Maximizing Influence with Graph Neural Networks

George Panagopoulos[1], Nikolaos Tziortziotis[2], Michalis Vazirgiannis[1], Fragkiskos D. Malliaros[3]

[1]*École Polytechnique, IP Paris, France*

{george.panagopoulos, michalis.vazirgiannis}@polytechnique.edu

[2]*Jellyfish, France*

ntziorzi@gmail.com

[3]*Université Paris-Saclay, CentraleSupélec, Inria, France*

fragkiskos.malliaros@centralesupelec.fr

Abstract—Finding the seed set that maximizes the influence spread over a network is a well-known NP-hard problem. Though a greedy algorithm can provide near-optimal solutions, the subproblem of influence estimation renders the solutions inefficient. In this work, we propose GLIE, a graph neural network that learns how to estimate the influence spread of the independent cascade. GLIE relies on a theoretical upper bound that is tightened through supervised training. Experiments indicate that it provides accurate influence estimation for real graphs up to 10 times larger than the train set. Subsequently, we incorporate it into two influence maximization techniques. We first utilize Cost Effective Lazy Forward optimization substituting Monte Carlo simulations with GLIE, surpassing the benchmarks albeit with a computational overhead. To improve computational efficiency we develop a provably submodular influence spread based on GLIE's representations, to rank nodes while building the seed set adaptively. The proposed algorithms are inductive, meaning they are trained on graphs with less than 300 nodes and up to 5 seeds, and tested on graphs with millions of nodes and up to 200 seeds. The final method exhibits the most promising combination of time efficiency and influence quality, outperforming several baselines.

Index Terms—influence maximization, graph neural networks, graph representation learning

I. INTRODUCTION

Several real-world problems can be cast as combinatorial optimization problem over a graph. From distributing packages [1] and vehicles' management [2] optimization on graphs lies in the core of many real-world problems that are vital to our way of living. Unfortunately, the majority of these problems are NP-hard, and hence we can only approximate their solution in a satisfactory time limit that matches the real world requirements. Recent machine learning methods have emerged as a promising solution to develop heuristic methods that provide fast and accurate approximations. The general idea is to train a supervised or unsupervised learning model to infer the solution given an unseen graph and the problem constraints. The models tend to consist of Graph Neural Networks (GNNs) to encode the graph and the nodes, Q-learning [3] to produce sequential predictions, or a combination of both. The practical motivation behind learning to solve combinatorial optimization problems, is that inference time is faster than running an exact combinatorial solver [4]. That said, specialized combinatorial algorithms like CONCORDE for the *Traveling Salesman Problem* (TSP) or GUROBI in general, cannot be surpassed yet [5].

Though many such methods have been proposed for a plethora of problems, influence maximization (IM) has not been addressed yet extensively. IM addresses the problem of finding the set of nodes in a network that would maximize the number of nodes reached by starting a diffusion from them [6]. The problem is proved to be NP-hard, from a reduction to the set-cover problem. Moreover, the influence estimation (IE) problem that is embedded in IM, i.e., estimating the number of nodes influenced by a given seed set, is #P-hard and would require $2^{|E|}$ possible combinations to compute exactly, where $|E|$ is the number of network edges [7]. Typically, influence estimation is approximated using repetitive Monte-Carlo (MC) simulations of the independent cascade (IC) diffusion model [8]. In general, the seed set is built greedily, taking advantage of the submodularity of the influence function which guarantees at least $(1 - \frac{1}{e})$ approximation to the optimal. Although the latter lacks of efficiency as one still has to estimate the influence of every candidate seed in every step of building the seed sets. Hence, several scalable algorithms [9] and heuristics [10] were developed capitalizing on sketches or the structure of the graph to produce more efficient solutions.

We address IM using graph neural networks to capitalize on the aforementioned advantages and their ability to easily incorporate contextual information such as user profiles and topics [11], a task that remains unsolvable for non-specialized IM algorithms and heuristics. We propose GLIE, a GNN that provides efficient IE for a given seed set and a graph with influence probabilities. It can be used as a standalone influence predictor with competitive results for graphs up to 10 times larger than the train set. Moreover, we leverage GLIE for IM, combining it with CELF [12], which typically does not

scale beyond networks with thousands of edges. The proposed method runs in networks with millions of edges in seconds and exhibits better influence spread than a state-of-the-art algorithm and previous GNN-RL methods for IM. In addition, we propose PUN, a method that uses GLIE's representations to compute the number of neighbors predicted to be uninfluenced and uses it as an approximation to the marginal gain. We prove PUN's influence spread is submodular and monotone, and hence can be optimized greedily with a guarantee, in contrast to prior learning-based methods. The experiments indicate that PUN provides the best balance between influence quality and efficiency.

The paper is organized as follows. Section II presents an overview of relevant approaches and clarifies the proposed models' advantages. Section III describes the proposed methods, starting with IE and advancing progressively towards faster methods for IM. Section IV presents the experimental results for IE and IM. Finally, Section V summarizes the contribution and presents future steps.

II. RELATED WORK

The first approach to solving combinatorial optimization (CO) using neural networks was based on attention NNs for discrete structures, POINTERNETS [13], followed by an architecture that combines POINTERNETS with an actor-critic training to find the best route for TSP [14]. The first architecture that utilized graph-based learning was S2N-DQN [15], using STRUCT2VEC to encode the states of the nodes and the graph, and training a Deep Q-network (DQN) model that chooses the right node to add in a solution given the current state.

