Dai et al., 2016), $S_{i,t}$ is the partially computed solution at time step t, and V_i^g represents the set of candidate nodes predicted from the pretrained GAT, $\mathcal{I}(.)$.

Action Space: In the context of our problem, at each time step t, the policy π will select an action that corresponds to a seed node $v \in V_i^g \backslash S_{i,t}$.

Reward Function. Recall in the Variable Grouping step, we used our proposed Node Grouping algorithm to identify highly correlated node groups denoted as $\mathcal{P} = [P_1, P_2, \ldots, P_q]$ which satisfies $V = \bigcup_{i=1}^{q} P_i$ (Details in Appendix **A.2**). Our grouping algorithm automatically divides the nodes into q clusters based on correlation without human involvement. Each group P_i contains nodes that exhibit high correlation with each other. We define the set of representative nodes by taking the node closest to the centroid from each group as $\mathcal{Y} = [v_1, \ldots, v_q]$, where each representative node $v_i \in \mathcal{Y}$ corresponds to a group $P_i \in \mathcal{P}$. During the training process, if the agent identifies a new seed node v_n and adds it to a partially computed solution $S_{i,t}$, the updated set $S_{i,(t+1)} = S_{i,t} \cup v_n$ can activate a subset of nodes $T \subseteq V$. It is important to note that each node $u \in T$ has its own representative node.

For convenience in determining which group a node $u \in T$ belongs to and which node is the representative node for u, we define a surjective function $f : T \longrightarrow \mathcal{Y}$, where for all nodes $\hat{v} \in \mathcal{Y}$, there would be at least an element $u \in T$ such that $f(u) = \hat{v}$. Suppose the policy π is currently trained on layer G_i, we define an activation score function that allows every node $u \in T$ to be based on the representative node set $\mathcal{Y}$ to infer its score for being activated by any other selected layer $G_{\hat{i}}$ using PGMs. This function can be defined as:

$$\mathcal{W}(u) = Q_{u,f(u)} \cdot \underset{\varphi_{\hat{i}} \in \mathcal{G}^{prev}}{\operatorname{argmax}} \varphi_{\hat{i}}(f(u), \mathcal{K}) \tag{4}$$

Here, $Q_{u,f(u)} \in [-1,1]$ represents the Pearson correlation between node u and its corresponding representative node $f(u)$. Meanwhile, $\mathcal{K} = \mathcal{Y} \cap T$ refers to the set of activated representative nodes. On the other hand, in layer G_i, the output of $\varphi_{\hat{i}}(f(u), \mathcal{K})$ represents the conditional probability of representative node $f(u)$, (i.e. $P_{\hat{i}}(f(u) \mid \mathcal{K})$) being activated by other layer $G_{\hat{i}}$, given that the current set of activated representative nodes is $\mathcal{K}$. Putting $\mathcal{K}$ as evidence to PGM indicates how likely the representative node $f(u)$ is to be activated by other layers when certain other representative nodes are already active. Based on activation score $\mathcal{W}$, we now present our customized evaluation

function $\mathcal{M}(.)$ which measures the total spread in layer G_i, given solution S_i. It can be defined as:

$$\mathcal{M}(S_i) = \sum_{i=1}^{|T|}[1 - \mathcal{W}(u)] \tag{5}$$

In Equation 5, if a node u has a high activation score, it means that u is more likely to be activated by previously selected layers. Thus, instead of counting the contribution of u to the total spread as 1, we can modify it to be $1 - \mathcal{W}(u)$. This means that if the reward function is based on $\mathcal{M}$ to calculate marginal gain, the policy π has less reward when activating node u, hence, it avoids finding a seed set that re-activates nodes that have been already activated by other layers. Finally, we provide our reward function to train policy π computed by considering the marginal gain of adding node v_t to the partially computed solution S_t, as follows:

$$r_t = \mathcal{M}(S_{i,(t+1)}) - \mathcal{M}(S_{i,t}) \tag{6}$$

Here, $\mathcal{M}(S_{i,(t+1)})$ is the total spread given a partially computed solution at time step $t+1$, and $\mathcal{M}(S_{i,t})$ represents the spread achieved by the current solution $S_{i,t}$ for the i^{th} layer of $\mathscr{G}$. The solution set for the whole $\mathscr{G}$ would be:

$$\hat{S} = \bigcup_{i=1}^{k} S_i \tag{7}$$

4.4 MIM-Reasoner Analysis

In order to bound the solution of MIM-Reasoner when policy π is converged, it is necessary to ensure that the optimal policy π^* follows a greedy strategy. The following lemma assists in ensuring this guarantee:

Lemma 1. *(Greedy Policy Guarantee). When policy π is converged, $\pi^*(v \mid S_{i,t})$ always selects nodes greedily at every time step t. (Proof in Appendix **B.2**)*

Based on Lemma 1, for every layer $G_i \in \mathscr{G}$, the solution provided by policy π^* is within $(1-1/e)$ of the optimal solution, given that σ_i is submodular and monotone increasing. Our next concern lies in the quality of MCKP. During phase 1, since $\mathcal{A}$ runs in parallel for each layer G_i, we also achieve an approximation ratio of $(1 - 1/e)$ as stated in the following lemma.

Lemma 2. *(Multiple Choice Knapsack Problem Guarantee). Let Opt_S be the value of the solution for MCKP instance S, and $Opt_{\tilde{S}}$ be the value of the optimal solution for $\tilde{S}$, we have: $Opt_S \geq (1 - 1/e)Opt_{\tilde{S}}$ (Proof in Appendix **B.3**)*

To ensure the solution is bounded, it is important to include near-optimal or optimal nodes in the search space $\mathcal{I}(.)$. If $\mathcal{I}(.)$ consists of K layers, the prediction of any node $v \in V$ is influenced by its K-hop neighbors (Zhou et al. (2018)). Although the graph itself can be changed, the underlying model generating the graph often remains consistent. These observations lead to an assumption that poor nodes will be eliminated in the solution set, while only good nodes are retained by the GNN-based model (Manchanda et al., 2020). Thus, it is possible that $\mathcal{I}(.)$ trained on a set of subgraphs $\mathcal{E}$ can also find solutions on similar graphs in $\mathscr{G}$. We formally restate these observations in the following assumption:

Assumption 1. *(Near Optimal Nodes Are Included In Search Space). In each layer $G_i \in \mathscr{G}$, the near optimal seed nodes S^* are included in V^g provided by pretrained $\mathcal{I}(.)$. In other word, we have $S^* \subseteq V^g$.*

We now approximate the solution quality in the worst case. Let us denote o, $\hat{S}^{\pi^*}$ as the total number of overlapping nodes in $\mathscr{G}$ and final solution found by π^*. The optimal policy π^* has an approximation guarantee for the worst case as follows:

Theorem 1. *(Approximation Ratio In The Worst Case) Suppose the propagation σ_i on each layer of the multiplex is submodular, the optimal policy π^* will find a solution $\hat{S}$ for multiplex network $\mathscr{G}$ with an approximation ratio of $\frac{(1-\epsilon)(1-1/e)}{(o+1)k}$. (Proof in Appendix **B.4**)*

In case the optimal policy π^* finds the solution $S_i^{\pi^*}$ for each layer G_i that completely avoids reactivating nodes already activated by other layers, we can express the approximation ratio of π^* for the best case.

Theorem 2. *(Approximation Ratio In The Best Case). Assume the π^* can avoid reactivating all the activated nodes, the spread of solution given by optimal policy π^* is at least: $\sigma\left(\hat{S}^{\pi^*}\right) \geq \frac{(1-\epsilon)(1-1/e)}{k+o}\sigma(\tilde{S})$*

Let us denote $\beta \in [0, 1]$ as the percentage of nodes that cannot be successfully avoided reactivation by the policy π^*, we can establish performance guarantees in a typical scenario where the optimal policy π^* can partially avoid reactivating nodes that have already been activated by other layers.

Theorem 3. *(Approximation Ratio In The General Case). Assume the π^* can avoid to partially reactivate the activated nodes by other layers. Thus, with $\beta \in [0, 1]$, the spread of solution given by optimal policy π^* is at least: $\sigma\left(\hat{S}^{\pi^*}\right) \geq \frac{(1-\epsilon)(1-1/e)}{(k-1)\beta o+o+k}\sigma(\tilde{S})$*

Proofs of Theorem 2 and Theorem 3 are shown in Appendix **B.5-B.6**. The time complexity of the MIM-Reasoner depends on the total number of PGMs stored in $\mathcal{G}^{prev}$. We have the following lemma:

Lemma 3. *(PGMs's Time complexity). The time complexity of structure learning for $\mathcal{G}^{prev}$ after k selection step is $|\mathcal{Y}|^2 \cdot (k-1)$. (Proof in Appendix **B.7**)*

Given Lemma 3, we now bound the time complexity of MIM-Reasoner. Assuming we have k layers, the time complexity of algorithm A on l seed nodes with graph G_h (representing the layer with the highest number of nodes and edges) is denoted as $tc(A, G_h, l)$. The time complexity of training MIM-Reasoner is described as:

Theorem 4. *(Time complexity of MIM-Reasoner). The time complexity of the Budget Allocation is: $\max_{h \in k} tc(A, G_h, l) + (kl)^{\lceil 1/\epsilon - 1 \rceil} \log k$ and the time complexity of Relation RL Optimization is : $O(|\mathcal{Y}|^2 \cdot (k-1) + \mathcal{Q})$ where $\mathcal{Q}$ is number of step for policy π converge to optimal. (Proof in Appendix **B.8**)*

5 EXPERIMENT EVALUATION

This section compares the performance of MIM-Reasoner across four real multiplex networks and one synthetic multiplex network in maximizing the influence under various settings, following a case study to qualitatively demonstrate its performance. The code and datasets for MIM-Reasoner are available at the following GitHub repository: `https://github.com/nguyendohoangkhoiUF/MIM-Reasoner`.

