A Worrying Analysis of Deep-RL Methods for Maximum Coverage over Graphs: A Benchmark Study

ABSTRACT

Recent years have witnessed a growing trend toward learning heuristics for combinatorial optimization (CO) problems over graphs from data through deep reinforcement learning (Deep-RL). Maximum Coverage Problem (MCP) and its probabilistic variant on social networks, Influence Maximization (IM) have drawn a lot of attention in this line of research. In this paper, we conduct an in-depth benchmark study to comprehensively examine the effectiveness and efficiency of five recent Deep-RL methods for MCP and IM that were published in top data science venues, namely S2V-DQN [1], Geometric-DQN [2], GCOMB [3], RL4IM [4] and LeNSE [5]. Our benchmark study shows that for MCP, the Lazy Greedy algorithm can beat all Deep-RL methods in all cases. For IM, in most cases, theoretically sound algorithms like IMM and OPIM significantly outperform all deep-RL methods. The cases when Deep-RL works better than IMM and OPIM have a weird phenomenon that the influence spread does not increase as the budget for IM increases. We also reveal by experimental results some common issues of applying these Deep-RL methods for MCP and IM in practice. Finally, we give our thoughts on why Deep-RL may not be a good solution for MCP and IM. Our benchmark study sheds light on potential problems in current deep reinforcement learning research for solving combinatorial optimization problems.

KEYWORDS

max cover, influence maximization, reinforcement learning, approximation algorithms, benchmarking

ACM Reference Format:
. 2018. A Worrying Analysis of Deep-RL Methods for Maximum Coverage over Graphs: A Benchmark Study. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 15 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Maximum Coverage Problem (MCP) and its probabilistic variant on social networks Influence Maximization (IM) aim at finding a set S of k nodes from a given input graph G for maximizing the node coverage and influence spread, respectively. MCP and IM enjoy many important data-intensive applications, ranging from scheduling [6, 7], facility location [8, 9], recommendation systems [10, 11], viral marketing [12], and sensor placement [13]. Therefore, algorithmic techniques for solving MCP and IM have drawn much attention from the data management research community.

Existing studies on MCP and IM have already made very good progress. A simple greedy algorithm can return a solution of $(1-\frac{1}{e})$-approximation guarantee and Feige [14] had demonstrated that achieving a better approximation ratio than $1-\frac{1}{e}$ is unlikely unless $\mathbf{P} = \mathbf{NP}$. The idea of the greedy algorithm can also be applied to IM [12]. Borgs et al. [15] proposed a Reverse Influence Sampling (RIS) method that can achieve $1-\frac{1}{e}-\epsilon$ approximation for IM when equipped with the greedy search. RIS algorithm has a time complexity of $O((m+n)\log n/\epsilon^2)$, which is nearly optimal (up to a logarithmic factor) with respect to network size. Tang et al. further improved the practical efficiency of the RIS-based algorithms [16, 17]. Tang et al. utilized the online approximation bound of submodular functions [13] to return online approximation bounds of RIS-based algorithms. Similar to MCP, any approximation ratio better than $1-\frac{1}{e}$ is impossible if $\mathbf{P} \neq \mathbf{NP}$.

Despite the good theoretical progress in MCP and IM, recently, there is a growing trend of applying Deep Reinforcement Learning (Deep-RL) to learn heuristics from data for solving combinatorial optimization problems like MCP and IM [1–5]. These Deep-RL methods hope to learn the "distribution" of real data to obtain a learned heuristic that can outperform theoretically sound algorithms in practice. In these studies, graph neural nets are often used to learn embeddings of nodes first, and then reinforcement learning building blocks such as Q-learning are employed to approximate the objective function. Empirical studies of these works show that these deep-RL methods sometimes can achieve better solutions than theoretically sound algorithms for MCP or IM.

However, in these Deep-RL methods, one common treatment that the time for training the Deep-RL model is not counted in the computation time concerns us. As training is similar to pre-processing, in rigorous data management research the training/pre-processing time should be counted in an amortized way in the query computation time. Totally ignoring the training time is only reasonable when the training is performed on a dataset of size independent of the following queries, making the training time a constant theoretically. However, in such a case, constant time training/pre-processing probably can only boost very limited queries. As a result, we wonder if these Deep-RL methods are as effective and efficient as claimed by the authors.

To this end, in this paper, we conduct a thorough benchmark study to comprehensively verify the effectiveness and efficiency of recent Deep-RL methods [1–5]. We adopt 18 benchmark datasets and test all commonly used edge-weight models for the IC influence model. Besides, we also deeply examine some potential issues of applying t

Summary of contributions

(1) We propose a benchmarking framework to compare Deep RL and algorithmic solvers in a common environment (figure 1),