Based on S2V-DQN, a DQN for the network dismantling problem was recently proposed [16]. The model, named FINDER, uses a deep Q-learning architecture where the representations are derived by three GRAPHSAGE layers. The reward is based on the size of the giant connected component size, i.e., every new node (seed) chosen, aims to dismantle the network as much as possible. Some of the main advantages of FINDER is that it is trained on small synthetic data, which are easy to make, and can extrapolate to relatively large graphs. On the other hand, one of the core disadvantages is that it can not work with directed graphs and weighted edges. Another recent supervised deep learning approach on IM, GCOMB [17], utilizes a probabilistic greedy to produce scores on graphs and trains a GNN to predict them. A Q-network receives the scores along with an approximate calculation of the node's neighborhood correlation with the seed set, to predict the next seed. This approach, though scalable and comparable to SOTA in accuracy, has to be trained on a large random subset of the graph (30% of it) and tested on the rest. This makes the model graph-specific, i.e., it has to be retrained to perform well on a new graph. This imposes a serious overhead, considering the time required for training, subsampling, and labeling these samples using the probabilistic greedy method with traditional IE. As shown in [17] Appendix G, it takes more than hundreds of minutes and is thus out of our scope. Another GNN that addresses influence prediction is [18]. DEEPIS uses the power

sequence of the influence probability matrix and a two-layer GNN to regress the susceptibility of each node. Subsequently, the estimation propagates in the neighbors based on the IC probability. DEEPIS is a different architecture then GLIE, which receives only indications of the seed set. Moreover, DEEPIS is not tested extensively in influence maximization and as we will see in the experimental section, its use of the powers of influence probability matrix is detrimental to its scalability. Finally, a recent work on learning approximations to general submodular policies [19] requires a specific model to capture the state of IM, which is non-trivial to devise. A different branch of learning-based IM relies on supplementary information such as diffusion cascades [20] to derive a more effective IM algorithm [21]. This is clearly diagonal to the current methodology, which does not assume any further information from the typical IM setting.

In this paper, we propose an approach that combines the advantages of the aforementioned methods in that it is only trained on small simulated data once and generalizes to larger graphs, and it addresses the problem of IM in weighted directed networks. Furthermore, the approach can be broken down into a GNN for influence estimation and two IM methods. The former can act alone as an influence predictor and be competitive with relevant methods, such as DMP [22] for graphs up to one scale larger than the train set. GLIE is used to propose: (1) CELF-GLIE, CELF [12] with GLIE as influence estimator; (2) PUN, an adaptive IM method [23] that optimizes greedily a submodular influence spread using GLIE's representations.

We note here that the majority of the relevant literature on deep learning for combinatorial optimization addresses small graphs [15] which makes them not applicable to our task. More scalable, unsupervised methods are tailored to specific problems and are non-trivial to adjust to our problem, with the exception of [24] which was found significantly worse than the SOTA algorithm we compare with [17].

III. METHODOLOGY

A. GLIE: Graph Learning-based Influence Estimation

In this section, we introduce GLIE, a GNN model that aims to learn how to estimate the influence of seed set S over a graph $G = (V, E)$. Let $A \in \mathbb{R}^{n \times n}$ be the adjacency matrix and $X \in \mathbb{R}^{n \times d}$ be the features of nodes, representing which nodes belong to the seed set by 1 and 0 otherwise:

$$X_u = \begin{cases} \{1\}^d, & u \in S \\ \{0\}^d, & u \notin S \end{cases} . \tag{1}$$

For the analysis that follows, we set $d = 1$. More dimensions will become meaningful when we parameterize the problem. If we normalize A by each row, we form a row-stochastic transition matrix, as:

$$A_{uv} = p_{vu} = \begin{cases} \frac{1}{\deg(u)}, & v \in \mathcal{N}(u) \\ 0, & v \notin \mathcal{N}(u) \end{cases} , \tag{2}$$

where $\deg(u)$ is the in-degree of node u and $\mathcal{N}(u)$ is the set of neighbors of u. Based on the weighted cascade model [6], each row u stores the probability of node u being influenced by

each of the other nodes that are connected to it by a directed link $v \to u$. Note that, in case of directed influence graphs, A should correspond to the *transpose* of the adjacency matrix. The influence probability $p(u|S)$ resembles the probability of a node u getting influenced if its neighbors belong in the seed set, i.e., during the first step of the diffusion. We can use message passing to compute a well-known upper bound $\hat{p}(u|S)$ of $p(u|S)$ for u:

$$\hat{p}(u|S) = A_u \cdot X = \sum_{v \in \mathcal{N}(u) \cap S} \frac{1}{\deg(u)} = \qquad (3)$$

$$\sum_{v \in \mathcal{N}(u) \cap S} p_{vu} \geq 1 - \prod_{v \in \mathcal{N}(u) \cap S} (1 - p_{vu}) = p(u|S), \quad (4)$$

where the second equality stems from the definition of the weighted cascade and the inequality from the proof in [25], App. A. As the diffusion covers more than one-hop, the derivation requires repeating the multiplication to approximate the total influence spread. To be specific, computing the influence probability of nodes that are not adjacent to the seed set requires estimating recursively the probability of their neighbors being influenced by the seeds. If we let $H_1 = A \cdot X$, and we assume the new seed set S^t to be the nodes influenced in the step $t-1$, their probabilities are stored in H_t, much like a diffusion in discrete time. We can then recompute the new influence probabilities with $H_{t+1} = A \cdot H_t$.