5.1 Experiment Setup

Our purpose is to evaluate the expected influence spread as defined in Equation (2) and time efficiency under a multiplex network scenario. For more detailed information about the experiment setup and our findings, we encourage readers to refer to Appendix **C.1 and C.2**.

Comparison Methods And Metrics. We compare the MIM-Reasoner with two sets of approaches. 1) Traditional multiplex influence maximization methods: *ISF (Influential Seed Finder)* (Kuhnle et al., 2018) is a greedy algorithm designed for multiplex influence maximization; *KSN (Knapsack Seeding of Networks)* (Kuhnle et al., 2018) utilizes a knapsack approach to find the best seed users in a multiplex network. 2) Deep Learning based influence maximization methods for single network: *ToupleGDD* (Chen et al., 2022) is a Deep Reinforcement Learning-based solution trained on many small networks for better generalization ability; *DeepIM* (Ling et al., 2023) is a state-of-the-art IM solution based on deep generative models. The comparison is based on three metrics: total influence spread (activated nodes), running time (in seconds) and inference time (in seconds).

Synthetic Dataset: We compare MIM-Reasoner with other baselines on a synthetic multiplex network with 5,000 nodes and 25,000 edges in various overlapping rates (30%, 50%, and 70%) and the number of layers (ranging from 3 to 9).

Real Dataset: We evaluate MIM-Reasoner on four real-world multiplex networks: 1) Celegans (Stark et al., 2006): 6 layers, 3879 nodes, and 8191 edges; 2) Drosophila (Stark et al., 2006; De Domenico et al., 2015): 7 layers, 8215 nodes, and 43,366 edges; 3) Twitter-Foursquare network (Shen et al., 2012): 2 layers, 93269 nodes, and 17,969,114 edges; 4) Pope-Election (Domenico and Altmann, 2020): 3 layers, 2,064,866 nodes, and 5,969,189 edges.

Hyperparameter Setting. The experiments were run with budget $l = 30$, and Monte Carlo simulation $mc = 100$. The early stopping was also applied for all methods. Specifically, DeepIM is stopped when the increment of reconstruction accuracy is not larger than ϵ for every 25% of the total number of epochs. For ToupleGDD and MIM-Reasoners, for every 25% of the total number of epochs, the early stop was applied if the current solution was not better than the best one.

5.2 Training Time Analysis

The "running time" of combinatorial algorithms (such as ISF or KSN) and learning-based approaches (such as DeepIM, ToupleGDD, and MIM-Reasoner) is interpreted as follows. Combinatorial algorithms measure running time as the duration to find a solution, while learning-based approaches refer to the running time as the time needed for training the model to converge and achieve a solution with a total spread reported. Therefore, this section focuses on analyzing the running time (training time) of learning-based methods.

Compared with Deep Learning-based influence maximization solutions, i.e., DeepIM and ToupleGDD, the running time for MIM-Reasoner is typically the lowest in both synthetic and real dataset as depicted in Figure 3 and Table 1. By parallelizing the IC model on each individual layer for every optimization step, MIM-Reasoner reduces the complexity of the propagation model significantly. In contrast, the size of the network directly affects the number of training samples needed for DeepIM, resulting in longer training times to provide decent solutions. Meanwhile, ToupleGDD running the propagation model on the entire multiplex network (with a much larger node and edge count) would naturally require more computational time for each step.

Interestingly, as illustrated in Figure 3, the training time of MIM-Reasoner decreases as the number of layers increases. In fact, it becomes comparable to the running time of combinatorial algorithms such as ISF

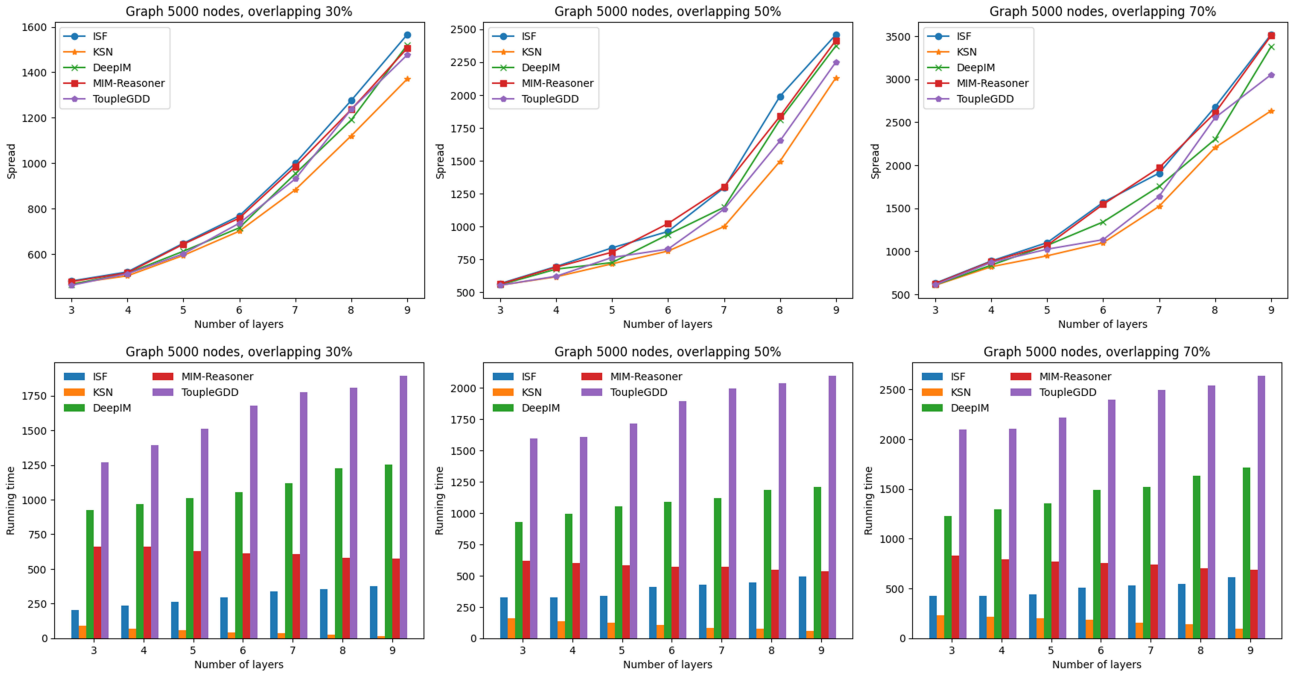

Figure 3: Comparison of five methods on a synthetic dataset, with different overlapping percentages and layers. The comparison is based on two metrics: total spreading and running time (in seconds).

Methods	Celegans			Drosophila			Twitter-Foursquare			Pope-Election		
	TS	**RT**	**IT**	**TS**	**RT**	**IT**	**TS**	**RT**	**IT**	**TS**	**RT**	**IT**
ISF	**1412.21**	335.96	**NaN**	3268.22	3087.58	**NaN**	**x**	**x**	**NaN**	**x**	**x**	**NaN**
KSN	1267.14	**176.02**	**NaN**	2911.46	**1331.23**	**NaN**	49282.36	**11820.24**	**NaN**	**x**	**x**	**NaN**
ToupleGDD	1309.13	1181.72	1.12	3150.71	10945.98	2.74	49927.11	71428.59	7.46	246873.52	317431.28	13.21
DeepIM	1310.91	715.11	**0.23**	3250.58	8740.67	**0.49**	51989.23	43158.84	**1.17**	248251.49	251195.93	**1.98**
MIM-Reasoner	1314.25	580.76	0.28	**3462.94**	4367.41	0.57	**53420.94**	28372.40	1.81	**254238.20**	**99713.08**	2.27

TS: Total Spread; RT: Running Time (Seconds); IT: Inference Time (Seconds).

Table 1: Comparison of five methods on different real datasets in terms of total spreading, running time in seconds, and inference time in seconds. Best performance is highlighted with bold while **x** indicates an out-of-memory error. **NaN** means that traditional methods do not have an inference time.

in the synthetic dataset. This demonstrates that MIM-Reasoner has an advantage of reducing training time as layer complexity increases, while still achieving competitive performance in terms of total spread.

5.3 Inference Time Analysis

As shown in Table 1, MIM-Reasoner stands out as an efficient method, particularly because it can parallelly process separate layers of a multiplex network and conduct batch inference. Even though DeepIM is the fastest because it predicts end-to-end solutions directly; however, the training of DeepIM entails learning a large model, which is resource-consuming. ToupleGDD has to infer step by step each seed node, which makes the inference time longer than others. Moreover, it is crucial to compare the running time of combinatorial algorithms with the inference time of learning-based approaches. For instance, even the fastest running combinatorial algorithm like KSN, which takes 176.02s and 1331.23s to produce a solution in two small real network datasets respectively, cannot be compared to the efficiency of MIM-Reasoner, which only requires 0.28s and 0.57s. When considering larger graph sizes such as Twitter-Foursquare and Pope-Election, MIM-Reasoner still performs impressively, taking only 1.81s and 2.27s respectively, while producing a comparable total spread. In contrast, KSN takes 11820.24s for Twitter-Foursquare and cannot provide a solution for Pope-Election. Furthermore, ISF fails to provide solutions for both the enormous network Twitter-Foursquare and Pope-Election. This comparison highlights the advantages of using machine learning-based approaches when addressing the MIM problem.

