Learning graph representations for influence maximization

George Panagopoulos, Nikolaos Tziortziotis, Michalis Vazirgiannis, Jun Pang, Fragkiskos D. Malliaros

Learning Graph Representations for Influence Maximization

George Panagopoulos[1*], Nikolaos Tziortziotis[2], Michalis Vazirgiannis[3], Jun Pang[1], Fragkiskos D. Malliaros[4]

[1]University of Luxembourg, Department of Computer Science, 6 avenue de la Fonte, Esch-sur-Alzette, L-4364, Luxembourg.
[2]Jellyfish, Rue Henner, Paris, 75009, France.
[3]École Polytechnique, Institut Polytechnique de Paris, Rte de Saclay, Palaiseau, 91120, France.
[4]Université Paris-Saclay, CentraleSupélec, Inria, 3 rue Joliot Curie, Gif-sur-Yvette, 91192, France.

*Corresponding author(s). E-mail(s): georgios.panagopoulos@uni.lu;
Contributing authors: ntziorzi@gmail.com; mvazirg@lix.polytechnique.fr; jun.pang@uni.lu; 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 three 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 PUN, a provably submodular influence spread based on GLIE's representations, to rank nodes while building the seed set adaptively. Furthermore, we take a step towards an end-to-end learnable approach in GRIM, a Q-learning model that learns how to choose seeds sequentially using GLIE's predictions. The proposed algorithms are inductive, meaning they are trained on graphs with a few hundred nodes and tested on graphs with millions. Our methods exhibit the most

promising combination of time efficiency and influence quality, outperforming several benchmarks.

Keywords: graph neural networks, influence maximization, combinatorial optimization, social networks

1 Introduction

Several real-world problems can be cast as a combinatorial optimization problem over a graph. From distributing packages [1] to vehicles' management [2] and graph clustering [3], optimization on graphs lies at 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, a reinforcement learning (RL) algorithm Q-learning [4] 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 [5]. In that regard, the field of neural algorithmic reasoning has made significant strides, including solutions to standard problems [6] and recursive ones [7]. The *Traveling Salesman Problem* (TSP) has been studied from multiple perspectives including attention-based and meta-learning solutions with promising generalization [8–10]. That said, specialized combinatorial algorithms like Concordess for the TSP or Gurobi in general, cannot be consistently surpassed yet at scale.

Though many such methods have been proposed for a plethora of problems, influence maximization (IM) has not yet been addressed 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 [11]. This problem definition can be formulated as $\text{argmax}_S I(S)$ s.t. $|S| < b$, i.e. we need to find the set of seed nodes $S \subset V$ in the graph $G = (E, V)$ such that the influence function I is maximized. 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 [12]. Typically, influence estimation is approximated using repetitive Monte-Carlo (MC) simulations of the independent cascade (IC) diffusion model [13]. 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. However, the latter lacks efficiency as one still has to estimate the influence of every candidate seed in every step of building the seed sets. Due to this, the complexity of

a greedy approach is lower, bounded by the complexity of the IE procedure and the number of uninfluenced nodes. In this work, we address IM using a GNN-based IE that is learned from small-scale examples and allows us to diminish this complexity in real-life IM cases. The main idea is to develop an algorithm that uses the GNN not only as a substitute for IE but also capitalizes on the learned representation to accelerate the seed selection. Our contributions can be summarized as follows:

1. 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.
2. We further leverage GLIE for IM, combining it with CELF [14], 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.
3. We devise 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.
4. Finally, we develop GRIM, a Q-learning-based architecture that utilizes GLIE's representations and predictions to obtain seeds sequentially while minimizing the number of influence estimations throughout steps.

The paper is organized as follows.

- Section 2 reviews relevant approaches, briefly explains them, and clarifies the proposed models' advantages.
- Section 3 describes the proposed methods, starting with IE and advancing progressively towards faster methods for IM.
- Section 4 presents the experimental results for IE and IM with a detailed comparison between them and various methods from the literature.
- Section 5 summarizes the contribution and presents future steps.

It should be noted that this is an extended version of the paper introducing GLIE [15]. GRIM is an alternative methodology to PUN that utilizes reinforcement learning to recover the seed selection policy, instead of optimization. Sections 3 and 4 have changed significantly to include GRIM, with an updated schematic representation of the pipeline in Fig. 1. Further experiments have been added to quantify the behavior of all the models in smaller seed sets (50 and 100) in Section 4, and an analysis of the effect of adaptive seed set selection on the influence set is added in Section 3.4. In the Appendix, we added more details in the proofs, a thorough exploration of the submodularity exhibited in practice by GLIE, and a comparison of GLIE's results for larger seed sets.

2 Related Work

IM is a well-known problem with numerous solutions of varying complexity [11]. A simple graph degeneracy metric such as k-core [16], which ranks nodes based on their core number, is widely used for identifying influencers. It is considered the most effective structural metric for this purpose because it accounts for influence overlap, unlike degree or PageRank. DegreeDiscount [17] constructs a seed set using degree-based rankings, which are recalculated based on the current seed set and its influence. An even more effective heuristic for the IC diffusion model is PMIA [12]. This method relies on the maximum influence in-arborescence (MIIA) and out-arborescence (MIOA) for each node, representing the union of all maximum influence paths that end or start at that node. The key aspect of this algorithm is the removal of paths with probabilities below a certain threshold. Although providing a substantial speedup, those heuristic methods do not retain theoretical guarantees, in contrast to sketch-based algorithms [18]. The idea of sketch-based approaches is to create several instances of the network that represent varying outcomes of the edge probabilities beforehand and use them to estimate the influence spread of a seed set. The IMM algorithm [19] utilizes sketches to compute influence spread through reverse reachable sets of a node, which include other nodes that can influence it. By generating enough of these sets for random nodes, the optimal seed set is determined by selecting nodes that cover the majority. The frequency of a node's appearance in these sets is proportional to its influence. This method is considered state-of-the-art and surpasses various heuristics [20] while retaining theoretical guarantees.

All aforementioned techniques are purely based on non-learning-based approaches. The first approach to solving combinatorial optimization (CO) using neural networks was based on attention neural networks for discrete structures, PointerNets [21], followed by an architecture that combines PointerNets with an actor-critic training to find the best route for TSP [22]. The first architecture that utilized graph-based learning was S2v-Dqn [23], 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. Although these models are some of the first for solving CO using neural networks, they have been known to perform worse than solvers specializing in a given problem such as IM.

Such specialized neural solvers have been developed for IM based on S2v-Dqn. Specifically, the network dismantling problem, which is akin to IM, was solved with two different variations [24, 25]. Focusing on the more recent and effective one, 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. The main advantage of Finder is that it is trained on small synthetic data, which are easy to construct. On the other hand, it has not been tested with directed graphs and weighted edges, which constitute the majority of real-life IM tasks. Another supervised deep learning approach on IM, Gcomb [26], utilizes a probabilistic greedy method 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 [26] App. G, it takes more than hundreds of minutes and is thus out of our scope. In that sense, both of the IM methods based on neural learning have disadvantages pertaining to either the nature of the evaluation or their generalizability.