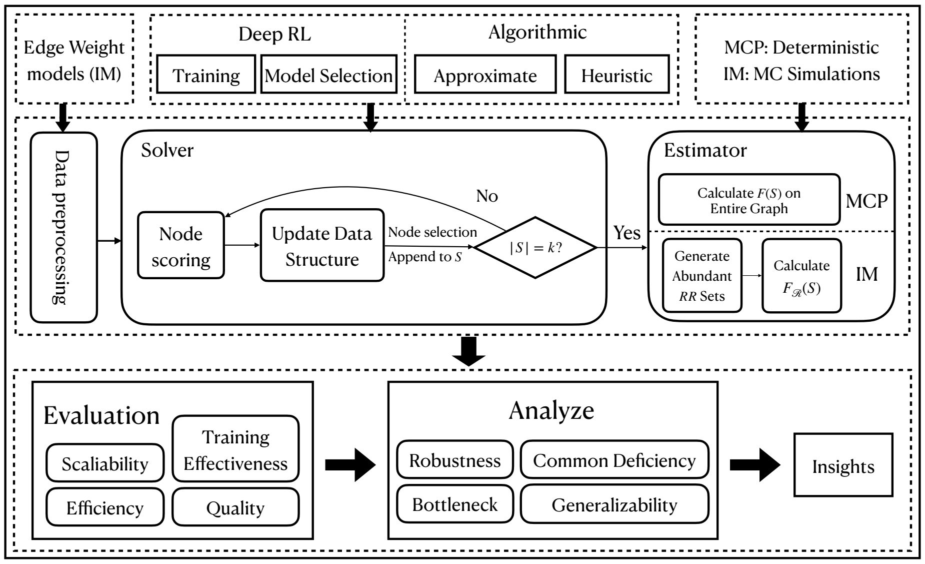


Figure 1: Benchmarking Framework: Datasets are first preprocessed specifically for each solver, and edge weight models are used to define propagation probability in IM. Deep-RL solvers are trained and validated before application. Algorithmic solvers include approximate and heuristic ones. Solvers find a solution set S of k nodes, and then a public estimator is used to calculate the coverage. In IM, the estimator generates a large amount of RR sets to estimate the influence spread $F_{\mathcal{R}}(S)$ while calculating coverage $F(S)$ on the input graph directly in MCP. Analysis of the results provides insights.

presenting empirical comparisons over a wide range of real-world datasets and synthetic datasets under various settings, which has not been addressed in the literature and debunks the claims of superiority of Deep RL methods in solving CO problems.

(2) We propose a strong baseline algorithm Lazy Greedy solving MCP, which adapts lazy forward idea from CELF [13]. It is up to 5 orders of magnitude faster than Normal Greedy while providing $(1 - \frac{1}{e})$-approximation.
(3) Deep RL methods are heuristic, and thus their solutions are not theoretically guaranteed. We not only reproduce the results of Deep RL methods S2V-DQN [1], GCOMB [3], RL4IM [4], LeNSE [5], Geometric-DQN [2], but also perform extensive experiments comparing them with algorithmic methods on solving MCP and IM to validate their performance more comprehensively.
(4) We analyze common limitations of Deep RL methods to further illustrate the essence of their deficiency.

Due to limited space, we skip some experimental results, including the reproduced results of RL4IM and Geometric-DQN, detailed experimental analysis of the noise predictor in GCOMB, and details of improving the efficiency of LeSNE. Those experimental results can be found in the Appendix of the full version[1] of our paper.

[1]https://anonymous.4open.science/r/MCPBenchmark-B2BF

2 PRELIMINARIES

This section provides a review of two important problems: the Maximum Coverage Problem (MCP) and its variant in social networks, Influence Maximization (IM).

2.1 Problem Formulations

The Maximum Coverage Problem can be formally defined as below:

Problem 1 (Maximum Coverage Problem (MCP) on Graphs). *Given a graph $G = (V, E)$, let $f(S) = \frac{|X_S|}{|V|}$ be the coverage function where $X_S = \{j | j \in S \vee \exists (i, j) \in E, i \in S\}$. For a given budget k, we aim at selecting a set $S \subseteq V$, $|S| = k$ to maximize the coverage $f(S)$.*

The Influence Maximization (IM) problem can be viewed as a maximum coverage or reachability problem on a probabilistic graph. IM primarily revolves around modeling the dynamics of influence diffusion within a network. In this particular paper, our focus is on the Independent Cascade (IC) model [12], which is widely recognized as the most popular influence diffusion model in the literature. **Independent Cascade (IC) Model** [12] Given a weighted and directed graph $G = \langle V, E, W \rangle$, we assume that the edge weight p_{uv} on an edge (u, v) represents its *influence probability* p_{uv}, i.e., $0 \leq puv = w_{uv} \leq 1$). An influence diffusion starts from a **seed set** $S \subseteq V$. S_i denotes the set of nodes that are active in time-step i, $i \in \mathbb{N}$, $S_0 = S$. Each newly activated vertex u in the previous step $i - 1$ has a single chance at the current step i to influence its inactive out-neighbor v independently with a probability p_{uv}. The diffusion process continues until there are no further activations, i.e., $S_t = S_{t-1}$. The **influence spread** of S, $I(S)$, is the expected number of active nodes at the end of the diffusion initiated from S.

As IC model is the most popular influence model, we employ IC model to formulate the Influence Maximization (IM) problem.

Problem 2 (Influence Maximization (IM)). *Given a social network $G = (V, E, W)$, a budget k, we want to select a set $S \subseteq V$, $|S| = k$ such that the influence spread $I(S)$ under IC model is maximized.*

2.2 Characteristics of MCP and IM

Intractability. Both MCP and IM problems are known to be **NP**-hard [12]. One important characteristic of both MCP and IM is that their objective functions exhibit **monotonicity** and **submodularity** [12]. These properties enable the development of efficient approximation algorithms that guarantee a performance ratio of at least $1 - \frac{1}{e}$ [12, 17]. Furthermore, it has been established that achieving an approximation ratio higher than $1 - \frac{1}{e}$ is not possible unless the complexity classes **P** and **NP** are equal [12].

Influence Spread Estimation. In addition to the **NP**-hardness, another challenge of IM is that computing the influence spread under the Independent Cascade (IC) model is known to be **#P**-hard [18]. Polling [15] serves as an efficient method for estimating influence spread while maintaining a $1 - \frac{1}{e} - \epsilon$ approximation guarantee for IM with high probability. In the polling method, we sample a so-called Reverse Reachable (RR) set [16] as follows. Each edge (u, v) in the graph is independently preserved with a probability p_{uv}. The RR set for vertex v consists of all nodes that can reach v through the preserved edges. To estimate the influence spread, the Polling method involves sampling a specified number, denoted as M, of random RR sets. Each RR set is constructed as described above. The quantity $D(S)$ represents the number of RR sets that contain at least one vertex from a given set S. It has been shown that $\frac{|V|D(S)}{M}$ provides an unbiased estimation of the influence spread $I(S)$ [15, 16]. In practice, by setting a large sample size M, it is possible to accurately estimate the influence spread $I(S)$.