5.4 Quantitative Analysis

Synthetic Dataset. In terms of other comparison methods, ISF, KSN, and DeepIM show similar trends in spreading, while ToupleGDD typically lags behind, particularly with higher overlap percentages. Among all the approaches evaluated, MIM-Reasoner consistently demonstrates competitive spreading values across all three overlapping percentages. It outperforms other methods, particularly as the number of layers in the multiplex network increases. Even with small graphs created in the synthetic dataset, MIM-Reasoner provides a total spread that is nearly as good as that of ISF. It is worth noting that while MIM-Reasoner may trade off solution quality for faster training and inference times, it still provides solutions with a total spread comparable to ISF. This is significant because ISF has a good approximation ratio $(1 - 1/e)$ in multiplex networks.

Real-world Dataset. The results of MIM-Reasoner on real-world datasets are summarized in Table 1. Among all the approaches evaluated, MIM-Reasoner consistently achieves either the highest or near-highest spreading across all datasets while providing fast inference times to generate solutions. This suggests the effectiveness of MIM-Reasoner in maximizing influence. In contrast, the performance of other methods, including ISF with its $1 - 1/e$ ratio in multiplex networks, varies across datasets, with none consistently outperforming MIM-Reasoner in terms of spreading. Traditional methods, such as ISF and KSN, can provide solutions with total spreads comparable to that of MIM-Reasoner. However, their running times are significantly slower than the inference time of MIM-Reasoner. Moreover, these traditional methods encounter issues such as out-of-memory errors or excessively long running times (e.g., running for over two weeks) on larger datasets like "Twitter-Foursquare" and "Pope-Election". This indicates their poor scalability in real-world scenarios.

6 CONCLUSION

We propose a novel framework named MIM-Reasoner to tackle the multiplex influence maximization problem. Specifically, we leverage deep reinforcement learning and decompose the multiplex network into separate layers. Our framework learns to allocate node selection budgets for each layer and employs a lightweight policy to find feasible solutions between each layer. A probabilistic graphical model is then designed to capture the propagation pattern between each layer in order to maximize the overall spread and improve the overall efficiency. The approximation ratio of the solution is provided with a theoretical guarantee. Finally, the proposed framework is compared with several SOTA approaches and demonstrates overall competitive performance on four real-world datasets and a synthetic dataset.

Acknowledgements

This material is partially supported by the National Science Foundation (NSF) under Grant No. FAI-1939725, SCH-2123809, and IIS-1908594, and the Department of Homeland Security (DHS) under Grant No. 17STCIN00001.

References

K. Ali, C.-Y. Wang, and Y.-S. Chen. Boosting reinforcement learning in competitive influence maximization with transfer learning. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 395–400. IEEE, 2018.

S. Banerjee, M. Jenamani, and D. K. Pratihar. A survey on influence maximization in a social network. *KAIS*, 62(9):3417–3455, 2020.

A. K. Chandra, D. S. Hirschberg, and C.-K. Wong. Approximate algorithms for some generalized knapsack problems. *Theoretical Computer Science*, 3(3): 293–304, 1976.

T. Chen, S. Yan, J. Guo, and W. Wu. Touplegdd: A fine-designed solution of influence maximization by deep reinforcement learning. *arXiv preprint arXiv:2210.07500*, 2022.

W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, page 1029–1038, New York, NY, USA, 2010a. Association for Computing Machinery. ISBN 9781450300551. doi: 10.1145/1835804.1835934. URL `https://doi.org/10.1145/1835804.1835934`.

W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *2010 IEEE International Conference on Data Mining*, pages 88–97, 2010b. doi: 10.1109/ICDM.2010.118.

T. Chowdhury, C. Ling, J. Jiang, J. Wang, M. T. Thai, and L. Zhao. Deep graph representation learning influence maximization with accelerated inference. *Available at SSRN 4663083*, 2024.

I. Cohen, Y. Huang, J. Chen, J. Benesty, J. Benesty, J. Chen, Y. Huang, and I. Cohen. Pearson correlation coefficient. *Noise reduction in speech processing*, pages 1–4, 2009.

H. Dai, B. Dai, and L. Song. Discriminative embeddings of latent variable models for structured data. *International Conference on Machine Learning*, 2016.

M. De Domenico, V. Nicosia, A. Arenas, and V. Latora. Structural reducibility of multilayer networks. *Nature communications*, 6(1):6864, 2015.

M. D. Domenico and E. G. Altmann. Unraveling the origin of social bursts in collective attention. *Scientific Reports*, 2020. doi: 10.1038/s41598-020-61523-z. URL `https://doi.org/10.1038/s41598-020-61523-z`.

A. Goyal, W. Lu, and L. Lakshmanan. Celf++: Optimizing the greedy algorithm for influence maximization in social networks. volume 47-48, pages 47–48, 03 2011. doi: 10.1145/1963192.1963217.

Q. Guo, S. Wang, Z. Wei, and M. Chen. Influence maximization revisited: Efficient reverse reachable set generation with bound tightened. In *Proc. of the SIGMOD*, pages 2167–2181, 2020.

Q. Jiang, G. Song, C. Gao, Y. Wang, W. Si, and K. Xie. Simulated annealing based influence maximization in social networks. In *Proc. of the AAAi*, 2011.

M. Katukuri, M. Jagarapu, et al. Cim: clique-based heuristic for finding influential nodes in multilayer networks. *Applied Intelligence*, 52(5):5173–5184, 2022.

D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proc. of the KDD*, 2003.

M. Kimura and K. Saito. Tractable models for information diffusion in social networks. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Knowledge Discovery in Databases: PKDD 2006*, pages 259–271, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46048-0.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009. ISBN 0262013193.

A. Kuhnle, M. A. Alim, X. Li, H. Zhang, and M. T. Thai. Multiplex influence maximization in online social networks with heterogeneous diffusion models. *IEEE Transactions on Computational Social Systems*, 5(2):418–429, 2018.

J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429, 2007.

H. Li, M. Xu, S. S. Bhowmick, J. S. Rayhan, C. Sun, and J. Cui. Piano: Influence maximization meets deep reinforcement learning. *IEEE Transactions on Computational Social Systems*, 2022.

X. Li, J. D. Smith, T. N. Dinh, and M. T. Thai. Tiptop: (almost) exact solutions for influence maximization in billion-scale networks. *IEEE/ACM Transactions on Networking*, 27:649–661, 2019. doi: 10.1109/TNET.2019.2898413.

Y. Li, J. Fan, Y. Wang, and K.-L. Tan. Influence maximization on social graphs: A survey. *TKDE*, 30(10):1852–1872, 2018.

Y. Li, H. Gao, Y. Gao, J. Guo, and W. Wu. A survey on influence maximization: From an ml-based combinatorial optimization. *ACM Transactions on Knowledge Discovery from Data*, 17(9):1–50, 2023.

J. S. Lim, S. Y. Ri, B. D. Egan, and F. A. Biocca. The cross-platform synergies of digital video advertising: Implications for cross-media campaigns in television, internet and mobile tv. *Computers in Human Behavior*, 48:463–472, 2015.

S.-C. Lin, S.-D. Lin, and M.-S. Chen. A learning-based framework to handle multi-round multi-party influence maximization on social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 695–704, 2015.

C. Ling, T. Chowdhury, J. Jiang, J. Wang, X. Zhang, H. Chen, and L. Zhao. Deepgar: Deep graph learning for analogical reasoning. In *2022 IEEE International Conference on Data Mining*, pages 1065–1070, 2022a.

C. Ling, J. Jiang, J. Wang, and L. Zhao. Source localization of graph diffusion via variational autoencoders for graph inverse problems. In *Proc. of the KDD*, 2022b.

C. Ling, J. Jiang, J. Wang, M. T. Thai, R. Xue, J. Song, M. Qiu, and L. Zhao. Deep graph representation learning and optimization for influence maximization. In *International Conference on Machine Learning*, pages 21350–21361. PMLR, 2023.

S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems*, 33:20000–20011, 2020.

D. T. Nguyen, H. Zhang, S. Das, M. T. Thai, and T. N. Dinh. Least cost influence in multiplex social networks: Model representation and analysis. In *2013 IEEE 13th International Conference on Data Mining*, pages 567–576, 2013. doi: 10.1109/ICDM.2013.24.

H. T. Nguyen, M. T. Thai, and T. N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proc. of the SIGMOD*, 2016.

Y. Shen, T. Dinh, H. Zhang, and M. Thai. Interest-matching information propagation in multiple online social networks. *ACM International Conference Proceeding Series*, 10 2012. doi: 10.1145/2396761.2398525.

S. S. Singh, K. Singh, A. Kumar, and B. Biswas. Mim2: Multiple influence maximization across multiple social networks. *Physica A: Statistical Mechanics and its Applications*, 526:120902, 2019.

C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. Biogrid: A general repository for interaction datasets. *Nucleic acids research*, 34:D535–9, 01 2006. doi: 10.1093/nar/gkj109.

J. Tang, X. Tang, X. Xiao, and J. Yuan. Online processing algorithms for influence maximization. In *Proc. of the SIGMOD*, pages 991–1005, 2018.

Y. Tang, X. Xiao, and Y. Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proc. of the SIGMOD*, pages 75–86, 2014.

Y. Tang, Y. Shi, and X. Xiao. Influence maximization in near-linear time: A martingale approach. In *Proc. of the SIGMOD*, 2015.

Q. Zhan, J. Zhang, S. Wang, P. S. Yu, and J. Xie. Influence maximization across partially aligned heterogenous social networks. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 58–69. Springer, 2015.

H. Zhang, D. T. Nguyen, H. Zhang, and M. T. Thai. Least cost influence maximization across multiple social networks. *IEEE/ACM Transactions on Networking*, 24(2):929–939, apr 2016. doi: 10.1109/tnet.2015.2394793.

C. Zhou, P. Zhang, W. Zang, and L. Guo. On the upper bounds of spread for greedy algorithms in social network influence maximization. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2770–2783, 2015. doi: 10.1109/TKDE.2015.2419659.