Another line of work focuses on addressing the influence prediction as a standalone problem with methods such as DeepIS [27]. 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 similarly to PPNP [28], but based on the IC probability instead of a random walk. DeepIS is a different architecture than 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, recent work on learning contingency-aware IM [29] addresses the possibility of unwilling seeds, which is a specific setting unrelated to our general solution to IM. A different branch of learning-based IM relies on supplementary information such as diffusion cascades [30–32] to derive more effective IM algorithms [33]. This is also diagonal to the current setting, which does not assume any further information from the typical IM. Similarly, for RL-based methodologies that work with non-conventional diffusion models [34], whose objectives are different. We further note here that the majority of the relevant literature on deep learning for combinatorial optimization address small graphs [8, 21, 23, 35], which makes them not applicable to our task. More scalable, unsupervised methods [36] are tailored to specific problems and are non-trivial to adjust for our problem, with the exception of [37], which was found significantly worse than the SOTA algorithm we compare with in [26].

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 three IM methods. The former can act alone as an influence predictor and be competitive with relevant methods, such as Dmp [38] for graphs up to one scale larger than the train set. Dmp achieves an accelerated computation of the influence spread by estimating an upper bound through message passing that is exact for trees. Dmp will be our primary baseline for IE experiments. In detail, our IE method Glie is used to propose: (1) Celf-Glie, i.e., Celf [14] with Glie as influence estimator; (2) Grim, a Q-network that learns how to choose seeds using Glie's estimations and hidden representations; (3) Pun, an adaptive IM method [39] that optimizes greedily a submodular influence spread using Glie's representations.

3 Proposed Methodology

3.1 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 $\mathbf{A} \in \mathbb{R}^{n \times n}$ be the adjacency matrix and $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the features of nodes, representing which nodes belong to the seed set by 1 and 0 otherwise:

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

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

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

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 [11], 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 the case of directed influence graphs, $\mathbf{A}$ 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:

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

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

where the second equality stems from the definition of the weighted cascade and the inequality from the proof in [40], 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 $\mathbf{H}_1 = \mathbf{A} \cdot \mathbf{X}$, and we assume the new seed set S^t to be the nodes influenced in the step $t - 1$, their probabilities are stored in $\mathbf{H}_t$, much like diffusion in discrete time. We can then recompute the new influence probabilities with $\mathbf{H}_{t+1} = \mathbf{A} \cdot \mathbf{H}_t$.

Corollary 1. *The repeated product $\mathbf{H}_{t+1} = \mathbf{A} \cdot \mathbf{H}_t$ computes an upper bound to the real influence probabilities of each infected node at step $t + 1$.*

The proof can be found in Appendix A.1. In reality, due to the existence of cycles, two problems arise with this computation. 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 [38], 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 batch norm and dropout omitted here, and starting from $\mathbf{H}_0 = \mathbf{X} \in \mathbb{R}^{n \times d}$ we have:

$$\mathbf{H}_{t+1} = \mathrm{ReLU}([\mathbf{H}_t, \mathbf{A}\mathbf{H}_t]\mathbf{W}_t). \tag{5}$$

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

$$\mathbf{H}_S^G = \sum_{v \in V} [\mathbf{H}_0^v, \mathbf{H}_1^v, \ldots, \mathbf{H}_t^v]. \tag{6}$$

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

$$\hat{\sigma}(S) = \mathrm{ReLU}(\mathbf{H}_S^G \mathbf{W}_o). \tag{7}$$

Our loss function is MSE as in a standard regression problem. Note that, in the case where $\mathbf{W}_t$ is an untrained positive semidefinite Gaussian random matrix in $[0, 1]$, the representations of each layer $\mathbf{H}_t^v$ would correspond to the upper bound of the influence probability of seed set's t-hop neighbors [38]. This upper bound is not retained once the weights $\mathbf{W}_t$ are trained. In our approach, the parameters of the intermediate layers $\mathbf{W}_t$ are trained such that the upper bound is reduced and the final layer $\mathbf{W}_o$ can 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.

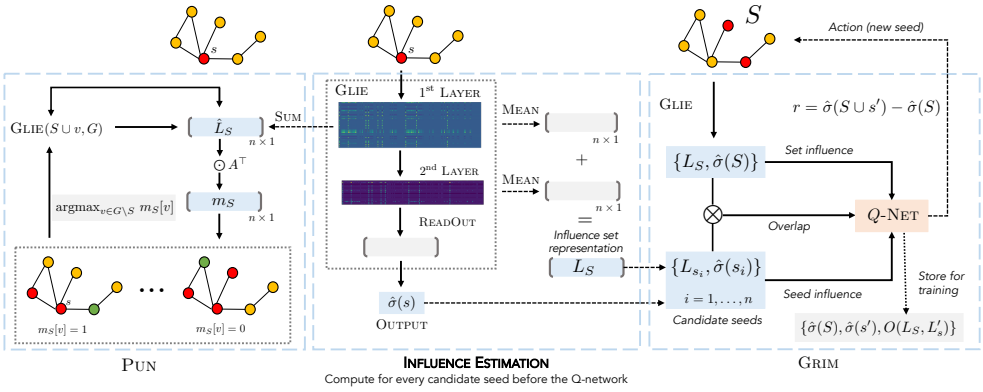


Fig. 1 A visual depiction of the pipeline for GRIM and PUN. The layers of GLIE are depicted by a heatmap of an actual seed during inference time, showing how the values vary through different nodes (columns). The pipeline is broken in two sides, representing PUN on the left and GRIM on the right. Left: PUN utilizes the first hop neighborhood to come up with the proxy for the marginal gain and recomputes the new influence with GLIE$(S \cup v, G)$. Right: The influence set L_S and influence prediction $\hat{\sigma}(S)$ is used as a state for the Q-learning model of GRIM that ranks nodes as next actions.

3.2 CELF-GLIE: Cost Effective Lazy Forward with GLIE

Cost Effective Lazy Forward (CELF) [14] is an acceleration to the original greedy algorithm that is based on the constraint that a seed's spread will never get bigger in subsequent steps. The influence spread is computed for every node in the first iteration and kept in a sorted list. In each step, the marginal gain is computed for the node with the best influence spread in the previous round. If it is better than the previous second node, it is chosen as the next seed because it is necessarily bigger than the rest. This property stems from submodularity, i.e., the marginal gain can never increase with the size of the seed set. If the first node's current gain is smaller than the second previous gain, the list is resorted and the process is repeated until the best node is found. The worst-case complexity is similar to greedy but in practice, it can be hundreds of times faster, while retaining greedy's original guarantee.

In our case, we propose a straightforward adaptation where we substitute the original CELF IE based on MC IC with the output of GLIE. We redirect the reader to Appendix D, where we show that the estimations of GLIE are monotonous and submodular in practice, and hence $\hat{\sigma}$ is suitable for optimizing with CELF. Since we do not prove the submodularity of $\hat{\sigma}$, we can not contend that the theoretical guarantee is retained. CELF-GLIE has two main computational bottlenecks. First, although it alleviates the need to test every node in every step, in practice, it still requires IE for more than one node in each step. Second, it requires computing the initial IE for every node in the first step. We will try to alleviate both with the two subsequent methods.

3.3 The influence set representation

Before moving to the algorithms, we first utilize the activations mentioned above to define the influence set representation $L_S \in \{0, 1\}^n$, which can be computed by adding

the activations of each layer $\mathbf{H}_t$, summing along the axis of the hidden layer size, which following suit with the input is d_t for layer t, and thresholding to get a binary vector:

$$L_S = \mathbb{1}\left\{\sum_{t=0}^{T} \frac{\sum_{i=0}^{d_t} \mathbf{H}_t^i}{d_t} \geq 0\right\}, \tag{8}$$

where T is the number of layers, and $\mathbf{H}_t^i \in \mathbb{R}^{n\times 1}$ is a column from $\mathbf{H}_t$. Note that, although the representations $\mathbf{H}$ are a result of ReLU transformation, the batchnorm layer that is used to stabilize the training, normalizes them and creates negative representations, which is why the thresholding is in 0.