2.3 Edge Weights in IM

The Influence Maximization (IM) problem takes as input a weighted graph, where the edge weights p_{uv} represent the direct influence from node u to node v. However, collecting relevant data to accurately learn these edge weights can often be challenging. In the IM literature, edge weights are typically set using various predefined models to address this issue, as listed below:

Tri-valency (TV) Model. The weight of an edge is chosen randomly from a set of weights {0.001, 0.01, 0.1}.

Constant (CONST) Model. In this model, The weight of an edge set as a constant value, e.g., 0.1.

Weighted Cascade (WC) Model. The weight of the edge (u, v) is set as $\frac{1}{|N^{in}(v)|}$, where $N^{in}(v)$ is the set of v's in-neighbors.

Learned (LND) Model. When we have historical data about user interactions, we may learn edge weights. One representative method to learn the edge weights is the Credit Distribution Model [19].

Remark. Note that for theoretically sound algorithms [16, 17, 20], the specific choice of edge weight setting does not affect the effectiveness of the algorithms. However, in our benchmark study, we observe that the choice of edge weight setting can impact the performance of these Deep-RL heuristics.

3 ALGORITHMS REVISITED

In this section, we revisit the deep-RL methods benchmarked in the experiments and Lazy Greedy for MCP.

3.1 Deep RL Methods

We summarize the general procedure of Deep RL methods solving coverage problems over graphs in Fig. 2. A GNN encoder is first used to learn the embedding of a graph, followed by a final layer that calculates the scores of nodes by the embedding matrix. The feature f_G is set as either the nodes' embedding matrix or the nodes' score array. Based on the scores of nodes, a collection of seed sets are randomly generated and then used to train the RL model. The environment uses f_G to set the state and calculate the reward for an RL agent to find the best solution set S in an iterative manner by a learned policy, where $|S| = k$. We illustrate how each method examined in this paper implements this general pipeline as follows.

S2V-DQN [1]. It first maps the input graph G into node embeddings via Struc2Vec [21]. A deep Q-network is learned to construct a solution set that maximizes the coverage based on the node embedding.

RL4IM [4]. Unlike S2V-DQN, RL4IM utilizes Struc2Vec to encode graph-level features rather than node-level ones. The input graph G is randomly selected from a set of training graphs $\mathcal{G}$ in each iteration. The reward for each action is calculated by Monte Carlo (MC) simulations on the fly. Two novel tricks, namely state abstraction and reward shaping, are used to improve performance.

Geometric-DQN [2]. Starting from a randomly initialized seed set S, Geometric-DQN iteratively enlarges a subgraph g by a random walk over the input graph G. Then DeepWalk [22] is used to generate node features and a GCN encoder excavates the structural information. Lastly, a deep Q-network (DQN) selects the node with the highest Q value and adds it into S, making it possible to expand g to cover more potentially influential nodes.

GCOMB [3]. Rather than unsupervised learning, GCOMB trains a GraphSAGE network in a supervised manner, where the label of each node is generated by calculating its marginal cover (influence spread) based on a probabilistic greedy method. A deep Q-network (DQN) then finds a solution set that might maximize the cover (influence spread). Node pruning techniques are also adopted to remove noisy nodes for reducing the search space, which makes GCOMB scalable to large graphs.

LeNSE [5]. Similar to Geometry-DQN, LeNSE aims to find a smaller optimal subgraph containing the optimal solution set. It generates multiple subgraphs with a fixed number of nodes, categorizes them into labels based on the likelihood of containing the optimal solution, and trains a GNN to cluster similar subgraphs. By leveraging both node-level and graph-level features generated by GNN, a DQN constructs the subgraph iteratively. Finally, a pre-existing heuristic is applied to discover a solution set from the constructed subgraph.

3.2 Concerns on Training in Deep-RL Methods

In all aforementioned Deep-RL methods [1–5], the time for training Deep-RL models is not counted in the computation time. If we treat training deep-RL models as **pre-processing**, not counting the training time is problematic. In rigorous Databases research, the pre-processing time should be amortized into the computation time of the following queries.

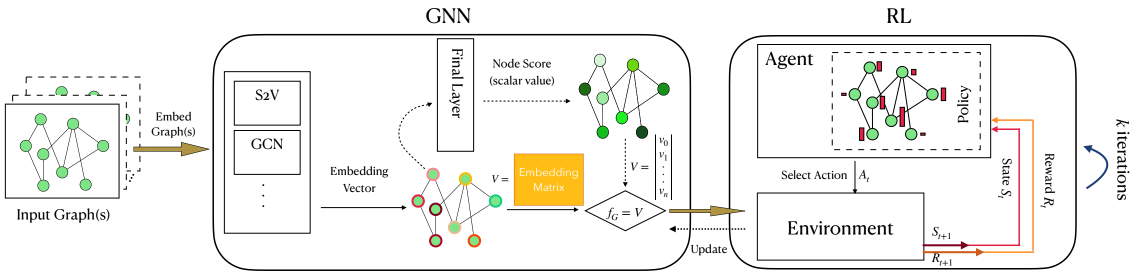


Figure 2: General pipeline of Deep RL methods for solving MCP/IM over Graphs

One potential excuse for not counting the training time is that the training dataset is of a fixed size which is independent of the following MCP or IM queries. This is actually adopted in all the Deep-RL studies for MCP and IM [1–5]. However, note that theoretically, such a pre-processing/training step takes a **constant** time but the trained model is expected to boost all queries. This obviously is too good to be true.