J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun. Graph neural networks: A review of methods and applications. *AI Open*, 2018. doi: 10.1016/J.AIOPEN.2021.01.001.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes, except for error statistics in the tables. The reason is the space limit.]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Yes]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Yes]
   (d) Information about consent from data providers/curators. [Yes]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Yes]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

MIM-Reasoner: Learning with Theoretical Guarantees for Multiplex Influence Maximization

A. Detail Steps of MIM-Reasoner

Algorithm 1: Budget Allocation (Phase 1)

Input: Algorithm $\mathcal{A}$, a multiplex network $\mathscr{G} = (G^1, G^2, \ldots, G^k)$, budget l
Output: Budget allocation table $\mathcal{U}$

Initialize $\mathcal{S} := \emptyset$
foreach $G^i \in \mathscr{G}$ **do**
  $S_{ij} :=$ `runAlgorithm(`$\mathcal{A}$, G^i, l`)` `// Run any algorithm A on each layer`
  $\mathcal{S} := \mathcal{S} \cup S_{ij}, \forall j \in [1, ..., l]$ `// Collect found solution`
$C := \emptyset; P := \emptyset$
foreach $S_{ij} \in \mathcal{S}$ **do**
  $p := \sigma(S_{ij}); c := |S_{ij}|$ `// Calculate the profit and cost`
  $P := P \cup [p]; C := C \cup [c]$ `// Store the profit and cost`
$\mathcal{H} := \mathcal{H}[P][C]$ `// Create a Profit-Cost table with profits` P `as rows and costs` C `as columns`
$\mathcal{U} :=$ MCKP-Solver($\mathcal{H}$) `// Solve the MCKP to obtain budget allocation` $\mathcal{U}$
return $\mathcal{U}$;

Algorithm 2: Relation RL Optimization (Phase 2)

Input: Budget Allocation table $\mathcal{U}$
Output: RL model π

Initialize $\mathcal{G}^{prev} := \emptyset$, $\mathcal{D} := \emptyset$
for $i \in 1, \ldots, k$ **do**
  Select layer $G_i = \arg\min_{q \in 1,\ldots,k} \mathcal{U}[q]$ `// Select layer with minimum budget`
  Remove row i^{th} in table $\mathcal{U}$ `// Prevent reselection`
  $j := U[i][0]$ `// Select budget for layer` G_i
  if $\mathcal{D} = \emptyset$ **then**
    Collect seed set S_{ij} from phase 1 `// Collect seed set from phase 1`
    Train policy π to learn seed set S_{ij} and record $\mathcal{D}$ `// Train policy with seed set`
  else
    Train π with budget j based on $\mathcal{G}^{prev}$ and record a new $\mathcal{D}$ `// Train policy with rewards generated from` $\mathcal{G}^{prev}$
  $\mathcal{Y} :=$ VariableGrouping($\mathcal{D}$) `// Perform variable grouping`
  $\varphi_i :=$ StructureLearning($\mathcal{Y}$) `// Training PGM`
  $\mathcal{G}^{prev} := \mathcal{G}^{prev} \cup \varphi_i$ `// Update previous structure set`
return π

A.1. Good Candidate Finding

Even when dividing the multiplex network into separate layers, each layer $G_i \in \mathscr{G}$ can still be at a billion-scale. The objective of Good Candidate Finding is to identify nodes that are unlikely to be part of the solution set, thereby reducing the search space. This is possible because although the graph itself can change, the underlying

Algorithm 3: The Probabilistic Greedy Approach

Input: Graph $\hat{G}_{i,z} = (V_z, E_z)$, propagation function $\sigma_i(.)$ of layer G_i, convergence threshold Δ
Output: Solution set S_z^q, $|S_z^q| = b$

$S_z^q \leftarrow \emptyset$ // Initialize solution set
while $\digamma > \Delta$ **do**
  $v \leftarrow$ Choose with probability $\frac{\sigma_i(S_z^q \cup v) - \sigma_i(S_z^q)}{\sum_{\forall v' \in V \setminus S} \sigma_i(S_z^q \cup \{v'\}) - \sigma_i(S_z^q)}$ // Choose vertex probabilistically
  $\digamma \leftarrow f(S_z^q \cup v) - f(S_z^q)$ // Calculate gain
  $S_z^q \leftarrow S_z^q \cup v$ // Add selected vertex to the solution set
return S_z^q // Return solution set

model generating the graph often remains consistent. Let $\mathcal{E}_i = [\hat{G}_{i,1}, ..., \hat{G}_{i,h}]$ be a set of subgraphs induced from the i^{th} layer of $\mathscr{G}$, where h is a hyperparameter representing the total number of subgraphs the trainer wants to sample. For each layer $G_i \in \mathscr{G}$, we sample a set of subgraphs $\mathcal{E}_i$, hence, our graph training dataset becomes $\mathcal{E} = [\mathcal{E}_1, \mathcal{E}_2, \cdots, \mathcal{E}_k]$. We utilize a classification-based method to train the Graph Attention Network (GAT) model, denoted as $\mathcal{I}$. In this method, given any training graph $\hat{G}_{i,z} \in \mathcal{E}_i$, a budget b, and its corresponding solution set S_z, a node v is labeled as positive if $v \in S_z$, and negative otherwise.

In certain cases, within each graph $\hat{G}_{i,z} = (V_z, E_z)$, there can be situations where two nodes $(v_1, v_2) \in V_z$ have the same influence spread (i.e., $\sigma_i(v_1) = \sigma_i(v_2)$). However, when considering the marginal gain of adding node v_2 to the solution set $S_z = v_1$, denoted as $\sigma_i((v_1, v_2)) - \sigma_i((v_1))$, it turns out to be zero and vice versa. Consequently, in such scenarios, even though both nodes have equal and individual quality, only one of them would be selected in the final answer set meaning only one of the two nodes is considered as a positive candidate. That is the reason why we need to use a Probabilistic Greedy strategy to ensure that both v_1 and v_2 are considered positive candidates (good candidates).

Probabilistic greedy: For each graph $\hat{G}_{i,z} \in \mathcal{E}_i$, we employ a probabilistic greedy approach that involves sampling a node from the search space V_z while considering marginal gain of the sampled node. Specifically, instead of always selecting the node $v \in V_z$ with the highest marginal gain, we choose v in a probabilistic manner where v is chosen with a probability that is proportional to its marginal gain (Algorithm 3). During each iteration, the probabilistic greedy algorithm performs $\varkappa$ times to create $\varkappa$ different solution sets $\mathbb{S}_z = \{S_z^1, \cdots, S_z^\varkappa\}$. To determine the score of a node $v \in V_z$, the algorithm assigns a value based on the following process:

$$\omega(v) = \frac{\sum_{q=1}^{\varkappa} \digamma_q(v)}{\sum_{q=1}^{\varkappa} \sigma_i(S_z^q)} \tag{8}$$

Here, S_z^q is the q^{th} seed set of graph $\hat{G}_{i,z}$ while $\digamma_q(v)$ is the marginal gain contribution of v to S_z^q. Suppose we have a seed set $S_z^q = \{1, 3, 5, 4\}$, the marginal gain contribution of node $v = 3$ can be computed as $\digamma_q(3) = \sigma_i((1, 3)) - \sigma_i((1))$. Based on the set $\mathbb{S}_z$, we can have a set of good nodes $V_z^g = \bigcup_{q=1}^{\varkappa} S_z^q$. For each good node $v \in V_z^g$, we want $\mathcal{I}$ with parameters Θ predict $\hat{\omega}(v)$ as close as much as possible with the found $\omega(v)$. Therefore, we use Mean Square Error (MSE) as the objective function as shown here:

$$J(\Theta) = \sum_{\sim \langle G_{i,z} \rangle} \frac{1}{|V_z^g|} \sum_{\forall v \in V_z^g} [\omega(v) - \hat{\omega}(v)]^2 \tag{9}$$

In the above equation, V_z^g denotes the set of good nodes in graph G_i. Since $\mathcal{I}$ is trained through message passing, in a $\mathcal{I}$ with K hidden layers, the computation of each node v is limited to the induced subgraph formed by its K-hop neighbors, instead of the entire graph.

A.2. Variables Grouping

Recall the fact that the number of nodes in the Status dataset $\mathcal{D}$ can be at billion-scale, it is necessary to reduce complexity before training a Probabilistic Graphical Model through Structure Learning. One preprocessing step that achieves this is Variable Grouping. Variable Grouping involves grouping highly correlated nodes in the Status dataset $\mathcal{D} = [0, 1]^{m \times |V|}$ together, resulting in a smaller set $\mathcal{P} = [P_1, P_2, \ldots, P_q]$ where q is the total

Algorithm 4: Variable Grouping

Input: Graph $\hat{G}_{i,z} = (V_z, E_z)$, Status dataset $\mathcal{D}$, correlation threshold ξ
Output: Set of representative nodes $\mathcal{Y}$

$\mathcal{P} := \emptyset$; // Initialize an empty set of variable groups
$\mathcal{Y} := \emptyset$; // Initialize an empty set of representative nodes
$Q =$ PearsonCorrelation($\mathcal{D}$) // Calculate Pearson Correlation for all node in $\mathcal{D}$
Initialize an array *isolated* having $|V_z|$ elements, and set each element to True
// To check whether a node has any group
$S =$ NaN-Grouping(Q) // Group nodes with constant status together to create a new set of nodes with changing status
foreach $x \in S$ **do**
  if *isolated[x]=True* **then**
    Create a new group $P_i \in \mathcal{P}$;
    $P_i := P_i \cup x$; // Add x to P_i
    $\mathcal{P} := \mathcal{P} \cup P_i$; // Add P_i to **P**
    foreach $y \in S$ **do**
      if *isolated[y]=True and* $Q[x, y] > \xi$ *and* $Q[x, y] \neq$ "NaN" **then**
        $P_i := P_i \cup y$; // Add y to P_i
    Find the representative node v_i in P_i by selecting the node closest to the centroid of P_i
    $\mathcal{Y} := \mathcal{Y} \cup v_i$
    // Add representative node v_i for group P_i to $\mathcal{Y}$
return $\mathcal{Y}$;

number of groups. This grouping satisfies the property that $V = \bigcup_{i=1}^{q} P_i$. The underlying idea is that when nodes are highly correlated, they often share similar characteristics or behaviors. For example, let's consider two nodes (v_1, v_2) in dataset $\mathcal{D}$. In a Monte Carlo Simulation step with m iterations, it is observed that when v_1 is activated, there is a 70 % percentage that v_2 is also activated. This means that v_1 and v_2 can have high correlations and similar patterns of activation. By knowing only a representative node, we can infer the properties of other nodes in the cluster without explicitly including them in the PGM, hence, reducing the complexity. In this work, we use Pearson Correlation to measure the correlation between two variables.