This vector contains a label for each node whose sign indicates if it is predicted as influenced or not. We compute the average representation because d_t varies throughout layers, and since we add all layer's outputs, we need an equal contribution from each layer's dimension to the final output. We utilize this to compute the difference between the influence of the current seed set and the initial influence of each other node.

3.4 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 hindrance and provide adequate performance. We first need to redefine a simpler influence set representation than in Eq. (8). 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:

$$\hat{L}_S = \mathbb{1}\left\{\sum_{i=0}^{d_1} \mathbf{H}_1^i \leq 0\right\} \qquad L'_S = \mathbb{1}\left\{\sum_{i=0}^{d_1} \mathbf{H}_1^i > 0\right\}. \tag{9}$$

L'_S is simpler than L_S defined in Eq. (8) and provides a more rough estimate, but it allows for a simpler influence spread which we can optimize greedily:

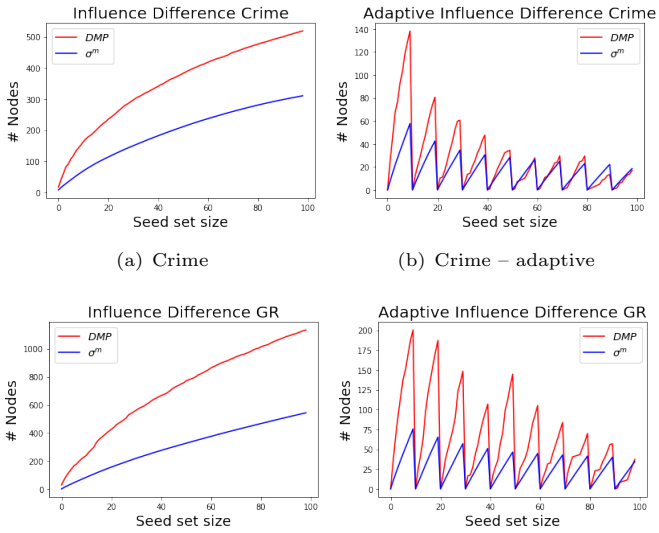
$$\sigma^m(S) = |L'_S|. \tag{10}$$

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,

$$m_S[u] = \sum_{v\in N(u)} \mathbf{A}_{u,v}\hat{L}_v = \mathbf{A}_u^\top \cdot \hat{L}_S \in \mathbb{R}^{n\times 1}. \tag{11}$$

For efficiency, we can compute $m_S = \mathbf{A}^\top \hat{L}$ 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, we prove its monotonicity and submodularity in Appendix A.2.

Theorem 1. *The influence spread σ^m is submodular and monotone.*



(a) Crime (b) Crime – adaptive

(c) GR (d) GR – adaptive

Fig. 2 Difference between DMP influence estimate and σ^m in standard IM and adaptive IM with full feedback every 10 seeds in two datasets (Crime and GR). It is obvious that as the seed set size increases, the difference between the two methods is diminishing in the adaptive setting, justifying the use of σ^m as a faster alternative in the adaptive setting.

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 based 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 proven in the seminal work of [41] 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 seed, 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. These can be observed in the

"Adaptive" plots of Fig. 2 (b), (d) where we employ an AFF every 10 seed and we contrast the aforementioned gap between the upper bound DMP and $\sigma^m(S)$. The plots underline that removing the chosen seeds in an AFF manner is not only theoretically grounded but also addresses the inherent disadvantage of the proposed model that has a diminished scope due to the number of GLIE's layers. This is shown by the diminishing gap DMP and $\sigma^m(S)$ as the seed set increases. We add a corresponding complexity analysis for PUN in Appendix B.

3.5 GRIM: Graph Reinforcement Learning for Influence Maximization

Our effort to make an end-to-end learning approach requires a model that learns how to pick seeds sequentially. The model needs to receive information from GLIE regarding the state (graph and seed set) and decide on the next action (seed). Note that GLIE can not provide a direct estimate of a new candidate's s marginal gain without rerunning GLIE$(S \cup s, G)$, which is what we try to avoid. To this end, we utilize a double Q-network [42], and the model is depicted in Fig. 1 (middle and right part). During the first step, GLIE provides an IE for all candidate seeds and the node with the highest is added to the seed set, similar to GLIE-CELF. We also keep a list of each node's initial influence set L_s. Subsequently, the Q-network produces a Q-value for each node s using as input the estimated influence of the current seed set $\hat{\sigma}(S)$, the initial influence of the node $\hat{\sigma}(s)$, and the interaction between them. The interaction is defined as the difference between their corresponding influence sets $O(S, s) = \sum_{i=0}^{n} \mathbb{1}\{L_S^i - L_s^i \geq 0\}$, as predicted by GLIE. The latter aims to measure how different is the candidate node from the seed set, in order to quantify the potential gain of adding it. The Q-network is called GRIM, and its architecture is composed of two layers:

$$Q(s, S, G) = \text{ReLU}(\text{ReLU}([\hat{\sigma}_S, \hat{\sigma}_s, O(S, s)]\mathbf{W}_k)\mathbf{W}_q), \tag{12}$$

where $\mathbf{W}_q \in \mathbb{R}^{\text{hid}\times 1}$, and hid is the hidden layer size. We utilize a greedy policy to choose the next seed, similar to [23]: $\pi(u|S) = \arg\max_{s\in S} Q(s, S, G)$. Given the chosen action s, the reward is computed based on the marginal gain, i.e., the estimated influence of the new seed set minus the influence of the seed set before the action, as computed by GLIE: $r = \hat{\sigma}(S \cup s) - \hat{\sigma}(S)$.

During training, we use ϵ-greedy to simulate an IM "game" and balance between exploration and exploitation. We store as a train tuple the current state-action embedding $\{\hat{\sigma}(S), \hat{\sigma}(s'), O(S, s')\}$, the new state embedding along with all next possible actions $\{\hat{\sigma}(S \cup s), \hat{\sigma}(s), O(S, s)\}, s \in V$ and the reward r. Throughout the IM, we randomly sample from the memory and train the parameters of GRIM. GLIE is frozen, because apart from providing graph and node embeddings, it is solely used for the computation of the reward—thus, it is not updated. The strategy to pre-train a graph encoding layer on a supervised task and use it as part of the Q-network has proven beneficial in similar works [43]. In our case, though, we found that without the overlap O and with simpler features (i.e., degree), the model performs rather poorly, which means that we can not refrain from computing IE for every node in the first step.

4 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 of the proposed model and baselines can be found in github[1].

4.1 Influence Estimation

For training in 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 Barabási-Albert [44] and Holme-Kim [45] undirected graphs ranging from 100 to 200 nodes and 30 graphs 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 [31], in the absence of extra data, the weighted cascade is considered more realistic than pure random assignments [11].

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 we 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., moving 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

[1] https://github.com/geopanag/learn_im

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 1 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 1, which includes also the large-scale datasets that we will use to evaluate influence maximization, namely *Enron* (EN), *Facebook* (FB), and *Youtube* (YT). The datasets come from open repositories [46] and have been used in similar studies [24, 26].

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 [38]. We could not utilize the influence estimation of UBLF [40] 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 2, along with the average time.