Theorem 1. *The repeated product $H_{t+1} = A \cdot H_t$ computes an upper bound to the real influence probabilities of each infected node at step $t + 1$.*

Proof.

$$\hat{p}^t(u|S^t) = A_u \cdot H_t = \sum_{v \in \mathcal{N}(u) \cap S^t} \hat{p}_v p_{vu} \geq \qquad (5)$$

$$\sum_{v \in \mathcal{N}(u) \cap S^t} p_v p_{vu} \geq 1 - \prod_{v \in \mathcal{N}(u) \cap S^t} (1 - p_v p_{vu}) = p^t(u|S^t) \quad (6)$$

- (5) stems from (4) in the manuscript:

$$\hat{p}(u|S) = A_u \cdot X = \sum_{v \in \mathcal{N}(u) \cap S} \frac{1}{\deg(u)} = \sum_{v \in \mathcal{N}(u) \cap S} p_{vu}$$

$$\geq 1 - \prod_{v \in \mathcal{N}(u) \cap S} (1 - p_{vu}) = p(u|S).$$

- (6) can be proved by induction similar to [25]. For every $p_v \leq 1$, the base case $\sum_{v \in \mathcal{X}} p_v p_{vu} \geq 1 - \prod_{v \in \mathcal{X}} (1 - p_v p_{vu})$ is obvious for $|\mathcal{X}| = 1$. For $|\mathcal{X}| > 1$, we have:

$$1 - \prod_{v \in \mathcal{X}} (1 - p_v p_{vu}) = 1 - (1 - p_x p_{xu}) \prod_{v \in \mathcal{X} \setminus x} (1 - p_v p_{vu})$$

$$= 1 - \prod_{v \in \mathcal{X} \setminus x} (1 - p_v p_{vu}) + p_x p_{xu} \prod_{v \in \mathcal{X} \setminus x} (1 - p_v p_{vu}) \quad (7)$$

$$\leq \sum_{v \in \mathcal{X} \setminus x} p_v p_{vu} + p_x p_{xu} \prod_{v \in \mathcal{X} \setminus x} (1 - p_v p_{vu})$$

$$\leq \sum_{v \in \mathcal{X} \setminus x} p_v p_{vu} + p_x p_{xu} = \sum_{v \in \mathcal{X}} p_v p_{vu}. \quad (8)$$

- In (6) we have $p(u|v) = p_v p_{vu}$ per definition of the IC, and thus $p(u|S) = 1 - \prod_{v \in \mathcal{N}(u) \cap S} (1 - p_v p_{vu})$,

where $p_v = 1$ for $v \in S^1$, which are the initial seed set that are activated deterministically. Thus, (6) stands, and these probabilities are an upper bound of the real influence probabilities. Hence, the influence spread $\hat{\sigma}(S) = \sum_{(u,v) \in E} p_{uv}$ is an upper bound to the real $\sigma(S)$.
□

In reality, due to the existence of cycles, two problems arise. Firstly, if the process is repeated, the influence of the original seeds may increase again, which comes in contrast with the independent cascade model. This can be controlled by minimizing the repetitions, e.g., four repetitions cause the original seeds to be able to reinfect other nodes in a network with triangles. To this end, we leverage up to three neural network layers. Another problem due to cycles pertains to the probability of neighbors influencing each other. In this case, the product of the complementary probabilities in Eq. (4) does not factorize for the non-independent neighbors. This effect was analyzed extensively in [22], App. B, and proved that the influence probability computed by $p(u|S)$ is itself an upper bound on the real influence probability for graphs with cycles. Intuitively, the product that represents non-independent probabilities is larger than the product of independent ones. This renders the real influence probability, which is complementary to the product, smaller than what we compute.

We can thus contend that the estimation $\hat{p}(u|S)$ provides an upper bound on the real influence probability—and we can use it to compute an upper bound to the real influence spread of a given seed set i.e., the total number of nodes influenced by the diffusion. Since message passing can compute inherently an approximation of influence estimation, we can parameterize it to learn a function that tightens this approximation based on supervision. In our neural network architecture, each layer consists of a GNN with batchnorm and dropout omitted here, and starting from $H_0 = X \in \mathbb{R}^{n \times d}$ we have:

$$H_{t+1} = \text{ReLU}([H_t, AH_t]W_t). \qquad (9)$$

The readout function that summarizes the graph representation based on all nodes' representations is a summation with skip connections:

$$H_S^G = \sum_{v \in V} [H_0^v, H_1^v, \ldots, H_t^v]. \qquad (10)$$

This representation captures the probability of all nodes being active throughout each layer. The output that represents the predicted influence spread is derived by:

$$\hat{\sigma}(S) = \text{ReLU}(H_S^G W_o). \qquad (11)$$

Our loss function is a simple least squares regression. Note that, in the case where W_t is an untrained positive semidefinite Gaussian random matrix in $[0, 1]$, the representations of each layer H_t^v would correspond to the upper bound of the influence probability of seed set's t-hop neighbors [22]. This upper bound is not retained once the weights W_t are trained. In our approach, the parameters of the intermediate layers W_t are trained such that the upper bound is reduced and the final layer W_o can
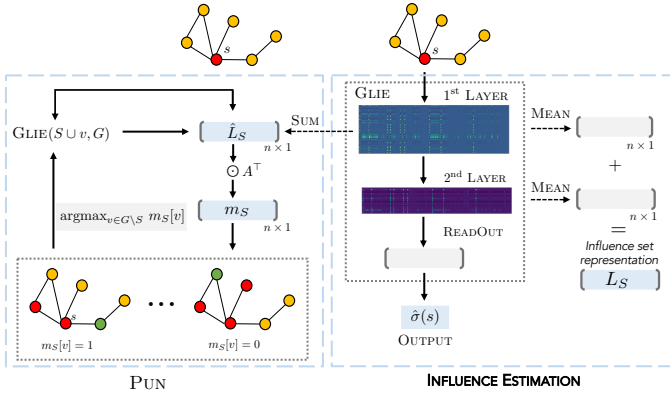
Fig. 1. A visual depiction of the pipeline. The layers of GLIE are depicted by a heatmap of an actual seed during inference time, showing how the values vary through different nodes (columns).

combine the probabilities to derive a cumulative estimate for the total number of influenced nodes. We empirically verify this by examining the layer activations which can be seen in Fig. 1. The heatmaps indicate a difference between columns (nodes) expected to be influenced, meaning we could potentially predict not only the number but also who will be influenced. However, since $\hat{\sigma}$ is derived by multiple layers, the relationships and thresholds to determine the exact influenced set are not straightforward.