Pearson Correlation metric: Pearson correlation is a statistical measure that quantifies the linear relationship between two variables. It is used to assess the strength and direction of the relationship between two variables and it is denoted as $\chi \in [-1, 1]$. A positive value of χ indicates a positive linear relationship, meaning that as one variable increases, the other tends to increase as well. Meanwhile, a negative value of χ indicates a negative linear relationship, where as one variable increases, the other tends to decrease. Lastly, $\chi = 0$ indicates no linear relationship between the variables. Given two variables x and y and paired data $\{(x_1, y_1), \ldots, (x_m, y_m)\}$ consisting of m pairs, the Pearson correlation coefficient is computed as follows:

$$\chi_{xy} = \frac{\sum_{i=1}^{m} (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{m} (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^{m} (y_i - \bar{y})^2}} \tag{10}$$

Here, m represents the sample size, x_i and y_i represent the individual sample points indexed with i, $\bar{x} = \frac{1}{m} \sum_{i=1}^{m} x_i$ represents the sample mean of x, and $\bar{y}$ represents the sample mean of y.

We base on the Pearson correlation coefficient defined above to calculate a correlation coefficient for all pairs of variables (nodes) in dataset $\mathcal{D}$ to obtain the correlation matrix $Q^{|V| \times |V|}$ where $Q_{x,y} = \chi_{xy} | 1 \leq x, y \leq |V|$. We then define a threshold ξ and traverse the correlation matrix Q. Any two variables with a correlation coefficient higher than ξ are considered highly correlated and should be grouped (Algorithm 2). An important thing to note is that there will be nodes that never change status, such as always active nodes or always de-active nodes. In that case, the correlation of those nodes to any remaining nodes will become "NaN" because their standard deviation (the denominator in χ_{xy} equation) becomes zero. Those nodes will be grouped with other nodes $v \in V$. After grouping nodes in dataset $\mathcal{D}$, we will have a set of highly correlated node groups $\mathcal{P} = [P_1, P_2, \ldots, P_q]$. The

next step is to define a representative node for each group $P_i \in \mathcal{P}$. In this work, we take the node closest to the centroid from each group as a representative node. The set of representative nodes is denoted as $\mathcal{Y} = [v_1, \ldots, v_q]$, where each representative node $v_i \in \mathcal{Y}$ corresponds to a group $P_i \in \mathcal{P}$.

A.3. Structure Learning

Structure learning in Probabilistic Graphical Models (PGMs) involves automatically discovering the optimal graphical structure that represents the relationships between random variables in a dataset. It helps infer conditional independence relationships, identify direct and causal dependencies, and understand the underlying causal mechanisms. This is crucial for tasks like prediction, inference, and gaining insights into the data.

After the Variables Grouping step, we have an updated set of representative variables $\mathcal{Y} = [v_1, \ldots, v_q]$, status dataset $\mathcal{D}$ which will be used to generate the structure of our Bayesian model. In this work, we employ the Tree-based algorithm called Chow Liu to learn structure because it offers a balance between complexity and representation power. Given any random directed tree, an approximation of the true joint probability distribution of $\mathcal{Y}$ is in the form as follows:

$$P_t(\mathcal{Y}) = \prod_{i=1}^{q} P\left(v_i \mid v_{\Gamma(i)}\right) \tag{11}$$

Here, $\Gamma(i)$ is the parent of node v_i. If a vertex i is the root, its parent $\Gamma(i) = \emptyset$, and the conditional probability $P\left(v_i \mid v_{\Gamma(i)}\right)$ simplifies to $P(v_i)$. It is important to note that in a directed tree, we must first select a vertex as the root. The edges are oriented away from the root, resulting in each vertex having at most one parent (but potentially multiple children).

Problem Definition: Let $P(\mathcal{Y})$ be the true joint probability distribution of $\mathcal{Y}$. Given a set of representative variables $\mathcal{Y} = [v_1, \ldots, v_q]$ and the set of all possible first-order dependence trees denoted as T_q, we want to find the optimal first-order dependence tree or the optimal structure denoted as Ψ such as $\mathrm{KL}\left(P, P_\Psi\right) \leq \mathrm{KL}\left(P, P_t\right)$ for all $t \in T_q$.

Chow Liu algorithm solves this Minimization Problem by searching a maximum weight spanning tree (MWST). Lets define the mutual information $I\left(v_i, v_j\right) = \sum_{v_i, v_j} P\left(v_i, v_j\right) \log\left(\frac{P(v_i, v_j)}{P(v_i)P(v_j)}\right)$ between two variables v_i and v_j. The key insight of MWST is that a probability distribution of tree dependence $P_t(\mathcal{Y})$ is an optimum approximation to $P(\mathcal{Y})$ if its tree model has maximum branch weight $\sum_{i=1}^{q} I\left(v_i, v_{\Gamma(i)}\right)$. Mathematically, we have:

$$\begin{aligned}
\mathrm{KL}\left(P, P_t\right) &= \sum_{\mathcal{Y}} P(\mathcal{Y}) \log P(\mathcal{Y}) - \sum_{\mathcal{Y}} P(\mathcal{Y}) \sum_{i=1}^{q} \log P\left(v_i \mid v_{\Gamma(i)}\right) \\
&= \sum_{\mathcal{Y}} P(\mathcal{Y}) \log P(\mathcal{Y}) - \sum_{\mathcal{Y}} P(\mathcal{Y}) \sum_{i=1, \neq \text{ root}}^{q} \log \frac{P\left(v_i, v_{\Gamma(i)}\right)}{P\left(v_{\Gamma(i)}\right)} \\
&= \sum_{\mathcal{Y}} P(\mathcal{Y}) \log P(\mathcal{Y}) - \sum_{\mathcal{Y}} P(\mathcal{Y}) \sum_{i=1, \neq \text{ root}}^{q} \log \frac{P\left(v_i, v_{\Gamma(i)}\right)}{P\left(v_i\right) P\left(v_{\Gamma(i)}\right)} - \sum_{\mathcal{Y}} P(\mathcal{Y}) \sum_{i=1}^{q} \log P\left(v_i\right)
\end{aligned} \tag{12}$$

Moreover, we should note that $-\sum_{\mathcal{Y}} P(\mathcal{Y}) \log P\left(v_i\right) = -\sum_{v_i} P\left(v_i\right) \log P\left(v_i\right)$. To understand why we have this, suppose $\mathcal{Y} = (v_1, v_2)$, let all variables are binary, let $i = 1$

$$\begin{aligned}
&-\sum_{\mathcal{Y}} P(\mathcal{Y}) \log P\left(v_i\right) \\
&= -\left[P\left(v_1 = 0, v_2 = 0\right) \log P\left(v_1 = 0\right) + P\left(v_1 = 0, v_2 = 1\right) \log P\left(v_1 = 0\right) + \right. \\
&\quad \left. P\left(v_1 = 1, v_2 = 0\right) \log P\left(v_1 = 1\right) + P\left(v_1 = 1, v_2 = 1\right) \log P\left(v_1 = 1\right)\right] \\
&= -\left[P\left(v_1 = 0\right) \log P\left(v_1 = 0\right) + P\left(v_1 = 1\right) \log P\left(v_1 = 1\right)\right] \\
&= -\sum_{v_1} P\left(v_1\right) \log P\left(v_1\right) = -\sum_{v_i} P\left(v_i\right) \log P\left(v_i\right)
\end{aligned} \tag{13}$$

We can also see that $-\sum_{\mathcal{Y}} P(\mathcal{Y}) \log P(\mathcal{Y})$, $-\sum_{v_i} P(v_i) \log P(v_i)$ are entropy terms $H(\mathcal{Y})$, $H(v_i)$, respectively. Then, we can rewrite equation 12 as:

$$\mathrm{KL}(P, P_t) = -\sum_{\mathcal{Y}} P(\mathcal{Y}) \sum_{i=1, \neq \text{ root}}^{q} \log \frac{P\left(v_i, v_{\Gamma(i)}\right)}{P(v_i) P\left(v_{\Gamma(i)}\right)} + \sum_{i=1}^{q} H(v_i) - H(\mathcal{Y})$$

Moreover, we also have:

$$\begin{aligned} \sum_{\mathcal{Y}} P(\mathcal{Y}) \log \frac{P\left(v_i, v_{\Gamma(i)}\right)}{P(v_i) P\left(v_{\Gamma(i)}\right)} &= \sum_{v_i, v_{\Gamma(i)}} P\left(v_i, v_{\Gamma(i)}\right) \log \frac{P\left(v_i, v_{\Gamma(i)}\right)}{P(v_i) P\left(v_{\Gamma(i)}\right)} \\ &= I\left(v_i, v_{\Gamma(i)}\right) \end{aligned} \tag{14}$$

Replace this result of equation 14 to equation 12, we have:

$$\mathrm{KL}(P, P_t) = -\sum_{i=1}^{q} \underbrace{I\left(v_i, v_{\Gamma(i)}\right)}_{\text{Mutual information}} + \underbrace{\sum_{i=1}^{q} H(v_i) - H(\mathcal{Y})}_{\substack{\text{Independent of the} \\ \text{dependence tree}}} \tag{15}$$

Minimizing $\mathrm{KL}(P, P_t)$ is the same as maximizing the total branch weight $\sum_{i=1}^{n} I\left(v_i, v_{\Gamma(i)}\right)$. With this insight, Chow Liu uses Kruskal's algorithm to construct a maximum weight-spanning tree as a structure for PGM.