We evaluate the retrieved seed set using the independent cascade, and the results are shown in Table 3. We should underline here that this task would require more than 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 3, 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 3, amounts to 0.11, 0.22 and 0.19 seconds for the three datasets respectively. Finally, we have added a relative error analysis for large seed sets of GLIE in Appendix C.

Table 2 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 3 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**

4.2 Influence Maximization

GRIM is trained on a dataset that consists of 50 random Barabasi-Albert graphs of 500 – 2,000 nodes. It is trained by choosing 100 seeds sequentially, to maximize the reward (delay = 2 steps) for each network. Since the immediate reward corresponds to the marginal gain, the sum of these rewards at the end of the "game" corresponds to the total influence of the seed set. An episode corresponds to completing the game for all 50 graphs; we play 500 episodes, taking roughly 40 seconds each. The exploration is set to 0.3 and declines with a factor of 0.99. The model is optimized using ADAM, as in GLIE. We store the model that has the best average influence over all train graphs in a training episode. To diminish the computational time of the first step in GLIE-CELF and GRIM, we focus on candidate seeds that surpass a certain degree threshold based on the distribution, a common practice in the literature [17, 26].

For comparison, we use a varying set of IM methods described in detail in Section 2, specifically IMM [19], FINDER [24], PMIA [12], DEGREEDISCOUNT [17] and K-CORES [16].

The results for the influence spread of 50, 100, and 200 seeds as computed by simulations of MC ICs can be seen in Tables 4, 5 and 6, while the time costs are shown in Tables 7 and 8. 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 is the second best followed by IMM, which exhibited better performance in smaller seed sets but deteriorates as the seed set size increases. GRIM surpasses significantly the previous reinforcement learning approach, FINDER. On the other hand, while GRIM is faster than GLIE-CELF it is still slower than the PUN. This quantifies the substantial overhead caused by computing the

Table 4 Influence spread computed by 10,000 MC ICs for 50 seeds.

Graph	Glie-Celf	Grim	Pun	K-core	Pmia	DegDisc	Imm	DeepIC-Celf	FINDER
CR	**381**	371	378	263	378	368	375	171	367
GR	**738**	553	700	303	654	556	725	204	635
HI	**1,914**	1,905	1,908	1,407	1,899	1,824	1,589	578	1,904
EN	**11,819**	11,114	11,757	9,796	11,686	11,183	10,698	–	7,133
FB	6,631	5,879	6,329	2,974	6,574	4,489	**6,724**	–	5,649
YT	147,631	148,250	145,796	78,575	145,863	143,161	**148,597**	–	9,152

Table 5 Influence spread computed by 10,000 MC ICs for 100 seeds.

Graph	Glie-Celf	Grim	Pun	K-core	Pmia	DegDisc	Imm	DeepIC-Celf	FINDER
CR	**522**	509	521	455	520	512	516	306	502
GR	**1,102**	997	1,076	421	1,013	919	1,085	418	897
HI	2,307	1,302	**2,308**	2,024	2,291	2,229	2,290	1,020	2,274
EN	**14,920**	14,022	14,912	10,918	14,855	13,808	14,848	-	12,596
FB	**8,710**	7,418	8,409	4,174	5,613	8,247	8,625	-	5,746
YT	189,515	187,808	187,808	89,546	189,746	**194,834**	194,521	-	34,941

Table 6 Influence spread computed by 10,000 MC ICs for 200 seeds.

Graph	Glie-Celf	Grim	Pun	K-core	Pmia	DegDisc	Imm	DeepIC-Celf	FINDER
CR	**661**	650	657	647	656	644	650	501	642
GR	1,617	1,502	**1,626**	701	1,566	1,415	1,617	835	1,286
HI	2,685	2,631	**2,688**	2,540	2,685	2,614	2,668	1,602	2,625
EN	17,601	16,642	**17,614**	13,015	17,534	16,500	17,497	-	17,244
FB	10,981	9,406	10,626	6,434	7,688	10,309	**11,007**	-	10,801
YT	246,439	241,000	244,579	110,409	242,057	236,726	247,178	-	50,435

influence spread of all candidate seeds in the first step. Their time difference amounts to how many more influence estimations Glie-Celf performs in every step compared to Grim, which performs only one. This is more obvious with Pun, which requires only one influence estimation in every step and no initial computation. It is from 3 to 60 times faster than 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 either first or second in the majority of the datasets and this effect becomes more clear as the seed set size increases. 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 is 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.

Table 7 Computational time in seconds.

	100 **seeds**					200 **seeds**				
Graph	GLIE-CELF	GRIM	PUN	IMM	FINDER	GLIE-CELF	GRIM	PUN	IMM	FINDER
CR	1.25	0.91	0.15	**0.13**	0.41	2.00	2.03	0.25	**0.19**	0.41
GR	3.41	0.69	**0.17**	0.57	2.36	4.55	1.79	**0.26**	0.95	2.36
HI	1.20	2.59	**0.17**	0.56	1.01	2.19	0.60	**0.27**	1.29	1.01
EN	5.89	4.85	**0.52**	4.78	9.30	15.49	5.49	**0.97**	10.47	9.30
FB	120.6	100.00	**1.42**	69.90	56.8	287.7	123.95	**3.1**	171.25	56.80
YT	119.00	48.00	**13.20**	55.40	191.00	151.33	100.00	**28.92**	82.13	191.00

Table 8 Computational time of heuristic approaches compared to PUN.

	100 **seeds**				200 **seeds**			
Graph	PMIA	DEGDISC	K-CORE	PUN	PMIA	DEGDISC	K-CORE	PUN
CR	0.13	0.04	**0.04**	0.15	0.21	0.06	**0.04**	0.25
GR	0.70	**0.12**	1.5	0.17	0.80	**0.13**	1.5	0.26
HI	1.24	0.13	**0.12**	0.17	1.36	0.14	**0.12**	0.27
EN	24.83	1.96	2.17	**0.52**	26.74	2.06	2.17	**0.97**
FB	21.2	8.86	10.62	**1.42**	22.77	9.29	10.62	**3.1**
YT	3838.5	52.39	74.91	**13.2**	4006.29	54.38	74.91	**28.92**

Table 9 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

Compared to the heuristics, DEGDISC's seed set quality is significantly worse, and it is faster than PUN in smaller graphs but slower in larger ones because of the overlap between high degree nodes. PMIA provides medium seed set quality but is very computationally inefficient, especially in large graphs where the in-arborescence trees become too costly. K-CORE is worse than DEGDISC as it requires the whole decomposition to take place in order to rank the nodes accordingly. Hence the computational time is independent of the seed set size because the whole ranking already exists and the seed set is retrieved by subsetting list.

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 [27]. 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 require more than 24 GB of GPU RAM. We thus report only the experiments with the smaller networks, where it is not competitive.