One may argue that a trained Deep-RL model should be used to answer MCP or IM queries with input graphs following the same distribution as the training graphs. However, what does "**same distribution**" mean for graphs as input for MCP or IM? Can we use some easy-to-compute statistics of graphs to decide whether a testing graph is suitable for the trained Deep-RL model before running the model?

Another issue of the training of these Deep-RL methods is that the size of training data is **independent** to the future MCP or IM queries. Even though magically we can guarantee that the training graphs and testing graphs follow the "same distribution", according to basic statistical learning theory [23], the size of the training set has a crucial impact on the generalizability of the trained model. Therefore, a better way is to vary the size of the training data based on the MCP/IM queries we want to answer in the future.

Motivated by our above concerns, in this paper, we conduct a thorough benchmark study to comprehensively examine the effectiveness and efficiency of the recent Deep-RL methods for MCP and IM [1–5].

3.3 Lazy Greedy for MCP

The idea of Lazy Greedy (also known as CELF [13]) is widely adopted in IM algorithms to accelerate the greedy search. In our benchmark study, we also apply Lazy Greedy for MCP. Specifically, we do not update the degrees of the rest nodes when adding a new node to the solution set S. Instead, we maintain a priority queue for all nodes and every time when we retrieve the head of the queue we check if the head node's degree is up-to-date. If not, we subtract its degree according to the current solution set S. Such a lazy greedy algorithm can still guarantee $(1 - \frac{1}{e})$-approximation for MCP. However, such a simple algorithm was neglected in studies of Deep-RL methods for MCP [1, 5].

4 BENCHMARKING

In this section, we report the results of our benchmark study. All the experiments were run on a server with 16 Intel i7-11700KF 3.60GHz cores, 64G RAM, 1 NVIDIA GeForce RTX 3090 24G GPU. Our source code and datasets can be found at https://anonymous.4open.science/r/MCPBenchmark-B2BF.

4.1 Experimental setup

Datasets. We adopt in total 20 real-world benchmark datasets that are widely used in existing studies. Basic information about the 20 datasets can be found in Table 1. For MCP, we conducted experiments on 12 datasets. For IM, we performed more extensive experiments on 18 datasets listed in Table 1 under the edge weight model TV, CONST and WC. As for the LND model, we generated the influence probabilities using the credit distribution model [19] on Flixster, Digg and Twitter. Stack dataset collected from [3] is also included in our experiments. Meanwhile, due to the poor scalability, we specifically tested RL4IM on small synthetic graphs using the power-law model [24] and tested Geometric-DQN on the small datasets mentioned in [2].

We followed the instruction mentioned in [1–5] to train deep-RL models for MCP and IM. For MCP, S2V-DQN, GCOMB and LeNSE were trained on BrightKite and tested on other datasets. For IM, we trained GCOMB on a subgraph sampled out of Youtube by randomly selecting 15% edges and tested the model on other datasets under all edge weight models except for LND (as there are no action logs). In addition to training models on real datasets, we also trained RL4IM on synthetic graphs as Chen *et al.* [4] did and took the best result obtained by different training settings.

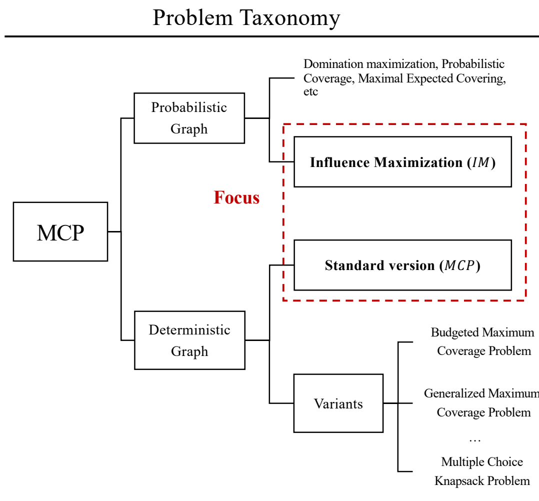
Implementation and hyperparameters setting. We implemented Degree Discount heuristic [25] and Lazy Greedy algorithm [13]. For the other methods, we used the code shared by the authors. All the parameters are set as the same as the ones suggested in [1–5, 17]. We set ϵ as 0.5 and 0.1 for IMM and OPIM[2], respectively.

4.2 Performance on MCP

In this section, we benchmark Normal Greedy, Lazy Greedy, S2V-DQN, GCOMB and LeNSE in MCP. Fig. 4 and Fig. 5 show the results.

Effectiveness. Both Normal Greedy and Lazy Greedy provide an $(1 - \frac{1}{e})$-approximation solution, and the results shown in figure 4 verify that these two methods perform equivalently well. GCOMB performs better than S2V-DQN, sometimes approaching or reaching the level of the greedy algorithms, which is consistent with the

[2]In GCOMB [3], ϵ was set to 0.05 for OPIM. However, setting $\epsilon = 0.1$ still can guarantee a meaningful approximation ratio and we will show that OPIM can still return high-quality solutions in such a case.