B. Proofs and Supplementary Lemmas

B.1. Preliminaries

Definition 3 *(Submodular).* For all $A, B \subseteq V_i$, we have:

$$\sigma_i(A) + \sigma_i(B) \geq \sigma_i(A \cup B) + \sigma_i(A \cap B) \tag{16}$$

Definition 4 *(Monotone Increasing).* An objective function $\sigma(S)$ is monotone increasing if

$$\sigma(S) \leq \sigma(T), S \subset T \tag{17}$$

Definition 5 (Diminishing Returns). An objective function $\sigma(S)$ is diminishing return if $\sigma(S \cup u) - \sigma(S) \geq \sigma(T \cup u) - \sigma(T)$, $\forall u \in T$ and $S \subset T$.

Definition 6 *(Generalized Deterministic Submodular).* Let σ be a model of influence propagation on the multiplex. σ satisfies the Generalized Deterministic Submodular Property (GDS) if the expected number of activations, given a seed set S, can be expressed as:

$$\sigma(S) = \sum_{i=1}^{k} p_i \sigma_i(S), \tag{18}$$

where each σ_i with $i \in \{1, \ldots, k\}$ is a deterministic submodular model of influence propagation, and $p_i \in [0, 1], \sum_{i=1}^{k} p_i = 1$.

Definition 7 (Multiple-choice knapsack problem (MCKP)). Let $(\mathscr{C}, k, l, c, p, B)$ be given. Here a set of k classes, $\mathscr{C} = C_1, \ldots, C_k$, where each class C_i consists of l objects denoted as $C_i = x_{i1}, \ldots, x_{il}$, and a budget $B \geq 0$ the goal is to select one item, x'_i, from each class in a way that maximizes the total profit, $\sum_{i=1}^{k} p(x'_i)$ while ensuring that the total cost, $\sum_{i=1}^{k} c(x'_i)$, does not exceed the budget constraint $\sum_{i=1}^{k} c(x'_i) < B$. Here, c and p represent the cost and profit functions associated with the objects x_{ij}, respectively.

B.2. Greedy Policy Convergence

Lemma 1 (Greedy Policy Convergence). When policy π is converged to optimal, $\pi^*(v \mid S_t)$ always selects nodes greedily at every time step t.

Proof. Recall the fact that our reward function to train policy π is computed by considering the marginal gain of adding node v_n to the partially computed solution S_t, as follows:

$$r_t = \mathcal{M}\left(S_{i,t} \cup v_n\right) - \mathcal{M}\left(S_{i,t}\right) \tag{19}$$

Here $\mathcal{M}$ is customized evaluation function which measures the total spread in layer G_i, given solution S_i. When policy π parameterized by θ is converged to optimal, the optimal policy π^* will follow these parameters:

$$\theta^* = \arg\max_{\theta} \sum_{t=1}^{\mathcal{Q}} \mathbb{E}_{(S_{i,t},v_n)\sim p(S_{i,t},v_n|\theta)} \left[\mathcal{M}\left(S_{i,t} \cup v_n\right) - \mathcal{M}\left(S_{i,t}\right)\right] \tag{20}$$

Considering these optimal parameters, we can use contradiction to prove this lemma. For any time step t progresses from 1 to $\mathcal{Q}$, if θ^* is truly optimal, then it must ensure that at every step t, the agent is making a decision to maximize expected immediate reward, which is to pick the node providing the highest marginal gain. If there was a better node v_n' that the agent didn't select at some step t, then $\mathbb{E}_{(S_{i,t},v_n)\sim p(S_{i,t},v_n|\theta)} \left[\mathcal{M}\left(S_{i,t} \cup v_n\right) - \mathcal{M}\left(S_{i,t}\right)\right]$ wouldn't be maximized, contradicting our assumption that θ^* is optimal. Hence, this proves the lemma and we conclude that the optimal policy $\pi^*\left(v \mid S_t\right)$ will select nodes greedily for all t.

B.3. Multiple Choice Knapsack Problem Guarantee

Lemma 2.(Multiple Choice Knapsack Problem Guarantee). Let $Opt_{S_{ij}}$ be the value of the solution for MCKP instance S_{ij}, and $Opt_{\tilde{S}_{ij}}$ be the value of the optimal solution for $\tilde{S}_{ij}$, we have:

$$Opt_{S_{ij}} \geq (1 - 1/e) Opt_{\tilde{S}_{ij}} \tag{21}$$

Proof. Since in Phase 1, we decompose MIM into separate layers, find a solution for each layer, then, combine back using MCKP. We need to examine how the MCKP problem can be represented for the MIM problem. From there, we can determine the approximation ratio of the solution combined with MCKP in the context of MIM.

Given a MIM instance $(\mathscr{G}, k, l, \sigma)$, where $\mathscr{G}$ represents the graph, k is the number of seed sets, and l is the size of each seed set. For each pair (i, j), where $1 \leq i \leq k$ and $1 \leq j \leq l$, let $\tilde{S}_{ij}$ be an unknown optimal seed set for G_i that satisfies two conditions: $\tilde{S}_{ij} \subset V$ (subset of nodes in G_i) and $|\tilde{S}_{ij}| = j$ (size of the seed set). In addtition, let S_{ij} found by an algorithm $\mathcal{A}$ be an approximation to $\tilde{S}_{ij}$ and $S_{ij} \subset V$. Then, based on **Definition 7**, $C_i = \{S_{i0}, \ldots, S_{il}\}, C_i^{opt} = \left\{\tilde{S}_{i0}^{opt}, \ldots, \tilde{S}_{il}^{opt}\right\}$. Finally, let $\mathscr{C} = \{C_i : 1 \leq i \leq k\}, \mathscr{C}^{\text{opt}} = \left\{C_i^{opt} : 1 \leq i \leq k\right\}$, and for each i, j, define $c\left(S_{ij}\right) = j, p\left(T_{ij}\right) = \sigma\left(S_{ij}\right)$, and likewise define c^{opt}, p^{opt} for each $\tilde{S}_{ij}$. Thus, we have two instances of the knapsack problem, namely $I_1 = (\mathscr{C}, k, l, c, p, l)$ and $I_2 = (\mathscr{C}^{opt}, k, l, c^{\text{opt}}, p^{opt}, l)$.

For convenience, let's assume that j represents the budget that should be spent for layer G_i, as determined by any MCKP solver. In that case, S_{ij} and $\tilde{S}_{ij}$ can be used to represent the output, where S_{ij} represents the approximate solution and $\tilde{S}_{ij}$ represents the unknown optimal solution. Since, our algorithm $\mathcal{A}$ is greedy-style, each element of C_i has approximation ratio $(1-1/e)$. Thus, the value of the approximate solution $Otp_{S_{ij}}$ decided by any MCKP solver for each layer G_i also has $(1 - 1/e)$ as follow:

$$\begin{aligned} (1 - 1/e) Opt_{\tilde{S}_{ij}} &= (1 - 1/e)\sigma\left(\tilde{S}_{ij}\right) \\ &\leq \sigma\left(S_{ij}\right) \\ &\leq \sigma\left(S_{ij}\right) \\ &\leq Opt_{S_{ij}} \end{aligned} \tag{22}$$

From there, we can also extend the analysis to get an approximation ratio for $\mathscr{G}$. Let, $S = \bigcup_{i=1}^{k} S_{ij}$ and $\tilde{S} = \bigcup_{i=1}^{k} \tilde{S}_{ij}$ represent the final solutions for MCKP instances I_1 and I_2, respectively. We have:

$$\begin{aligned}(1-1/e)Opt_{\tilde{S}} &= (1-1/e)\sum_{i=1}^{k}\sigma\left(\tilde{S}_{ij}\right)\\ &\leq \sum_{i}\sigma\left(S_{ij}\right)\\ &\leq \sum_{i}\sigma\left(S_{ij}\right)\\ &\leq Opt_S\end{aligned} \tag{23}$$

Thus, proving the lemma.

B.4. Approximation Ratio In The Worst Case

Theorem 1 (Approximation Ratio In The Worst Case) Suppose the propagation σ_i on each layer of the multiplex is submodular, the optimal policy π^* will find a solution $\hat{S}$ for multiplex network $\mathscr{G}$ with an approximation ratio of $\frac{(1-\epsilon)(1-1/e)}{(o+1)k}$.

Proof. We start by assuming that the model σ_i on each layer G_i satisfies the Generalized Deterministic Submodular (GDS), as shown in Definition 6. If all σ_i of all G_i satisfy the Generalized Deterministic Submodular (GDS) property, then σ is submodular (Kuhnle et al., 2018).