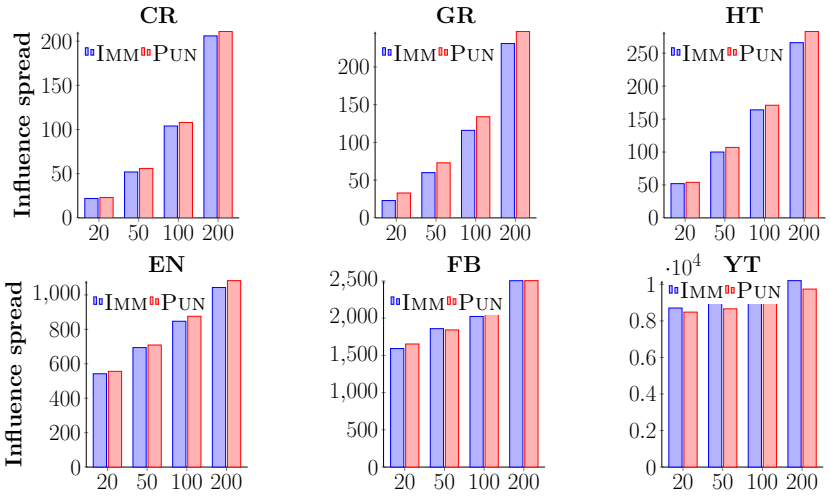


Fig. 3 PUN vs. IMM for IC with uniform $p = 0.01$. This contrasts the two best methods in a setting with fixed influence probabilities instead of the weighted cascade assignment used in the rest of the experiments. This is a common evaluation method in IM literature, and we see that PUN overall outperforms IMM in the majority of datasets and seed set sizes

.

In an effort to generalize the results, we compare IMM and PUN on the same graphs with uniform influence probabilities $p = 0.01$ in Figure 3, as a substitute to the weighted cascade assignment. We observe that PUN outperforms IMM in the majority of the datasets and seed set sizes. Finally, we performed an experiment to compare PUN without the use of GPU for 100 seeds. The results are reported in Table 9. It is visible that GPU provides a substantial acceleration, but PUN remains the faster option even without it. From this we can conclude that overall PUN can be considered a solid, learning-based alternative to IMM. However, it should be noted that this evaluation is limited as it is specially tailored to compare PUN and IMM because these are the top-performing methods in terms of efficiency and seed set quality in our main results. Hence this ablation study is based on the current findings but could be extended to other IM methods.

Overall, from the above, we can contend that PUN provides the best accuracy-efficiency tradeoff from the examined methods. GRIM provides promising results for larger networks and seed set sizes, and it could improve with further training or deeper architectures, given that it outperforms previously proposed RL-based solutions.

5 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 three 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; PUN, a submodular function that acts as proxy

for the marginal gain and can be optimized adaptively, and GRIM, a Q-learning model that learns to retrieve seeds sequentially using GLIE's predictions and representations.

One of the most important takeaways of the current paper is that decomposing the problem of IE and IM allows us to utilize more modular techniques and learning-based methods. We see that GLIE is a fast alternative to DMP, although it does not scale well when the test networks are numerous magnitudes larger in terms of size compared to the train set. However, this does not limit its applicability to IM since the influence spread of a seed set in real-world graphs can be limited to the extended neighborhood of the seed. On the other hand, its significant acceleration allows for methods such as CELF, which can take days or weeks to run on a large-scale network, to finish in less than 2 minutes. It should also be noted that GLIE assumes a rather simple training procedure that requires small simulated graphs (hundreds of nodes) with few seeds (up to 5) in order to be easily reproducible and manageable. A larger training dataset could indeed increase the model's effectiveness, but it requires a significant amount of time to run thousands of Monte Carlo computations to find the training label for each combination of graph and seed set in the training set.

The best IM method can change depending on the task at hand. For instance, if the goal is to maximize seed set quality, it suffices for a method to be simply feasible (i.e., not the fastest and quite interpretable), then GLIE-CELF has clearly outperformed the rest. However, if the aim is to strike the best balance between acceleration and seed set quality, PUN seems to be the best method, followed by the current state-of-the-art IMM. All the methods are dataset-agnostic, meaning they only require training once and can be utilized in any dataset afterward, rendering them efficient for real-world scenarios.

In terms of future work, our overall target is to expand this work with both technical and methodological advancements. On the one hand, we plan to examine the effectiveness of the methods when they are trained with larger networks. On the other hand, we aim to focus on using extraneous information to make the approach more realistic. Specifically for a typical IM algorithm, it is not straightforward to consider the topic of the information shared or the user's characteristics [47]. 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 [48]. We thus deem an experiment with contextual information a natural next step, given a proper dataset. Moreover, we plan to develop techniques that rely on attention-based architectures, similar to [8], which have been shown to outperform RL-based methods in numerous experiments. Finally, we also plan to examine the potential of training online the 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.

Acknowledgements. Supported in part by ANR (French National Research Agency) under the JCJC project GraphIA (ANR-20-CE23-0009-01).

References

[1] Mathew, N., Smith, S.L., Waslander, S.L.: Planning paths for package delivery in heterogeneous multirobot teams. IEEE Transactions on Automation Science and Engineering **12**(4), 1298–1308 (2015)

[2] Touati-Moungla, N., Jost, V.: Combinatorial optimization for electric vehicles management. Journal of Energy and Power Engineering **6**(5), 738–743 (2012)

[3] Duval, A., Malliaros, F.: Higher-order clustering and pooling for graph neural networks. In: ACM International Conference on Information & Knowledge Management (CIKM), pp. 426–435 (2022)

[4] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, (2018)

[5] Joshi, C.K., Laurent, T., Bresson, X.: On learning paradigms for the travelling salesman problem. CoRR (2019)

[6] Veličković, P., Badia, A.P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., Blundell, C.: The clrs algorithmic reasoning benchmark. In: International Conference on Machine Learning (ICML), pp. 22084–22102 (2022). PMLR

[7] Jayalath, D., Jürß, J., Veličković, P.: Recursive algorithmic reasoning. arXiv preprint arXiv:2307.00337 (2023)

[8] Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (ICLR) (2019)

[9] Seyfi, M., Banitalebi-Dehkordi, A., Zhou, Z., Zhang, Y.: Exact combinatorial optimization with temporo-attentional graph neural networks. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD), pp. 268–283 (2023). Springer

[10] Manchanda, S., Michel, S., Drakulic, D., Andreoli, J.-M.: On the generalization of neural combinatorial optimization heuristics. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD), pp. 426–442 (2022). Springer

[11] Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: ACM Conference on Knowledge Discovery and Data Mining (SIGKDD) (2003)

[12] Wang, C., Chen, W., Wang, Y.: Scalable influence maximization for independent cascade model in large-scale social networks. Data Mining and Knowledge Discovery **25**(3), 545–576 (2012)

[13] Tian, Y., Lambiotte, R.: Unifying information propagation models on networks

and influence maximization. Physical Review E **106**(3), 034316 (2022)

[14] Leskovec, J., Krause, A., Guestrin, C., Faloutsos, C., VanBriesen, J., Glance, N.: Cost-effective outbreak detection in networks. In: ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD) (2007)

[15] Panagopoulos, G., Tziortziotis, N., Malliaros, F.D., Vazirgiannis, M.: Maximizing influence with graph neural networks. In: IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM) (2023)

[16] Malliaros, F.D., Giatsidis, C., Papadopoulos, A.N., Vazirgiannis, M.: The core decomposition of networks: Theory, algorithms and applications. The VLDB Journal **29**(1), 61–92 (2020)

[17] Chen, W., Wang, Y., Yang, S.: Efficient influence maximization in social networks. In: ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD) (2009)

[18] Borgs, C., Brautbar, M., Chayes, J., Lucier, B.: Maximizing social influence in nearly optimal time. In: SIAM Symposium on Discrete Algorithms (SODA), pp. 946–957 (2014). SIAM

[19] Tang, Y., Shi, Y., Xiao, X.: Influence maximization in near-linear time: A martingale approach. In: ACM International Conference on Management of Data (SIGMOD) (2015)

[20] Jung, K., Heo, W., Chen, W.: Irie: Scalable and robust influence maximization in social networks. In: IEEE International Conference on Data Mining (ICDM) (2012)

[21] Vinyals, O., Fortunato, M., Jaitly, N.: Pointer networks. In: Advances in Neural Information Processing Systems (NeurIPS) (2015)

[22] Bello, I., Pham, H., Le, Q.V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. CoRR (2016)

[23] Dai, H., Khalil, E.B., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. arXiv preprint arXiv:1704.01665 (2017)

[24] Fan, C., Zeng, L., Sun, Y., Liu, Y.-Y.: Finding key players in complex networks through deep reinforcement learning. Nature Machine Intelligence, 1–8 (2020)

[25] Li, H., Xu, M., Bhowmick, S.S., Sun, C., Jiang, Z., Cui, J.: Disco: Influence maximization meets network embedding and deep learning. arXiv preprint arXiv:1906.07378 (2019)

[26] Manchanda, S., Mittal, A., Dhawan, A., Medya, S., Ranu, S., Singh, A.: Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs.