Problem Taxonomy
MCP
Probabilistic Graph
Deterministic Graph
Domination maximization, Probabilistic Coverage, Maximal Expected Covering, etc
Influence Maximization (IM)
Focus
Standard version (MCP)
Variants
Budgeted Maximum Coverage Problem
Generalized Maximum Coverage Problem
...
Multiple Choice Knapsack Problem

Methods & Performance

Name	Problem	Quality	Memory	Efficiency	Robustness
Deep RL-Based Methods					
S2V-QN (2017)	*MCP*	★★	★★	★★	★★★
Geometric-DQN (2020)	*IM*	★	★	☆	★
GCOMB (2020)	*MCP*	★★★★	★★★	★★★	★★★★
	IM	★★	★★★	★★☆	★★
RL4IM (2021)	*IM*	★★★	★★	★★	★★★
LeNSE (2022)	*MCP*	★	★	★	★
	IM	★★★★	★	★	★★★
Traditional Algorithmic Methods					
Normal Greedy	*MCP*	★★★★★	★★★★★	★★	★★★★★
Lazy Adaptive Greedy	*MCP*	★★★★★	★★★★★	★★★★★	★★★★★
IMM (2015)	*IM*	★★★★★	★★★	★★★☆	★★★★★
OPIM (2018)	*IM*	★★★★★	★★★★	★★★★	★★★★
Degree Discount (2009)	*IM*	★★★	★★★★★	★★★★★	★★★★★
Single Discount (2009)	*IM*	★★★	★★★★★	★★★★★	★★★★★

Figure 3: Taxonomy of the Problems and Rating Scale

dataset	n	m	avg. degree	category
Damascus	3K	7.7K	2.5	Tweets
Israel	3.6K	8.3K	2.3	Tweets
CondMat	23K	186K	8.08	Collaboration
Digg∗	26K	200K	14.9	Social
Flixster∗	96K	484K	10.1	Social
BrightKite	58K	214K	29.1	Social
Gowalla	200K	846K	8.8	Social
Twitter∗	300K	2.1M	13.3	Tweets
DBLP	310K	1.0M	14.4	Collaboration
Amazon	334K	925K	5.5	e-Commerce
Higgs	456K	14.9M	65.2	Tweets
Youtube	1.1M	2.7M	4.99	Social
Pokec	1.6M	19.9M	37.5	Social
Skitter	1.7M	11.1M	13.19	Traceroutes
Wiki Topcats	1.8M	28.5M	31.83	Hyperlinks
Wiki Talk	2.4M	5.0M	4.2	Communication
Stack∗	2.6M	36.2M	27.9	Q&A
Orkut	3.1M	104.4M	76.3	Social
Live Journal	4.8M	69.0M	28.5	Social
FriendSter	65.6M	1.8B	27.7	Social

Table 1: Summary of datasets (K=10^3 M = 10^6, B = 10^9). Datasets marked with ∗ are only used in LND edge weight model.

results of [3]. However, GCOMB performs significantly worse than the greedy algorithms on some datasets, indicating that GCOMB can not generalize well to these datasets. On the other hand, LeNSE provides very poor-quality solutions in most cases.

Efficiency. Figure 5 shows that GCOMB is much faster than S2V-DQN and LeNSE, and is about one order of magnitude and sometimes even more than two orders of magnitude faster than Normal Greedy when the budget is small, which accords with that in [3]. GCOMB's runtime fluctuates wildly rather than steadily as expected because the number of good nodes predicted by the nodes pruner varies drastically[3]. LeNSE is even much slower than Normal Greedy.

Lazy Greedy runs more than one order of magnitude faster than GCOMB when the budget is small. As the budget increases, it is even up to two orders of magnitude faster. Note that the runtime only takes inference time into account, without considering extra time consumption in preprocessing and training phases. Even though deep-RL methods are given some unfair advantages in efficiency comparison, Lazy Greedy still beats all deep-RL methods.

Memory Usage. We report the memory usage of each method on several representative datasets in Table 2. Compared to greedy algorithms, deep-RL methods consume significantly more memory.

Summary. Combining all the above results, we find that in our experiments for MCP, Lazy Greedy dominates all deep-RL methods on effectiveness, efficiency, and memory usage.

4.3 Performance on IM

In this section, we test Deep-RL methods GCOMB, RL4IM, and Geometric-DQN in our experiment. Meanwhile, the experiment also benchmarks algorithms IMM, OPIM, Degree Discount, and Single Discount. We ran all algorithms under four edge weight models, namely CONST, TV, WC and LND. Worth noting that none of the deep-RL studies [1–5] have tested their methods under the WC model, which arguably is the most popular edge weight model for IC model in IM literature.

As RL4IM often crashes on large datasets due to the high demand for memory and too long running time, we only test RL4IM on 6 datasets. Geometric-DQN has a similar issue that it is too slow to solve a problem on large datasets due to its graph exploration policy. Thus we only report Geometric-DQN's result on BrightKite given small budgets. We also test these two methods on datasets mentioned in their papers and reproduce similar results[4]. The effectiveness and efficiency of each algorithm are reported in Fig. 6~12.

Effectiveness. Observe Fig. 6, Fig. 8, Fig. 10 and Fig. 12. In line with the findings presented in [3], GCOMB works as well as IMM on Youtube under TV and on Stack under LND, but it is slightly worse than IMM on Youtube under CONST. RL4IM exhibits greater stability than GCOMB and provides a more effective solution under

[3]In the full version of our paper, we report more detailed experimental results on this matter in Appendix.

[4]The results can be found in the Appendix of the full version.