B. CELF-GLIE: Cost Effective Lazy Forward with GLIE

Cost Effective Lazy Forward (CELF) [12] is an improved version of the greedy algorithm for influence maximization that exploits the property of submodularity to select seed nodes efficiently. By maintaining a sorted list of nodes based on their influence spread, CELF identifies the best node with the highest marginal gain in each iteration, resulting in significantly faster execution times compared to the greedy algorithm without sacrificing its original effectiveness. In our case, we propose an adaptation where we substitute the original CELF IE based on MC IC with the output of GLIE. Since we do not prove the submodularity of $\hat{\sigma}$, we can not contend that the theoretical guarantee is retained, so we use this as a heuristic. CELF-GLIE has two main computational bottlenecks. First, it requires computing the initial IE for every node in the first step. Second, although it alleviates the need to test every node in every step, it still requires IE for at least one node in each step. Second, We will try to alleviate both.

C. PUN: Potentially Uninfluenced Neighbors

Computing the influence spread of every node in the first step is computationally demanding. We thus seek a method that can surpass this hinder and provide adequate performance. We first utilize the activations mentioned above to define the set of influenced nodes on the step that corresponds to that layer. Let $\hat{L}_S, L'_S \in \{0,1\}^n$ be the binary vectors with 1s in

nodes predicted to be uninfluenced and nodes predicted to be influenced respectively:

$$\hat{L}_S = \mathbb{1}\left\{\sum_{i=0}^{d_1} H_1^i \le 0\right\} \qquad L'_S = \mathbb{1}\left\{\sum_{i=0}^{d_1} H_1^i > 0\right\}. \quad (12)$$

where T is the number of layers, and $H_t^i \in \mathbb{R}^{n \times 1}$ is a column from H_t. This vector contains a label for each node whose sign indicates if it is predicted to be influenced. L'_S provides a rough estimate, but it allows for a simpler influence spread which we can optimize greedily:

$$\sigma^m(S) = |L'_S|. \quad (13)$$

We can use $\hat{L}_S$ and message passing to predict the amount of a node's neighborhood that remains uninfluenced, i.e., the **P**otentially **U**ninfluenced **N**eighbors (PUN), weighted by the respective probability of influence for a node u,

$$m_S[u] = \sum_{v \in N(u)} A_{uv} \hat{L}_v = A_u^\top \cdot \hat{L}_S \in \mathbb{R}^{n \times 1}. \quad (14)$$

For efficiency, we can compute $m_S = A^\top \hat{L}$ which can be considered an approximation to all nodes' marginal gain on their immediate neighbors. We can thus optimize this using $\arg\max(m_S)$, as shown in Fig. (1). In order to establish that σ^m can be optimized greedily with a theoretical guarantee of $(1 - \frac{1}{e})$OPT, in Theorem 2 we prove its monotonocity and submodularity.

Theorem 2. *The influence spread σ^m is submodular and monotone.*

Proof. The proof can be found in Appendix A. □

PUN can be seen in the left part of Fig. 1. We start by setting the first seed as the node with the highest degree, which can be considered a safe assumption as in practice it is always part of seed sets. We use GLIE(S, G) to retrieve $\hat{L}_S$, which we use to find the next node on $\arg\max_{v \in G \setminus S} m_S[v]$ and the new $\hat{L}_{S \cup \{v\}}$. One disadvantage of PUN is that σ^m is an underestimation of the predicted influence. Contrasted with the upper bound, DMP, σ^m is not as accurate as $\hat{\sigma}$, but allows us to compute efficiently a submodular proxy for the marginal gain. This underestimation means that a part of the network considered uninfluenced in $\hat{L}_S$ is measured as a potential gain for their neighbors, hence the ranking based on m_S can be affected negatively. For this purpose, we will use adaptive full-feedback selection (AFF), where after selecting a new seed node, we remove it from the network along with nodes predicted to be influenced. It has been proved in the seminal work of [26] that an AFF greedy algorithm for a submodular and monotonic function is guaranteed to have a competitive performance with the optimal policy. In our case, we will use an AFF update every k seeds, as it adds a small computational overhead if we do it in every step. The benefit of PUN is twofold. Firstly, as we remove the influenced node and truncate the seed set, GLIE produces an increasingly more valid estimate because it performs better when the graph and seed set are smaller.

Secondly, as the neighborhood size decreases, the effect of missed influenced nodes is diminished in m_S.

IV. Experimental Evaluation

All the experiments are performed in a PC with an NVIDIA GPU TITAN V (12GB RAM), 256GB RAM and an Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz. The source code can be found in the supplementary files.

A. Influence Estimation

In order to train for the influence estimation task, we create a set of labeled samples, each consisting of the seed set S and the corresponding influence spread $\sigma(S)$. We create 100 Barabasi-Albert [27] and Holme-Kim [28] undirected graphs ranging from 100 to 200 nodes and 30 from 300 to 500 nodes. 60% are used for training, 20% for validation, and 20% for testing. We have used these network models because the degree distribution resembles the one of real world networks. The influence probabilities are assigned based on the weighted cascade model, i.e., a node u has an equal probability $1/\deg(u)$ to be influenced by each of her $\mathcal{N}(u)$ nodes. This model requires a directed graph, hence we turn all undirected graphs into directed ones by appending reverse edges. Though estimating influence probabilities is a problem on its own [29], in the absence of extra data, the weighted cascade is considered more realistic than pure random assignments [6]. To label the samples, we run the CELF algorithm using $1,000$ Monte Carlo (MC) ICs for influence estimation, for up to 5 seeds. Note that, we expect running $10,000$ simulations would provide a more qualitative supervision. However, on the one hand, the training time would increase exponentially, and on the other, due to the training graphs being relatively small, the difference is minuscule. The optimum seed set for size 1 to 5 is stored, along with 30 random negative samples for each seed set size. This amounts to a total of $20,150$ training samples. Each training sample for GLIE corresponds to a triple of a graph G, a seed set S, and a ground truth influence spread $\sigma(S)$ that serves as a label to regress on. The random seed sets are used to capture the average influence spread expected for a seed set of about that size. This creates "average samples" which would constitute the whole dataset in other problems. In IM however, the difference in σ between an average seed set and the optimal can be significant, hence training solely on the random sets would render our model unable to predict larger values that correspond to the optimum. That is why we added the aforementioned samples of the optimum seed set computed using CELF. We deem the combination of 30 random and 1 optimum a more balanced form of supervision, as you expect the crucial majority of the seed sets to have an average σ.