Advances in Neural Information Processing Systems (NeurIPS) **33** (2020)

[27] Xia, W., Li, Y., Wu, J., Li, S.: Deepis: Susceptibility estimation on social networks. In: ACM International Conference on Web Search and Data Mining (WSDM) (2021)

[28] Klicpera, J., Bojchevski, A., Günnemann, S.: Predict then propagate: Graph neural networks meet personalized pagerank. In: International Conference on Learning Representations (ICLR) (2019)

[29] Chen, H., Qiu, W., Ou, H.-C., An, B., Tambe, M.: Contingency-aware influence maximization: A reinforcement learning approach. In: Internationbal Conference on Uncertainty in Artificial Intelligence (UAI) (2021)

[30] Ling, C., Jiang, J., Wang, J., Thai, M.T., Xue, R., Song, J., Qiu, M., Zhao, L.: Deep graph representation learning and optimization for influence maximization. In: International Conference on Machine Learning (ICML), pp. 21350–21361 (2023). PMLR

[31] Panagopoulos, G., Malliaros, F., Vazirgiannis, M.: Multi-task learning for influence estimation and maximization. IEEE Transactions on Knowledge and Data Engineering (2020)

[32] Panagopoulos, G., Malliaros, F.D., Vazirgiannis, M.: Diffugreedy: An influence maximization algorithm based on diffusion cascades. In: International Conference on Complex Networks and Their Applications (CNA), pp. 392–404 (2018). Springer

[33] Panagopoulos, G., Malliaros, F.D., Vazirgianis, M.: Influence maximization using influence and susceptibility embeddings. In: International AAAI Conference on Web and Social Media (ICWSM), pp. 511–521 (2020)

[34] Chen, H., Wilder, B., Qiu, W., An, B., Rice, E., Tambe, M.: Complex contagion influence maximization: a reinforcement learning approach. In: International Joint Conference on Artificial Intelligence (IJCAI), pp. 5531–5540 (2023)

[35] Prates, M., Avelar, P.H., Lemos, H., Lamb, L.C., Vardi, M.Y.: Learning to solve np-complete problems: A graph neural network for decision tsp. In: AAAI Conference on Artificial Intelligence (AAAI) (2019)

[36] Karalias, N., Loukas, A.: Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. In: Advances in Neural Information Processing Systems (NeurIPS) (2020)

[37] Li, Z., Chen, Q., Koltun, V.: Combinatorial optimization with graph convolutional networks and guided tree search. arXiv preprint arXiv:1810.10659 (2018)

[38] Lokhov, A.Y., Saad, D.: Scalable influence estimation without sampling. CoRR (2019)

[39] Cautis, B., Maniu, S., Tziortziotis, N.: Adaptive influence maximization. In: ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD) (2019)

[40] Zhou, C., Zhang, P., Zang, W., Guo, L.: 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)

[41] Golovin, D., Krause, A.: Adaptive submodularity: Theory and applications in active learning and stochastic optimization. Journal of Artificial Intelligence Research **42**, 427–486 (2011)

[42] Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: AAAI Conference on Artificial Intelligence (AAAI) (2016)

[43] Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J.W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., *et al.*: A graph placement methodology for fast chip design. Nature **594**(7862), 207–212 (2021)

[44] Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)

[45] Holme, P., Kim, B.J.: Growing scale-free networks with tunable clustering. Physical Review E **65**(2), 026107 (2002)

[46] Leskovec, J., Sosič, R.: Snap: A general-purpose network analysis and graph-mining library. ACM Transactions on Intelligent Systems and Technology **8**(1), 1–20 (2016)

[47] Chen, W., Lin, T., Yang, C.: Real-time topic-aware influence maximization using preprocessing. Computational Social Networks **3**(1), 1–19 (2016)

[48] Tian, S., Mo, S., Wang, L., Peng, Z.: Deep reinforcement learning-based approach to tackle topic-aware influence maximization. Data Science and Engineering **5**(1), 1–11 (2020)

[49] James, G., Witten, D., Hastie, T., Tibshirani, R.: An Introduction to Statistical Learning, (2013)

[50] Boyd, S., Boyd, S.P., Vandenberghe, L.: Convex Optimization, (2004)

[51] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Philip, S.Y.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems **32**(1), 4–24 (2020)

A. Proofs

Corollary 1. *The repeated product* $\mathbf{H}_{t+1} = \mathbf{A} \cdot \mathbf{H}_t$ *computes an upper bound to the real influence probabilities of each infected node at step* $t+1$.

Proof.

$$\hat{p}^t(u|S^t) = \mathbf{A}_u \cdot \mathbf{H}_t = \sum_{v \in \mathcal{N}(u) \cap S^t} \hat{p}_v p_{vu} \geq \tag{A1}$$

$$\sum_{v \in \mathcal{N}(u) \cap S^t} p_v p_{vu} \geq 1 - \prod_{v \in \mathcal{N}(u) \cap S^t} (1 - p_v p_{vu}) = p^t(u|S^t) \tag{A2}$$

- (A1) stems from (4) in the manuscript:

$$\begin{aligned} \hat{p}(u|S) = \mathbf{A}_u \cdot \mathbf{X} = \sum_{v \in \mathcal{N}(u) \cap S} \frac{1}{\deg(u)} &= \sum_{v \in \mathcal{N}(u) \cap S} p_{vu} \\ &\geq 1 - \prod_{v \in \mathcal{N}(u) \cap S} (1 - p_{vu}) = p(u|S). \end{aligned}$$

- (A2) can be proved by induction similar to [40]. 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:

$$\begin{aligned} 1 - \prod_{v \in \mathcal{X}} (1 - p_v p_{vu}) &= 1 - (1 - p_x p_{xu}) \prod_{v \in \mathcal{X} \setminus x} (1 - p_v p_{vu}) \\ &= 1 - \prod_{v \in \mathcal{X} \setminus x} (1 - p_v p_{vu}) + p_x p_{xu} \prod_{v \in \mathcal{X} \setminus x} (1 - p_v p_{vu}) \qquad \text{(A3)} \\ &\leq \sum_{v \in \mathcal{X} \setminus x} p_v p_{vu} + p_x p_{xu} \prod_{v \in \mathcal{X} \setminus x} (1 - p_v p_{vu}) \\ &\leq \sum_{v \in \mathcal{X} \setminus x} p_v p_{vu} + p_x p_{xu} = \sum_{v \in \mathcal{X}} p_v p_{vu}. \qquad \text{(A4)} \end{aligned}$$

- In (A2) 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, (A2) stands, and these probabilities are the 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 $\sigma(S)$.