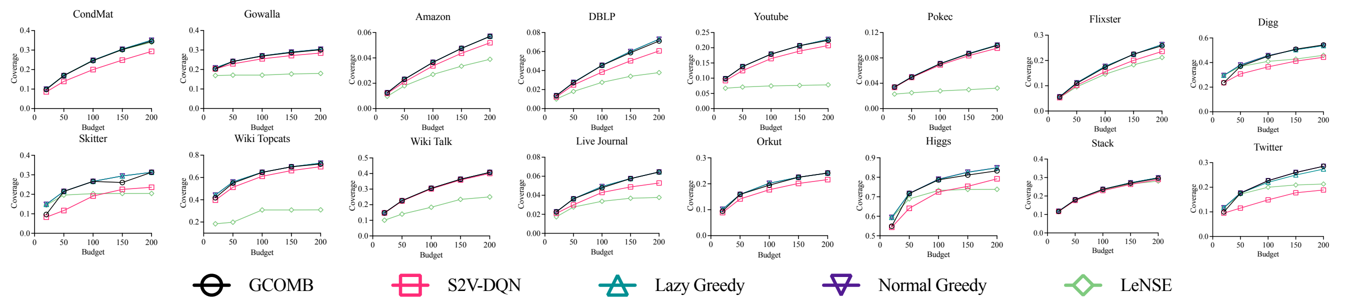


Figure 4: MCP: Coverage curve

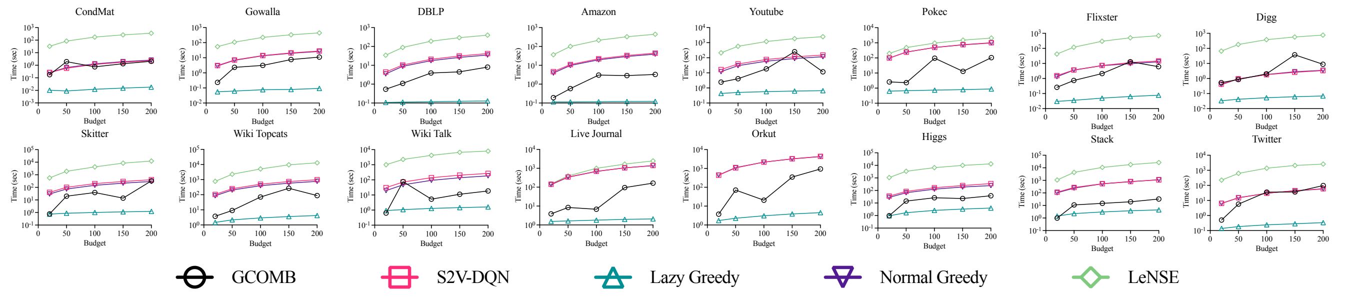


Figure 5: MCP: Runtime curve

the CONST model. LeNSE emerges as the most effective learning method in most cases, but it still falls short compared to algorithmic algorithms. These methods demonstrate limited effectiveness across different datasets, indicating poor generalizability. All the cases when deep-RL methods are as effective as IMM are those where **the influence spread does not increase as the budget increases**. Additionally, when using the WC or LND model, deep-RL methods consistently perform significantly worse than IMM.

We find that IMM is still the most effective algorithm and OPIM's effectiveness is similar to IMM. Especially under WC model and LND model, the performance gap between theoretically sound algorithms (IMM and OPIM) and deep-RL methods is very significant. Moreover, the discount algorithms, though they are heuristics, outperform deep-RL methods in most cases. Such experimental results demonstrate that compared to traditional IM algorithms, **the effectiveness of deep-RL methods is questionable**.

Efficiency. First, note that we give unfair advantages to deep-RL methods by ignoring their training time. However, traditional IM algorithms (IMM, OPIM, DegreeDiscount, and SingleDiscount) still are much faster than Deep-RL methods in most cases, except for the cases when the influence spread is insensitive to the budget. When the influence spread hardly increases as the budget increases, there are actually too many solution sets having very similar influence spread. To distinguish from these abundant highly-similar solution sets, theoretically sound algorithms like IMM and OPIM need to generate many RR sets and thus, they are slow in such a situation.

With the help of node pruning techniques, GCOMB can sometimes be orders of magnitude faster than IMM. However, the node pruner cannot even guarantee that the search space for a small budget is smaller than that of a large budget. The evidence of this can be found in Fig. 7, where the running time of GCOMB is often not monotone w.r.t. the budget. Such instability of the node pruner makes GCOMB much slower than IMM in some cases.

Unlike GCOMB, which uses a simple linear interpolation method to estimate pruning thresholds, LeNSE iteratively constructs subgraphs in a Markov decision process manner to achieve pruning effects. This makes LeNSE significantly require more time compared to other methods, and can not find solutions for large datasets like Orkut and Stack in limited time.

RL4IM finds the solution set in the entire graph and Geometric-DQN has a costly real-time graph exploration policy during the inference process, making these two methods not scalable to large graphs.

Memory Usage. The lower part of table 2 lists the peak memory usage of algorithms in the IM experiments. Deep RL methods of different network designing consume vastly different amounts of memory. RL4IM and Geometric-DQN need much less memory than GCOMB and LeNSE but take a considerably longer time to find a solution, resulting in poor scalability. IMM usually needs to generate a large number of RR sets under CONST (CO) and TV models on datasets like Pokec, Wiki Talk, and Wiki Topcats when given a huge budget. Nevertheless, OPIM can be an alternative in such cases.

Summary. Based on the above experimental results for IM, we find that except for the weird cases when the influence spread is insensitive to the increasing budget, traditional IM algorithms outperform deep-RL methods in effectiveness, efficiency, and memory usage.