Regarding model training, we have used a small-scale grid-search using the validation set to find the optimum batch size 64, dropout 0.4, number of layers 2, hidden layer size 64, and feature dimension 50. More importantly, we observed that it is beneficial to decrease the hidden layer size (by a factor of 2) as the depth increases, i.e., go from 32 to 16. This means that the

1-hop node representations are more useful compared to the 2-hop ones and so on—validating the aforementioned conclusion that the approximation to the influence estimation in Eq. (4), diverges more as the message passing depth increases. The training then proceeds for 100 epochs with an early stopping of 50 and a learning rate of 0.01.

TABLE I
GRAPH DATASETS.

	Graph	# Nodes	# Edges
Sim	Test/Train	$100 - 500$	$950 - 4,810$
	Large	$1,000 - 2,000$	$11,066 - 19,076$
Small	Crime (CR)	829	$2,946$
	HI-II-14 (HI)	$4,165$	$26,172$
	GR Colab (GR)	$5,242$	$28,980$
Large	Enron (EN)	$33,697$	$361,622$
	Facebook (FB)	$63,393$	$1,633,660$
	Youtube (YT)	$1,134,891$	$5,975,246$

We evaluate the models in three different types of graphs. The first is the test set of the dataset mentioned above. The second is a set of 10 power-law large graphs ($1,000 - 2,000$ nodes) to evaluate the capability of the model to generalize in networks that are larger by one factor. The third is a set of three real-world graphs, namely the *Crime* (CR), *HI-II-14* (HI), and *GR collaborations* (GR). More information about the datasets is given in Table I.

The real graphs are evaluated for varying seed set sizes, from 2 to 10, to test our model's capacity to extrapolate to larger seed set sizes. Due to the size of the latter two graphs (HI and GR), we take for each seed set size the top nodes based on the degree as the optimum seed set along with a 30 random seed sets for the large simulated graphs and 3 for the real graphs, to validate the accuracy of the model in non-significant sets of nodes. We have compared the accuracy of influence estimation with DMP [22]. We could not utilize the influence estimation of UBLF [25] because its central condition is violated by the weighted cascade model and the computed influence is exaggerated to the point it surpasses the nodes of the network. The average error throughout all datasets and the average influence can be seen in Table II, along with the average time.

We evaluate the retrieved seed set using the independent cascade, and the results are shown in Table III. We should underline here that this task would require more then 3 hours for the *Crime* dataset and days for *GR* using the traditional approach with $1,000$ MC IC. As we can see in Table III, GLIE-CELF allows for a significant acceleration in computational time, while the retrieved seeds are more effective. Moreover, in CELF, the majority of time is consumed in the initial computation of the influence spread, i.e., the overhead to compute 100 instead of the 20 seeds shown in Table III, amounts to 0.11, 0.22 and 0.19 seconds for the three datasets respectively.

TABLE II
AVERAGE MAE DIVIDED BY THE AVERAGE INFLUENCE AND TIME (IN SECONDS) THROUGHOUT ALL SEED SET SIZES AND SAMPLES, ALONG WITH THE REAL AVERAGE INFLUENCE SPREAD.

| Graph (seeds)	DMP		GLIE	
	MAE	Time	MAE	Time
Test $(1-5)$	0.076	0.05	**0.046**	**0.0042**
Large $(1-5)$	**0.086**	0.44	0.102	**0.0034**
CR $(1-10)$	**0.009**	0.11	0.044	**0.0029**
HI $(1-10)$	**0.041**	2.84	0.056	**0.0034**
GR $(1-10)$	0.122	4.32	**0.084**	**0.0042**

TABLE III
IM FOR 20 SEEDS WITH CELF, USING THE PROPOSED (GLIE) SUBSTITUTE FOR INFLUENCE ESTIMATION AND EVALUATING WITH 10,000 MC INDEPENDENT CASCADES (IC).

Graph (seeds)	Seed overlap	DMP-CELF		GLIE-CELF	
		Infl	Time	Infl	Time
CR (20)	14	221	83	**229**	**1.0**
HI (20)	13	1,235	8,362	**1,281**	**5.49**
GR (20)	12	295	16,533	**393**	**7.01**

Fig. 2. PUN vs. IMM for IC with $p = 0.01$.

B. Influence Maximization

Our main benchmark is a state-of-the-art IM method, IMM [9], which capitalizes on reverse reachable sets to estimate influence. Specifically, it produces a series of such sketches and uses them to approximate the influence spread without simulations. This results in remarkable acceleration with a theoretical guarantee. Note that, IMM is considered one of the state-of-the-art algorithms and surpasses various heuristics [30]. We set $e = 0.5$ as proposed by the authors. We also compare with FINDER, which is analyzed in Section II, and with the most well-known heuristic methods for the Independent Cascade. PMIA [7] computes the influence spread based on local approximations. DEGREEDISCOUNT [10] builds a seed set using the node's degree, which is recomputed based on the current seed set and its influence. Finally, K-CORES [31] is the a graph degeneracy metric that uncovers nodes that are part of densely connected subgraphs.