□

Theorem 1. *The influence spread* σ^m *is submodular and monotone.*

For the proof, $P \in \{1\}^{hd \times 1}$ and we define the support function $sup(v) = \{i \in [1, n], v_i \neq 0\}$ [49] as the set of indices of non zero rows in a matrix such as $\mathbf{X}_i$, of layer i. Let R represent ReLU and b_{tr}, st_{tr} the mean and standard deviation computed by the batchnorm.

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

$$sup(\mathbf{X}_j) \supset sup(\mathbf{X}_i) \tag{A5}$$
$$sup(\mathbf{X}_j\mathbf{W}) \supseteq sup(\mathbf{X}_i\mathbf{W}) \tag{A6}$$
$$sup(\mathbf{A}\mathbf{X}_j\mathbf{W}) \supseteq sup(\mathbf{A}\mathbf{X}_i\mathbf{W}) \tag{A7}$$
$$sup(R(\mathbf{A}\mathbf{X}_j\mathbf{W})) \supseteq sup(R(\mathbf{A}\mathbf{X}_i\mathbf{W})) \tag{A8}$$
$$sup\left(\frac{R(\mathbf{A}\mathbf{X}_j\mathbf{W}) - b_{tr}}{st_{tr}}\right) \supseteq sup\left(\frac{(R(\mathbf{A}\mathbf{X}_i\mathbf{W}) - b_{tr})}{st_{tr}}\right) \tag{A9}$$
$$sup(\mathbf{H}_j) \supseteq sup(\mathbf{H}(S_i)) \tag{A10}$$
$$sup(\mathbf{H}_jP) \supseteq sup(\mathbf{H}_iP) \tag{A11}$$
$$|\mathbb{1}_{>0}\{\mathbf{H}_jP\}| \geq |\mathbb{1}_{>0}\{\mathbf{H}_iP\}| \tag{A12}$$
$$|L'_j| \geq |L'_i| \tag{A13}$$
$$\sigma^m(S_j) \geq \sigma^m(S_i) \tag{A14}$$
$$\tag{A15}$$

1. (A5) stems by the definition of X in Eq. (1).
2. (A6) $\mathbf{X}_j$ is a convex hull that contains $\mathbf{X}_i$ [50]. We multiply both sides by a real matrix $\mathbf{W} \in \mathbb{R}^{d \times hd}$ which can equally dilate both convex hulls in terms of direction and norm. This equal transformation cannot change the sign of the difference between the elements of $\mathbf{X}_i$ and $\mathbf{X}_j$ and hence cannot interfere with the support of $\mathbf{X}_j$ over $\mathbf{X}_i$. The statement becomes more obvious for $X \in \{0,1\}^{n \times 1}$ and $\mathbf{W} \in \mathbb{R}^{1 \times 1}$. Note that both can result in zero matrices so we use subset or equal.
3. (A7) $\mathbf{A}$ is a non-negative matrix.
4. (A8) ReLU is a nonnegative monotonically increasing function.
5. (A9) Subtract by the same number and divide by the same positive number.
6. (A10) Definition in Eq. (5).
7. (A11) P is positive.
8. (A12) By definition of the support.
9. (A13) By definition of L'_S.

□

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

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

$$|\mathbf{X}_{ju} - \mathbf{X}_j| = |\mathbf{X}_{iu} - \mathbf{X}_i| \tag{A16}$$
$$\mathbf{A}|\mathbf{X}_{ju} - \mathbf{X}_j| = \mathbf{A}|\mathbf{X}_{iu} - \mathbf{X}_i| \tag{A17}$$
$$|\mathbf{A}(\mathbf{X}_{ju} - \mathbf{X}_j)| = |\mathbf{A}(\mathbf{X}_{iu} - \mathbf{X}_i)| \tag{A18}$$

$$|\mathbf{AX}_{ju} - \mathbf{AX}_j| = |\mathbf{AX}_{iu} - \mathbf{AX}_i| \tag{A19}$$

$$|\mathbf{AX}_{ju}\mathbf{W} - \mathbf{AX}_j\mathbf{W}| = |\mathbf{AX}_{iu}\mathbf{W} - \mathbf{AX}_i\mathbf{W}| \tag{A20}$$

$$R(|\mathbf{AX}_{ju}\mathbf{W} - \mathbf{AX}_j\mathbf{W}|) - 2b_{tr} = R(|\mathbf{AX}_{iu}\mathbf{W} - \mathbf{AX}_i\mathbf{W}| - 2b_{tr}) \tag{A21}$$

$$|R(\mathbf{AX}_{ju}\mathbf{W}) - R(\mathbf{AX}_j\mathbf{W}) - 2b_{tr}| = |R(\mathbf{AX}_{iu}\mathbf{W}) - R(\mathbf{AX}_i\mathbf{W}) - 2b_{tr}| \tag{A22}$$

$$sup(R(\mathbf{AX}_{ju}\mathbf{W}) - R(\mathbf{AX}_j\mathbf{W}) - 2b_{tr}) = sup(R(\mathbf{AX}_{iu}\mathbf{W}) - R(\mathbf{AX}_i\mathbf{W}) - 2b_{tr}) \tag{A23}$$

$$sup(R(\mathbf{AX}_{ju}\mathbf{W} - b_{tr})) - sup(R(\mathbf{AX}_j\mathbf{W}) - b_{tr}) \subseteq sup(R(\mathbf{AX}_{iu}\mathbf{W} - b_{tr})) - sup(R(\mathbf{AX}_i\mathbf{W}) - b_{tr}) \tag{A24}$$

$$sup(\mathbf{H}_{ju}) - sup(\mathbf{H}_j) \subseteq sup(\mathbf{H}_{iu}) - sup(\mathbf{H}_i) \tag{A25}$$

$$\sigma^m(S^j \cup \{u\}) - \sigma^m(S^j) \leq \sigma^m(S^i \cup \{u\}) - \sigma^m(S^i) \tag{A26}$$

(A27)

1. (A19) Distributive property
2. (A20) Similar to multiplication by $\mathbf{A}$.
3. (A24) The norm of the difference is distributed equally, but the right-hand difference has at least the same or more positive elements because the norm of $\mathbf{A}$, which is stochastic, is bounded by V hence $\mathbf{X}_u$ can give up to the same gain to $\mathbf{AX}_j$ and $\mathbf{AX}_i$, the same number b_{tr} is subtracted, and more elements are activated by $\mathbf{X}_j$ then $\mathbf{X}_i$ as shown in Eq. A13.
4. (A25) We skipped dividing by st_{tr} for brevity.
5. (A26) **A**rrive with similar steps as A11 - A14.