Let $\hat{S} = \bigcup_{i=1}^{k} S_i$ be the solution returned by the MIM-Reasoner for the multiplex network $\mathscr{G}$, where S_i represents the seed set found on each layer G_i using policy π^*. Let $\tilde{S}$ be the unknown optimal solution for the multiplex network $\mathscr{G}$, and $\sigma(\hat{S})^i$ be the expected activation under σ in layer G_i. Note that, $\sigma(\hat{S})^i$ only counts duplicated nodes as a single instance within layer G_i, rather than adding up all instances. Based on these definitions, we can state the following inequality:

$$\sigma\left(\tilde{S}\right) \leq \sum_{i=1}^{k}\sigma\left(\tilde{S}\right)^i \tag{24}$$

From this, we define O as the set of native overlapping nodes. This set excludes isolated vertices that are added by other layers. Given the set O of native overlapping nodes, it is established that:

$$\sigma\left(\tilde{S}\right)^i \leq \sigma_i\left(\tilde{S}\cup O\right) \tag{25}$$

Equation 25 arises because $\sigma(\tilde{S})^i$ considers duplicated nodes as a single instance, while $\sigma_i(\tilde{S}\cup O)$ counts all duplicated nodes. Moreover, any node in G_i can be activated by the model σ_i receiving seed nodes in $\tilde{S}\cap G_i$ or native overlapping nodes O. In addition, because of submodularity of σ_i, we have:

$$\sigma_i\left(\tilde{S}\cup O\right) \leq \sigma_i\left(\tilde{S}\right) + \sigma_i(O) \tag{26}$$

Recall that in Lemma 2, Opt_S, $Opt_{\tilde{S}}$ denotes the value of MCKP on instance I_1 and I_2, respectively. We also know that π^* will select node greedily based on Lemma 1. Let $\sigma(\hat{S}^{\pi^*})$ denote the total spreading in multiplex network $\mathscr{G}$ of the solution $\hat{S}$ sampled by optimal policy π^*. In phase two, solution S obtained from phase one is tuned by optimal policy π^* to become $\hat{S}^{\pi^*}$, then:

$$\sigma(\hat{S}^{\pi^*}) \geq \sigma(S) \geq (1-\epsilon)Opt_S \tag{27}$$

Here, in the context of this work, $(1-\epsilon)$ with $\epsilon > 0$ represents the approximation ratio for the nearly exact solution found by the MCKP solver (Chandra et al., 1976) that we employ. In addition, for any set S (could be optimal or just feasible) with size at most l and any fixed layer i, we base on Lemma 2 to have :

$$\frac{1}{(1-\epsilon)(1-1/e)}\sigma(\hat{S}^{\pi^*}) \geq Opt_{\tilde{S}} \geq \sigma_i(S) \tag{28}$$

By Lemma 2, and since $\sigma_i(S)$ is the value of a feasible solution to the multiplex network instance in layer G_i, we have the above inequality. Combining the inequalities, we get:

$$\begin{aligned}
\sigma\left(\tilde{S}\right) &\leq \sum_{i=1}^{k} \sigma\left(\tilde{S}\right)^i \\
&\leq \sum_{i=1}^{k} \sigma_i\left(\tilde{S}\right) + \sum_{i=1}^{k} \sigma_i(O) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)} \sigma(\hat{S}^{\pi^*}) + \sum_{i=1}^{k} \sigma_i(O) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)} \sigma(\hat{S}^{\pi^*}) + \sum_{v \in O} \sum_{i=1}^{k} \sigma_i(v) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)} \sigma(\hat{S}^{\pi^*}) + \frac{ok}{(1-\epsilon)(1-1/e)} \sigma(\hat{S}^{\pi^*}) \\
&\leq \frac{(o+1)k}{(1-\epsilon)(1-1/e)} \sigma(\hat{S}^{\pi^*}).
\end{aligned} \tag{29}$$

We can rewrite this inequality as:

$$\sigma(\hat{S}^{\pi^*}) \geq \frac{(1-\epsilon)(1-1/e)}{(o+1)k} \sigma\left(\tilde{S}\right) \tag{30}$$

Thus, MIM-Reasoner has approximation ratio $\frac{(1-\epsilon)(1-1/e)}{(o+1)k}$.

B.5. Approximation Ratio In The Best Case

Theorem 2 (Accuracy Gap In The Best Case). Assume the π^* can avoid reactivating all the activated nodes, the spread of solution given by optimal policy π^* is at least: $\sigma\left(\hat{S}^{\pi^*}\right) \geq \frac{(1-\epsilon)(1-1/e)}{k+o} \sigma(\tilde{S})$.

Proof. Recall that the optimal policy π^* will avoid reactivating nodes that have already been activated, starting from layer 2 up to layer k. We modify the proof of Theorem 1 (Equation 29) to have:

$$\begin{aligned}
\sigma\left(\tilde{S}\right) &\leq \sum_{i=1}^{k} \sigma\left(\tilde{S}\right)^i \\
&\leq \sum_{i=1}^{k} \sigma_i\left(\tilde{S}\right) - \sum_{i=2}^{k} \sigma_i(O) + \sum_{i=1}^{k} \sigma_i(O) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)} \sigma(\hat{S}^{\pi^*}) + \sigma_i(O) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)} \sigma(\hat{S}^{\pi^*}) + \sum_{v \in O} \sigma_i(v) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)} \sigma\left(\hat{S}^{\pi^*}\right) + \frac{o}{(1-\epsilon)(1-1/e)} \sigma\left(\hat{S}^{\pi^*}\right) \\
&\leq \frac{k+o}{(1-\epsilon)(1-1/e)} \sigma\left(\hat{S}^{\pi^*}\right)
\end{aligned} \tag{31}$$

We can rewrite this inequality as:

$$\sigma(\hat{S}^{\pi^*}) \geq \frac{(1-\epsilon)(1-1/e)}{k+o} \sigma\left(\tilde{S}\right) \tag{32}$$

Therefore, the theorem is proved.

B.6. Approximation Ratio In The General Case

Theorem 3 (Accuracy Gap In General Case). Assume the π^* can avoid reactivating the activated nodes by other layers partially. Thus, with $\beta \in [0,1]$, the spread of solution given by optimal policy π^* is at least: $\sigma\left(\hat{S}^{\pi^*}\right) \geq \frac{(1-\epsilon)(1-1/e)}{(k-1)\beta o+o+k}\sigma(\tilde{S})$.

Proof. Given $\beta \in [0,1]$ representing the percentage of nodes that cannot be successfully avoided reactivation by the policy π^*, we have:

$$\begin{aligned}
\sigma\left(\tilde{S}\right) &\leq \sum_{i=1}^{k} \sigma\left(\tilde{S}\right)^i \\
&\leq \sum_{i=1}^{k} \sigma_i\left(\tilde{S}\right) - (1-\beta)\sum_{i=2}^{k}\sigma_i(O) + \sum_{i=1}^{k}\sigma_i(O) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)}\sigma(\hat{S}^{\pi^*}) + \sigma_i(O) - (1-\beta)\sum_{i=2}^{k}\sigma_i(O) + \sum_{i=2}^{k}\sigma_i(O) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)}\sigma(\hat{S}^{\pi^*}) + \sum_{v\in O}\sigma_i(v) + \beta\sum_{v\in O}\sum_{i=2}^{k}\sigma_i(v) \\
&\leq \frac{k}{(1-\epsilon)(1-1/e)}\sigma\left(\hat{S}^{\pi^*}\right) + \frac{o}{(1-\epsilon)(1-1/e)}\sigma\left(\hat{S}^{\pi^*}\right) + \frac{o(k-1)\beta}{(1-\epsilon)(1-1/e)}\sigma(\hat{S}^{\pi^*}) \\
&\leq \frac{k+o+o(k-1)\beta}{(1-\epsilon)(1-1/e)}\sigma\left(\hat{S}^{\pi^*}\right)
\end{aligned} \tag{33}$$

Thus, proving the theorem.

B.7. Time complexity of structure learning for PGMs

Lemma 3 (PGMs's Time complexity). The time complexity of structure learning for $\mathcal{G}^{\text{prev}}$ after k selection step is $|\mathcal{Y}|^2 \cdot (k-1)$.

Proof. In the Structure Learning step, given a set of representative nodes $\mathcal{Y} = [v_1, \ldots, v_q]$, we calculate pairwise mutual information for all $\frac{q(q-1)}{2}$ pairs and employ Kruskal's algorithm to construct a Minimum Weight Spanning Tree (MWST). The algorithm constructs the tree one edge at a time, in decreasing order of weights. The running time of this step is $\mathcal{O}(|\mathcal{Y}|^2)$ for $|\mathcal{Y}| = q$ variables, as it needs to consider all $\frac{q(q-1)}{2}$ edges. In addition, at each layer selection (except for the final layer selection), we have to train a PGM. Therefore, the total number of PGMs after training with a multiplex network consisting of k layers will be $k-1$. Each PGM has a complexity is $\mathcal{O}(|\mathcal{Y}|^2)$ and we have $k-1$ PGMs after training process. Thus, the time complexity of structure learning after k selection step is $O(|\mathcal{Y}|^2 \cdot (k-1))$.

B.8. Time complexity of MIM-Reasoner

Theorem 4 (Time complexity of MIM-Reasoner). The time complexity of the Budget Allocation is: $\max_{h\in k} tc\left(A, G_h, l\right) + (kl)^{\lceil 1/\epsilon - 1\rceil}\log k$ and the time complexity of Relation RL Optimization is : $O\left(|\mathcal{Y}|^2 \cdot (k-1) + \mathcal{Q}\right)$ where $\mathcal{Q}$ is number of step for policy π converge to optimal.