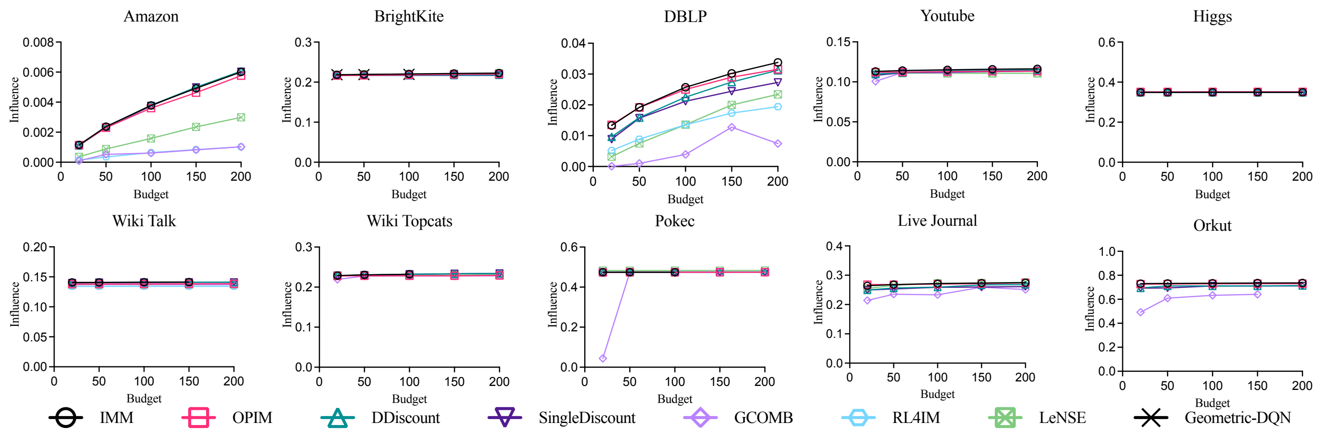


Figure 6: IM: Influence curve under CONST. Influence represents the proportion of active nodes in the graph.

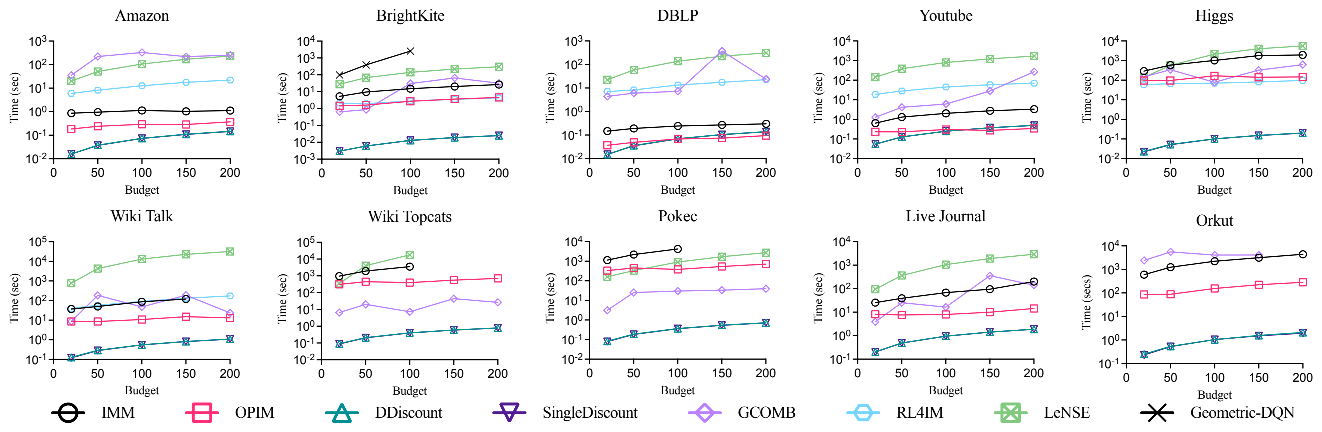


Figure 7: IM: Runtime curve under CONST

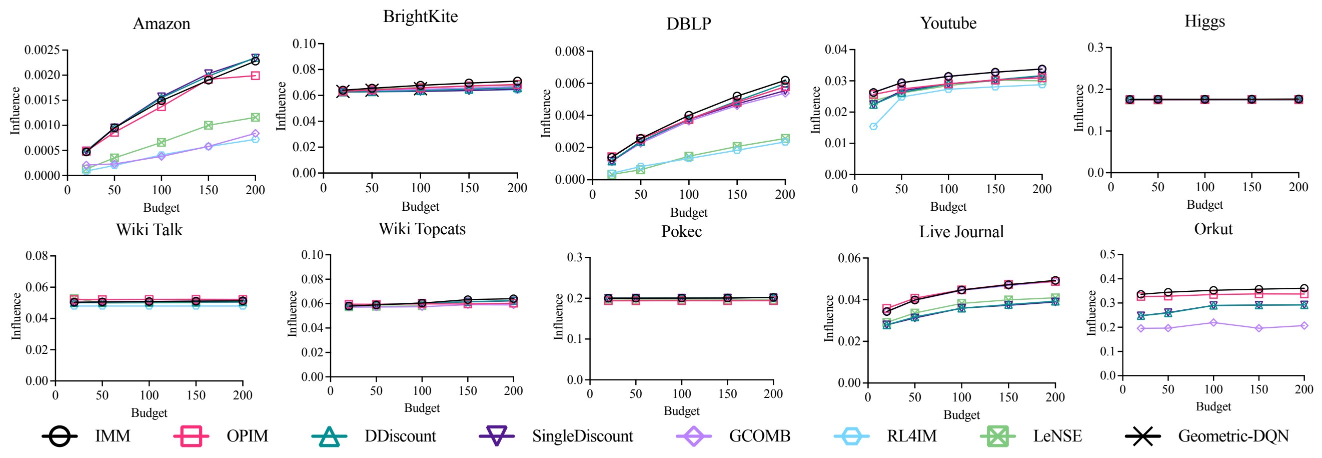


Figure 8: IM: Influence curve under TV. Influence represents the proportion of active nodes in the graph.