□

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

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

This means that m_S gives the node u that improves the biggest number of rows in $\mathbf{AX}_{ju}$ that is not already considered influenced. Since we know from Eq. (A14) that $\mathbf{AX}_{iu} \geq \mathbf{AX}_{iv} \Rightarrow |L'_{iu}| \geq |L'_{iv}|$, 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 (A14):

$$|\mathbb{1}_{>0}\{\mathbf{H}_jP\}| \geq |\mathbb{1}_{>0}\{\mathbf{H}_iP\}|$$

$$|\mathbb{1}_{\leq 0}\{\mathbf{H}_j P\}| \leq |\mathbb{1}_{\leq 0}\{\mathbf{H}_i P\}| \tag{A28}$$

$$\mathbf{A}'_u \hat{L}_j \leq \mathbf{A}'_u \hat{L}_i \tag{A29}$$

$$m_{S_j}[u] \leq m_{S_i}[u] \tag{A30}$$

$$(|L'_j| + m_{S_j}[u]) - |L'_j| \leq (|L'_i| + m_{S_i}[u]) - |L'_i| \tag{A31}$$

$$\sigma^m(S^j \cup \{u\}) - \sigma^m(S^j) \leq \sigma^m(S^i \cup \{u\}) - \sigma^m(S^i) \tag{A32}$$

1. (A14) Complementarity between elements that are ≤ 0 and elements > 0.
2. (A16) Definition in Eq. (9) and multiply with non-negative row u from matrix $\mathbf{A}'$.
3. (A17) Definition in Eq. (11).
4. (A18) Adding and subtracting $|L_j|$ and $|L_i|$.
5. (A19) By definition of σ^m in Eq. (10) and the marginal gain of u, we arrive at submodularity in Eq. (A19).

□

B. Complexity of PUN

In order to estimate the theoretical complexity of PUN, we can break it down into three modules. The complexity of influence estimation (GLIE), the complexity of seed choice, and the complexity of adaptive full-feedback greedy algorithm [41]. Following the notation of the paper, the complexity analysis of GLIE is similar to other graph neural networks and corresponds to $\mathcal{O}(|E|)$ [51]. The complexity of choosing the next seed, which contains the sum in Eq. (9) ($\mathcal{O}(d)$), the message passing to compute m_S in Eq. (11) ($\mathcal{O}(|E|)$), and the complexity of argmax which is $\mathcal{O}(|V|)$ in the worst case which is the first seed. Putting the above together, each iteration will have $\mathcal{O}(d+|E|+|V|)$. Given that the adaptive full-feedback greedy algorithm has the same complexity as greedy, we finally get $\mathcal{O}(|S|(d+|E|+|V|)$.

C. Relative error of GLIE for larger seed sets

To quantify the potential of GLIE for larger seed sets, we sample 9 random seed sets and 1 with the highest degree nodes and compute the error of DMP and GLIE, with the ground truth influence divided by the average influence in Table D1. We see that the error increases differently depending on the dataset, with GLIE outperforming DMP in GR while the reverse happens in CR and HT.

D. GLIE submodularity and monotonicity

In this section, we empirically prove that GLIE's output is submodular and monotonous. For our datasets, we use the seed set retrieved by GLIE-CELF and a random seed set to quantify the differences between subsequent estimations. To be specific, we have a sequence S that represents the seed set and a sequence R that represents the random nodes, with S_j being the seed set up to j^{th} element and s_j being

Graph	Seeds	DMP	GLIE
CR	20	0.005	0.031
CR	50	0.006	0.059
CR	100	0.017	0.152
GR	20	0.161	0.029
GR	50	0.125	0.042
GR	100	0.093	0.082
HT	20	0.010	0.105
HT	50	0.004	0.062
HT	100	0.002	0.113

Table D1 Relative error for diffusion prediction of larger seed sets.

the j^{th} element, and similarly r_j for R. We compute the marginal gain to check for monotonicity:

$$m_{ss} = \hat{\sigma}(S_j \cup s_{j+1}) - \hat{\sigma}(S_j) \tag{C33}$$

$$m_{sr} = \hat{\sigma}(S_j \cup r_{j+1}) - \hat{\sigma}(S_j), \tag{C34}$$

and for submodularity, we have, with $i = j - 1$:

$$s_{ss} = (\hat{\sigma}(S_i \cup s_{j+1}) - \hat{\sigma}(S_i)) - (\hat{\sigma}(S_j \cup s_{j+1}) - \hat{\sigma}(S_j))$$

$$s_{sr} = (\hat{\sigma}(S_i \cup r_{j+1}) - \hat{\sigma}(S_i)) - (\hat{\sigma}(S_j \cup r_{j+1}) - \hat{\sigma}(S_j)).$$

In Figure C1, we plot m and s for some of our datasets. Regarding s, since we require a constant node, we randomly sample one of the seeds s_j and a random node r_j and visualize the sequences of both s with regard to adding them in every step. The values of these functions correspond to nodes, and range from tens to thousands, depending on the datasets. For monotonicity and submodularity, we verify that m and s are always more than zero. Moreover, we see that they decrease with the size of the seed set and that adding a random seed provides worse marginal gains (in monotonicity plots) than adding the chosen seed.

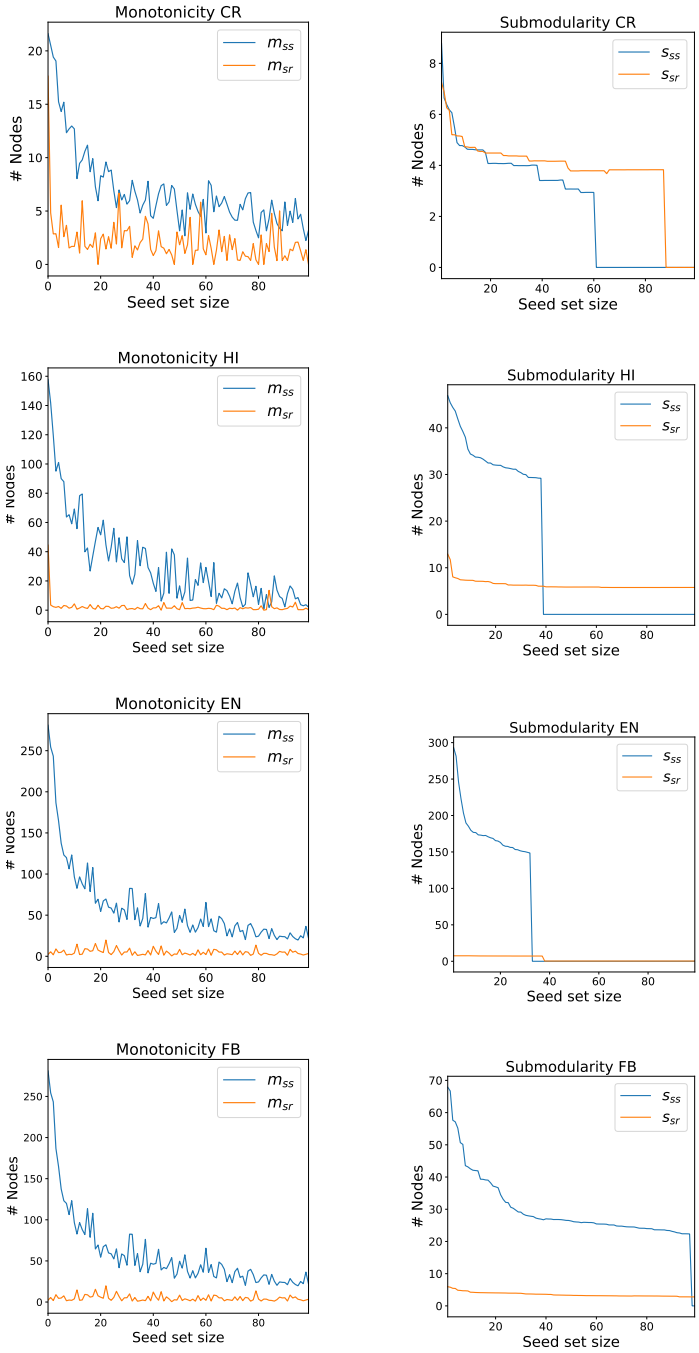


Fig. C1 Monotonicity and submodularity for the examined datasets.