Proof. Given the V_i^g predicted by GAT model $\mathcal{I}(.)$ for each layer G_i, MIM-Reasoner runs algorithm A in parallel k times with the search space $V_i^g \in G_i$ and then utilizes the $(1-\epsilon)$ Multiple Choice Knapsack Problem (MCKP) solver. Let's denote the layer that takes the longest time to run as G_h. If $O\left(tc\left(A, G_h, l\right)\right)$ represents the time complexity of algorithm A on l seed nodes with graph G_h, then the time complexity of the $(1-\epsilon)$ Multiple Choice Knapsack Problem is determined by $O\left((kl)^{\lceil 1/\epsilon-1\rceil}\log k\right)$. Therefore, we have an overall time complexity for the Budget Allocation phase:

$$O\left(\max_{h\in k} tc\left(A, G_h, l\right) + (kl)^{\lceil 1/\epsilon-1\rceil}\log k\right) \tag{34}$$

In the second phase, our time complexity comes from the time complexity of PGMs, and the total number of

training step for policy π become optimal. Let denotes $\mathcal{Q}$ as a total number of training steps for policy π becomes optimal. Based on Lemma 3, we have time complexity for PGMs for multiplex with k layers is $|\mathcal{Y}|^2 \cdot (k-1)$. Thus, the time complexity for the Relation RL Optimization phase is:

$$O\left(|\mathcal{Y}|^2 \cdot (k-1) + \mathcal{Q}\right) \tag{35}$$

C EXPERIMENT DETAILS

We conducted our experiments on a machine equipped with an Intel(R) Core i9-13900k processor, 128 GB RAM, and two Nvidia RTX 4090 GPUs with 24GB VRAM each.

C.1 Experimental Analysis

Synthetic Multiplex Network. We compare MIM-Reasoner with other baselines on a synthetic multiplex network with 5,000 nodes and 25,000 edges, considering different overlapping rates and the number of layers. Each layer is a random graph generated using the Erdos-Renyi algorithm, with the overlapping percentage (30 %, 50 %, 70 %) calculated based on the layer with the highest number of nodes. In cases where the number of overlapping users exceeds the number of nodes in a layer, we create isolated nodes to maintain the correct number of overlaps. For the propagation models, we consider both the Independent Cascade (IC) model and the Linear Threshold (LT) model. In the IC model, the propagation probability for each edge in a layer is determined as 1 divided by the degree of the target node. Additionally, in the LT model, the propagation threshold for each node in a layer is randomly assigned in the range $[0.5, 0.9]$.

Real World Multiplex Network. We evaluate MIM-Reasoner and other methods on four real-world multiplex networks: 1) Celegans (Stark et al., 2006): This network consists of 6 layers with 3879 nodes and 8191 edges; 2) Drosophila (Stark et al., 2006; De Domenico et al., 2015): The Drosophila network comprises 7 layers, 8215 nodes, and 43,366 edges; 3) Pope-Election (Domenico and Altmann, 2020): The Pope-Election network includes 3 layers, with 2,064,866 nodes and 5,969,189 edges. The link of these dataset can be found in this link: https://manliodedomenico.com/data.php. Meanwhile, the Twitter-Foursquare network (Shen et al., 2012) has 2 layers, with 93269 nodes and 17,969,114 edges. It can be found in this link: "https://url1.io/s/Vd3YD".

Table 2: *The synthetic network used for evaluation consists of 5000 nodes and 25,000 edges. It is important to note that while the initial multiplex network has 5,000 nodes and 25,000 edges, the number of nodes and edges can vary depending on the overlapping user percentage or the number of layers employed. This is because, If a vertex does not exist in some other layer, we can simply add it as an isolated vertex.*

	3 Layers	4 Layers	5 Layers	6 Layers	7 Layers	8 Layers	9 Layers
Layer 1 node count	500	500	200	200	100	100	100
Layer 2 node count	2000	1000	600	400	200	200	200
Layer 3 node count	2500	1500	1000	600	400	300	300
Layer 4 node count	0	2000	1400	800	600	500	400
Layer 5 node count	0	0	1800	1200	800	600	500
Layer 6 node count	0	0	0	1800	1200	800	600
Layer 7 node count	0	0	0	0	1700	1000	700
Layer 8 node count	0	0	0	0	0	1500	800
Layer 9 node count	0	0	0	0	0	0	1400
Total Nodes	7500	8000	9000	10800	11900	12000	12600
Total Edges (30 % case)	26500	26800	27160	27700	28060	28150	28360
Total Edges (50 % case)	27500	28000	28600	29500	30100	30250	30600
Total Edges (70 % case)	28500	29200	30040	31300	32140	32350	32840

C.1.1 Training Time.

When running the algorithms on both synthetic and real-world datasets, we observed the same characteristics in terms of training time for all the methods. First, as the number of layers increases, the overall network becomes more complex with a higher number of connections (Table 1). Conversely, each layer becomes simpler. This explains the experimental results observed in methods that operate on the whole network, such as ISF, DeepIM,

and ToupleGDD, which exhibit increasing running times (training time for ML-based methods) as the number of layers increases and take longer compared to parallel algorithm operating on individual layers like KSN or MIM-Reasoner model.

In contrast to approaches that operate on the whole graph, network decomposition methods that operate on individual layers tend to exhibit decreased running time (for the CO algorithm) or training time (for ML-based methods). This reduction in time can be attributed to the fact that as the number of layers increases, each layer becomes simpler since the number of edges is divided among the layers, independent of the number of connections (edges) between overlapping users. Therefore, it is understandable that the running time of KSN or the training time of MIM-Reasoner tends to decrease as the number of layers increases, distinguishing them from other methods. Another interesting observation is related to the training of ML-based approaches. In the case of MIM-Reasoner, it requires a lightweight model with only 63,960 parameters in our setting. In contrast, ToupleGDD and DeepIM have significantly larger models with 925,090 and 9,238,780 parameters, respectively. Consequently, the training time for each optimization step in DeepIM and ToupleGDD is much slower compared to MIM-Reasoner due to the increased complexity and parameter count of their models.

C.1.2 Inference Time.

For both synthetic and real datasets, DeepIM shows faster inference time compared to other methods because it predicts end-to-end solutions directly. On the other hand, ToupleGDD, with the giant model, has to infer the seed nodes step by step, resulting in longer inference times. MIM-Reasoner has a similar inference mechanism to ToupleGDD, but it leverages batch inference to parallelize the solution inference process for each layer of the multiplex network. This approach combines the states of each layer to create a batch, allowing for efficient computation. As a result, the majority of the time is spent on the layer that requires the highest budget when making inference using MIM-Reasoner.

C.1.3 Propagation Performance.

ISF is a greedy-based algorithm with an approximation ratio of (1 - 1/e) under the GDS property, making it the most consistent algorithm among the five methods in terms of algorithm quality when tested on small-scale synthetic graphs. However, applying ISF to large real-world multiplex networks is computationally challenging due to the need for multiple propagation simulations. Meanwhile, KSN is a parallel algorithm that finds solutions for each layer independently, making it the fastest algorithm. However, it does not consider the activation of nodes in other layers through overlapping nodes, resulting in a smaller approximation ratio as the number of overlapping nodes increases. This limitation makes its solution worse compared to other methods in both synthetic and real multiplex networks.

DeepIM utilizes deep graph representation learning and optimization in continuous space to discover important seed sets in large complex graphs with high accuracy and efficiency. However, DeepIM may suffer from instability and convergence issues when dealing with more complex networks, leading to longer training times or poorer performance. In large multiplex networks such as Twitter-Foursquare or Pope Election, training DeepIM can take a very long time for the feature loss to converge. If the feature loss has not converged and only the reconstruction loss has converged, stopping the training process would result in a very poor solution. On the other hand, ToupleGDD is a well-designed solution that utilizes Deep Reinforcement Learning (DRL) for optimization problems. It has demonstrated effectiveness in experiments with both synthetic and real-world datasets. However, one limitation of ToupleGDD is its performance in large search spaces, where it can suffer from sample efficiency problems. This means that it may not consistently provide good results due to the challenges of exploring and finding optimal solutions in such an expansive large multiplex graph with limited training time steps.

Our proposed approach, MIM-Reasoner, decomposes the multiplex network into separate layers, effectively reducing the search space and observation space, even in large real-world multiplex networks such as Twitter-FourSquare or Pope-Election. This layer-wise approach allows the policy to focus on and explore the unique characteristics of each layer more efficiently, leading to improved learning and optimization within the multiplex network. Additionally, MIM-Reasoner employs Probabilistic Graphical Models (PGMs) to analyze how nodes are influenced by different layers within each layer. This utilization of PGMs enables MIM-Reasoner to find effective solutions for the MIM problem while mitigating the impact of overlapping nodes, addressing a limitation of methods like KSN. Consequently, MIM-Reasoner consistently provides good results in both synthetic and real multiplex networks. The scalability and generalizability of MIM-Reasoner make it applicable to various

Table 3: *Hyperparameters for Reinforcement Learning framework and Graph Attention Network*

Hyperparameter	Value
Learning rate for Actor Model	0.0003
Learning rate for Critic Model	0.001
Learning rate for GAT Model	0.01
Optimizer	Adam (Kingma & Ba, 2015)
Total epoch per update	8
Update time step	600
Minibatch size	256
Discount factor γ	1
Layers for GAT Model	GATConv
Layers for Actor-Critic Model	Fully Connected Layer
Activation function for GAT Model	Elu
Activation function for Actor-Critic Model	Tanh
Target network smoothing coefficient	1024
Entropy coefficient	0.01
Lambda	0.95
PPO Epsilon	0.2
Gradient Norm	0.5

multiplex network scenarios. Its ability to handle large real-world multiplex networks while still achieving good performance demonstrates its strength in solving MIM problems.

C.2 Hyperparameters and MIM-Reasoner Implementations

C.2.1 Hyperparameters For policy training, we utilize Proximal Policy Optimization (PPO) as it offers stability and ease of implementation. The hyperparameters for the Graph Attention Network (GAT) and Policy model are depicted in Table 2. Note that the hyperparameters of PPO are based on the original paper.