The results for the influence spread of 200 seeds as computed by simulations of MC ICs can be seen in Table IV, while the time results are shown in Tables V and VI. The best result is in bold and the second best is underlined. One can see that GLIE-CELF exhibits overall superior influence quality compared to the rest of the methods, but is quite slower. PUN requires only one influence estimation in every step and no initial computation. It exhibits 3 to 60 times acceleration compared to IMM while its computational overhead moving from smaller to larger graphs is sublinear to the number of nodes. In terms of influence quality, PUN is first or second in the majority of the datasets and this effect becomes more clear as the seed set size increases. DEGDISC is faster than PUN in smaller graphs
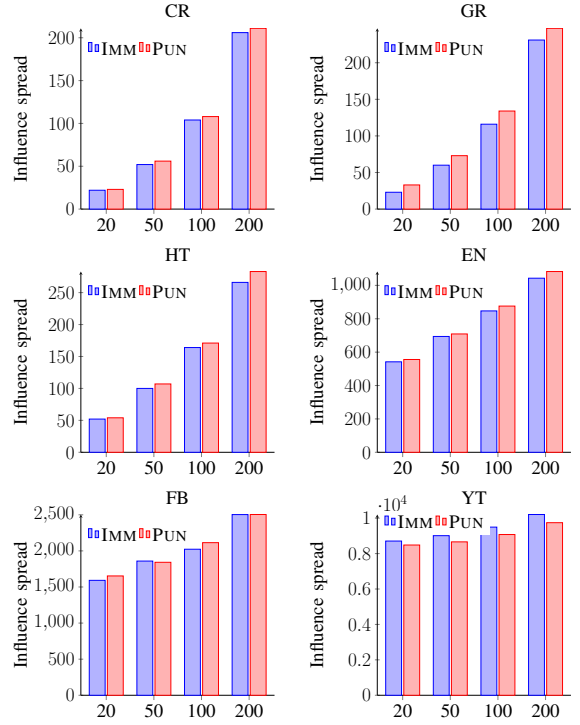
but slower in larger and overall worse in seed set quality. PMIA provides medium seed set quality but is computationally inefficient. IMM is clearly not the fastest method, but it is very accurate, especially for smaller seed set sizes. FINDER exhibits the least accurate performance, which is understandable given that it solves a relevant problem and not exactly IM for IC. The computational time presented is the time required to solve the node percolation, in which case it may retrieve a bigger seed set than 100 nodes. Thus, we can hypothesize it is quite faster for a limited seed set, but the quality of the retrieved seeds is the least accurate among all methods. Overall, we can contend that PUN provides the best accuracy-efficiency tradeoff from the examined methods.

DEEPIS, as analyzed in related work, resembles GLIE, in that it computes influence estimation using a neural network. We follow the authors' methodology and train the model using their code on the proposed Cora ML [18]. We use it as an influence estimation oracle in CELF, similar to GLIE-CELF. Unfortunately, it is infeasible to scale in the larger datasets due to the need for explicit powers of the influence matrices that required more than 24 GB of GPU RAM. For comparison, this model required 501.61, 835.4, and 1,602.5 seconds for the CR, GR, and HI datasets, respectively. This further supports the superiority of PUN.

Furthermore, we compare IMM and PUN on the same graphs with uniform influence probabilities $p = 0.01$ in Figure 2, as a substitute to the weighted cascade assignment. We observe that PUN outperforms IMM. Finally, we performed an experiment to compare PUN without the use of GPU for 100 seeds. The results are reported in Table VII. It is visible that GPU provides

TABLE IV
INFLUENCE SPREAD COMPUTED BY 10,000 MC ICS FOR 200 SEEDS.

Graph	GLIE-CELF	PUN	K-CORE	PMIA	DEGDISC	IMM	DEEPIS-CELF	FINDER
CR	**661**	657	647	656	644	650	501.61	642
GR	1,617	**1,626**	701	1,566	1415	835.40	1,617	1,286
HI	2,685	**2,688**	2,540	2,685	2,614	2,668	1602.5	2,625
EN	17,601	**17,614**	13,015	17,534	16,500	17,497	-	17,244
FB	10,981	10,626	6,434	7,688	10,309	**11,007**	-	10,801
YT	246,439	244,579	110,409	242,057	236,726	247,178	-	50,435

TABLE V
COMPUTATIONAL TIME IN SECONDS (VS ALGORITHMS).

Graph	GLIE-CELF	PUN	IMM	FINDER
CR	2.00	0.25	**0.19**	0.41
GR	4.55	**0.26**	0.95	2.36
HI	2.19	**0.27**	1.29	1.01
EN	15.49	**0.97**	10.47	9.30
FB	287.7	**3.1**	171.25	56.80
YT	151.33	**28.92**	82.13	191.00

TABLE VI
COMPUTATIONAL TIME IN SECONDS (VS HEURISTICS).

Graph	PUN	DEGDISC	K-CORE	PMIA
CR	0.25	0.21	0.06	**0.04**
GR	0.26	0.80	**0.13**	1.5
HI	0.27	1.36	0.14	**0.12**
EN	**0.97**	26.74	2.06	2.17
FB	**3.1**	22.77	9.29	10.62
YT	**28.92**	4006.29	54.38	74.91

TABLE VII
PUN CPU VS. GPU TIME (SEC).

Graph	PUN GPU	PUN CPU	IMM
CR	0.15	0.17	0.13
GR	0.17	0.27	0.57
HT	0.17	0.20	0.56
EN	0.52	2.44	4.78
FB	1.42	17.5	69.9
YT	13.2	97.5	55.4

a substantial acceleration, but PUN remains the faster option even without it.

V. CONCLUSION

We have proposed GLIE, a GNN-based model for influence estimation. We showcase its accuracy in that task and further utilize it to address the problem of IM. We developed two methods based on the representations and the predictions of GLIE. GLIE-CELF, an adaptation of a classical algorithm that surpasses SOTA but with significant computational overhead, and textscPun, a submodular function that acts as proxy for the marginal gain and can be optimized adaptively, striking a balance between efficiency and accuracy.

For a typical IM algorithm, it is not straightforward to consider the topic of the information shared or the user's characteristics [32]. A significant practical advantage of a neural network approach is the easy incorporation of such complementary data by adding the corresponding embeddings in the input, as has been done in similar settings [11]. We thus deem an experiment with contextual information a natural next step, given a proper dataset. Finally, we also plan to examine the potential of training a reinforcement learning module, i.e., receiving real feedback from each step of the diffusion that could update both the Q-NET and GLIE. This would allow the model to adjust its decisions based on the partial feedback received during the diffusion.

APPENDIX A
PROOF OF THEOREM 2

For the purposes of the proof, $P \in \{1\}^{hd \times 1}$ and we define the support function $\mathcal{S}(v) = \{i \in [1, n], v_i \neq 0\}$ as the set of indices of non zero rows in a matrix such as X_i, of layer i. Let R represent ReLU and b_{tr}, st_{tr} the mean and standard deviation computed by the batchnorm.

Proof. Monotonicity; $\forall i < j, \mathcal{S}_i \subset \mathcal{S}_j$.

$$\mathcal{S}(X_j) \supset \mathcal{S}(X_i) \Rightarrow \mathcal{S}(X_j W) \supseteq \mathcal{S}(X_i W) \quad (15)$$

$$\mathcal{S}(AX_j W) \supseteq \mathcal{S}(AX_i W) \Rightarrow \mathcal{S}((R(AX_j W) - b_{tr})/st_{tr})$$
$$\supseteq \mathcal{S}((R(AX_i W) - b_{tr})/st_{tr}) \quad (16)$$

$$\mathcal{S}(H_j) \supseteq \int(H(\mathcal{S}_i)) \Rightarrow \mathcal{S}(H_j P) \supseteq \mathcal{S}(H_i P) \quad (17)$$

$$|\mathbb{1}_{>0}\{H_j P\}| \geq |\mathbb{1}_{>0}\{H_i P\}| \Rightarrow \sigma^m(\mathcal{S}_j) \geq \sigma^m(\mathcal{S}_i) \quad (18)$$

(15) First subset is by definition. Second is because X_j is a convex hull that contains X_i [33]. We multiply both sides by a real matrix $W \in \mathbb{R}^{d \times hd}$ which can equally dilate both convex hulls in terms of direction and norm. This transformation cannot change the sign of the difference between the elements of X_i and X_j and hence cannot interfere with the support of X_j over X_i. This becomes more obvious for $X \in \{0, 1\}^{n \times 1}$ and $W \in \mathbb{R}^{1 \times 1}$. Note that both can result in zero matrices so we use subset or equal. (16) A is a non-negative matrix and ReLU is a nonnegative monotonically increasing function. Subtract by the same number and divide by the same positive number in the right inequality. (17) Definition in Eq. (9); P is positive. (18) By definition of the support and of L'_S. $\square$

For the proof of submodularity we have to define $X_{iu} = X_{S_i \cup u}, u \in V$ and note by the definition of the input that $|X_{ju} - X_j| = |X_{iu} - X_i|$ for the l_1 norm (sum of all elements):

Proof. Submodularity $\forall i < j, S_i \subset S_j$:

$$|X_{ju} - X_j| = |X_{iu} - X_i| \Rightarrow |A(X_{ju} - X_j)| = |A(X_{iu} - X_i)| \tag{19}$$

$$|AX_{ju}W - AX_jW| = |AX_{iu}W - AX_iW| \tag{20}$$

$$R(|AX_{ju}W - AX_jW|) - 2b_{tr} = R(|AX_{iu}W - AX_iW| - 2b_{tr}) \tag{21}$$

$$|R(AX_{ju}W) - R(AX_jW) - 2b_{tr}| = \\ |R(AX_{iu}W) - R(AX_iW) - 2b_{tr}| \tag{22}$$

$$\mathcal{S}(R(AX_{ju}W) - R(AX_jW) - 2b_{tr}) = \\ \mathcal{S}(R(AX_{iu}W) - R(AX_iW) - 2b_{tr}) \tag{23}$$

$$\mathcal{S}(R(AX_{ju}W - b_{tr})) - \mathcal{S}(R(AX_jW) - b_{tr}) \subseteq \\ \mathcal{S}(R(AX_{iu}W - b_{tr})) - \mathcal{S}(R(AX_iW) - b_{tr}) \tag{24}$$

$$\mathcal{S}(H_{ju}) - \mathcal{S}(H_j) \subseteq \mathcal{S}(H_{iu}) - \mathcal{S}(H_i) \tag{25}$$

$$\sigma^m(S^j \cup \{u\}) - \sigma^m(S^j) \le \sigma^m(S^i \cup \{u\}) - \sigma^m(S^i) \tag{26}$$

(20) Distributive property and W similar to multiply by A. (24) The norm of the difference is distributed equally, but the right hand difference has as least the same or more positive elements because the norm of A, which is stochastic, is bounded by V hence X_u can give up to the same gain to AX_j and AX_i, the same number b_{tr} is subtracted, and more elements are activated by X_j then X_i as shown in (18). (25) We skipped dividing by st_{tr} for brevity.(26) Similar steps as (17) - (18). $\square$

Regarding the approximation of the marginal gain we first show that the node corresponding to the maximum m_S gives the maximum L'_{ju}: $A'_u \hat{L}_j \ge A'_v \hat{L}_i \Rightarrow L'_{ju} \ge L'_{iv}$.

$$A'_u \hat{L}_j = \sum_{v \in N(u)} A'_{uv} \hat{L}_j[v] = \sum_{v \in N(u)} A_{uv} L'_j[u] = \sum_{v \in N(u)} A_{uv} X_{ju}.$$

This means that m_S gives the node u that improves the biggest number of rows in AX_{ju} that are not already considered influenced. Since we know from Eq. (18) that $AX_{iu} \ge AX_{iv} \Rightarrow |L'_{iu}| \ge |L'_{iv}|$, the claim concludes. Hence, choosing the best node using the marginal gain approximation is as good as the real influence spread. Now we prove the submodularity of the proposed marginal gain.

Proof. Submodularity for the approximation of the marginal gain, $\forall i < j, S_i \subset S_j$, starting from (18):

$$|\mathbb{1}_{>0}\{H_j P\}| \ge |\mathbb{1}_{>0}\{H_i P\}|$$
$$|\mathbb{1}_{\le 0}\{H_j P\}| \le |\mathbb{1}_{\le 0}\{H_i P\}| \tag{27}$$

$$A'_u \hat{L}_j \le A'_u \hat{L}_i \rightarrow m_{S_j}[u] \le m_{S_i}[u] \tag{28}$$

$$(|L'_j| + m_{S_j}[u]) - |L'_j| \le (|L'_i| + m_{S_i}[u]) - |L'_i| \tag{29}$$

$$\sigma^m(S^j \cup \{u\}) - \sigma^m(S^j) \le \sigma^m(S^i \cup \{u\}) - \sigma^m(S^i) \tag{30}$$

(27) Complementarity between elements that are ≤ 0 and > 0. (28) Definition in Eq. (12) and multiply with non-negative row u from matrix A', Definition in Eq. (14). (30) By definition of σ^m in Eq. (13) and the marginal gain of u. $\square$

REFERENCES

[1] N. Mathew, S. L. Smith, and S. L. Waslander, "Planning paths for package delivery in heterogeneous multirobot teams," *IEEE T-ASE*, vol. 12, no. 4, pp. 1298–1308, 2015.

[2] N. Touati-Moungla and V. Jost, "Combinatorial optimization for electric vehicles management," *Journal of Energy and Power Engineering*, vol. 6, no. 5, pp. 738–743, 2012.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] C. K. Joshi, T. Laurent, and X. Bresson, "On learning paradigms for the travelling salesman problem," *CoRR*, 2019.

[5] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" in *ICLR*, 2019.

[6] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *KDD*, 2003.

[7] C. Wang, W. Chen, and Y. Wang, "Scalable influence maximization for independent cascade model in large-scale social networks," *DMKD*, vol. 25, no. 3, pp. 545–576, 2012.

[8] Y. Tian and R. Lambiotte, "Unifying information propagation models on networks and influence maximization," *Physical Review E*, vol. 106, no. 3, p. 034316, 2022.

[9] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *SIGMOD*, 2015.

[10] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *SIGKDD*, 2009.

[11] S. Tian, S. Mo, L. Wang, and Z. Peng, "Deep reinforcement learning-based approach to tackle topic-aware influence maximization," *Data Science and Engineering*, vol. 5, no. 1, pp. 1–11, 2020.

[12] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *KDD*, 2007.

[13] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *NeurIPS*, 2015.

[14] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *ICLR*, 2017.

[15] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *CoRR*, 2017.

[16] C. Fan, L. Zeng, Y. Sun, and Y.-Y. Liu, "Finding key players in complex networks through deep reinforcement learning," *Nature Machine Intelligence*, pp. 1–8, 2020.

[17] S. Manchanda, A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh, "Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs," in *NeurIPS*, 2020.

[18] W. Xia, Y. Li, J. Wu, and S. Li, "Deepis: Susceptibility estimation on social networks," in *WSDM*, 2021.

[19] A. Alieva, A. Aceves, J. Song, S. Mayo, Y. Yue, and Y. Chen, "Learning to make decisions via submodular regularization," in *ICLR*, 2020.

[20] G. Panagopoulos, F. D. Malliaros, and M. Vazirgiannis, "Diffugreedy: An influence maximization algorithm based on diffusion cascades," in *Complex Networks*, 2018, pp. 392–404.

[21] G. Panagopoulos, F. D. Malliaros, and M. Vazirgianis, "Influence maximization using influence and susceptibility embeddings," in *ICWSM*, 2020, pp. 511–521.

[22] A. Y. Lokhov and D. Saad, "Scalable influence estimation without sampling," *CoRR*, 2019.

[23] B. Cautis, S. Maniu, and N. Tziortziotis, "Adaptive influence maximization," in *KDD*, 2019.

[24] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," *CoRR*, 2018.

[25] C. Zhou, P. Zhang, W. Zang, and L. Guo, "On the upper bounds of spread for greedy algorithms in social network influence maximization," *IEEE TKDE*, vol. 27, no. 10, pp. 2770–2783, 2015.

[26] D. Golovin and A. Krause, "Adaptive submodularity: Theory and applications in active learning and stochastic optimization," *JAIR*, vol. 42, pp. 427–486, 2011.

[27] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[28] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Physical review E*, vol. 65, no. 2, p. 026107, 2002.

[29] G. Panagopoulos, F. Malliaros, and M. Vazirgiannis, "Multi-task learning for influence estimation and maximization," *IEEE TKDE*, 2020.

[30] K. Jung, W. Heo, and W. Chen, "Irie: Scalable and robust influence maximization in social networks," in *ICDM*, 2012.

[31] F. D. Malliaros, C. Giatsidis, A. N. Papadopoulos, and M. Vazirgiannis, "The core decomposition of networks: Theory, algorithms and applications," *The VLDB Journal*, vol. 29, no. 1, pp. 61–92, 2020.

[32] W. Chen, T. Lin, and C. Yang, "Real-time topic-aware influence maximization using preprocessing," *Computational social networks*, vol. 3, no. 1, pp. 1–19, 2016.

[33] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.