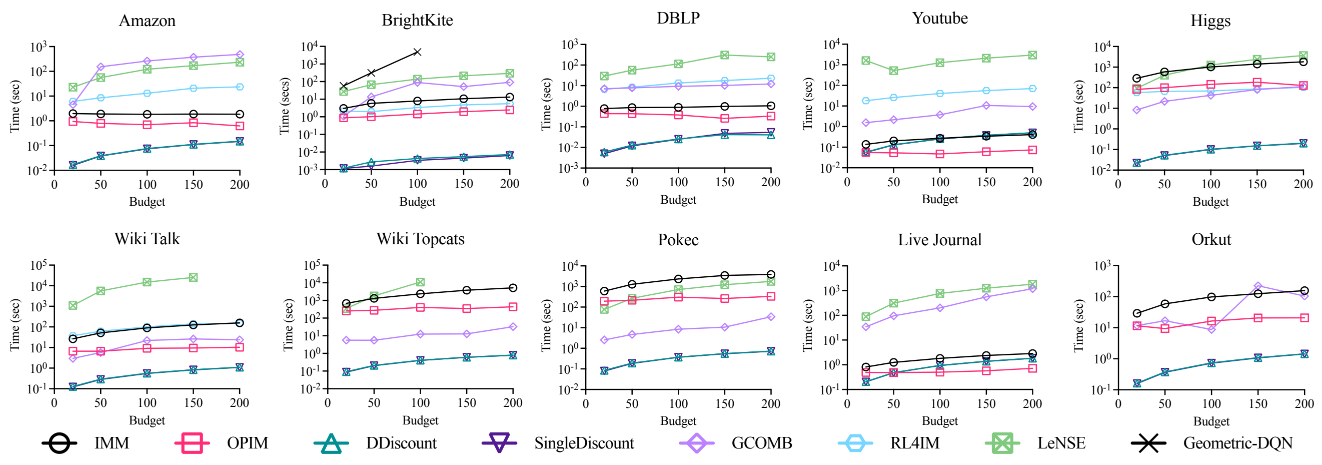


Figure 9: IM: Runtime curve under TV

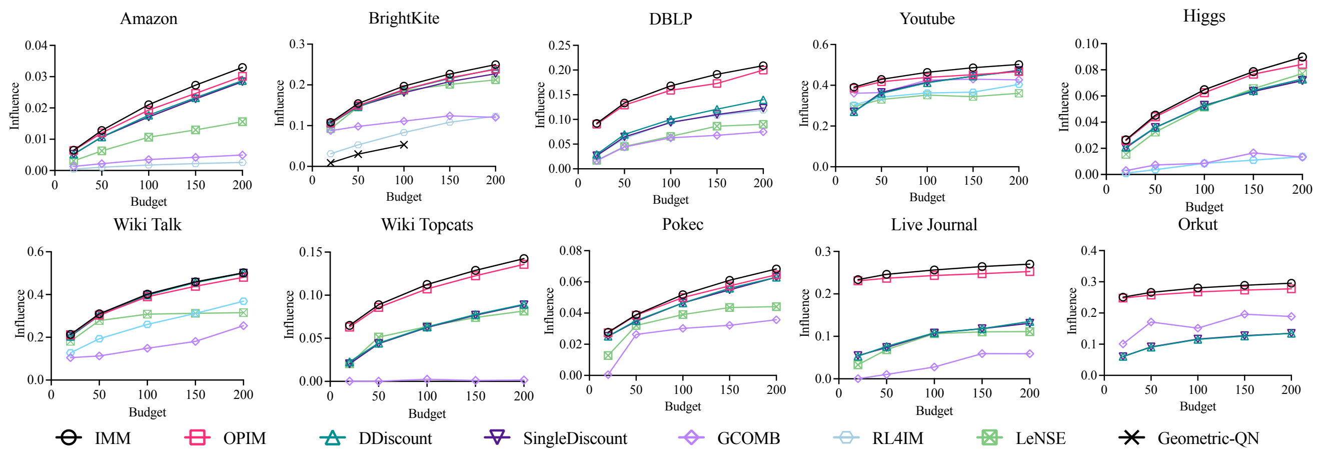


Figure 10: IM: Influence curve under WC. Influence represents the proportion of active nodes in the graph.

	Gowalla	Youtube	Higgs	Pokec	Wiki Talk
S2V-DQN	0.58	2.12	6.47	11.96	3.69
GCOMB	0.91	3.38	9.61	17.95	6.05
LeNSE	0.78	3.00	8.53	15.1	5.45
Lazy Greedy	0.18	0.71	1.61	3.23	1.28
Normal Greedy	0.01	0.03	0.01	0.04	0.06
	BK-WC	BK-TV	YT-CO	PK-WC	PK-CO
IMM	0.02	0.38	0.41	0.84	27.18
OPIM	0.02	0.23	0.18	0.43	19.00
DDiscount	0.01	0.02	0.14	0.64	0.64
LeNSE	0.34	0.34	3.30	20.3	19.99
GCOMB	2.15	1.43	3.80	13.77	13.47
RL4IM	0.05	0.05	0.69	/	/
Geometric-DQN	0.28	0.3559	/	/	/

Table 2: Memory usage (Gbyte). The upper part of the table records peak memory usage of the algorithms in the MCP experiment, whereas the lower part records the usage in the IM experiment for the datasets BrightKite (BK), Youtube (YT), and Pokec (PK) under the WC, TV, and CONST (CO) modes

5 COMMON ISSUES OF DEEP-RL METHODS

The results in Section 4 contradict the hope of all the deep-RL methods [1–5] that using deep learning and RL can efficiently learn a better approximation of the coverage function or the influence function to enable more efficient and effective approximations to MCP or IM. In this section, we conduct more experiments to reveal some common issues of the deep-RL methods [1–5] that make them not work well in practice.

5.1 Study of Graph Distribution

Deep-RL studies [1–5] argue that deep-RL methods can learn heuristics to solve combinatorial problems on graphs and generalize well to graphs following a similar distribution. However, the "graph distribution" has not been defined rigorously in these studies. In practice, to use the trained deep-RL models for MCP and IM, we should be able to efficiently decide whether a new input graph follows the "same distribution" as the training graphs. To this end,

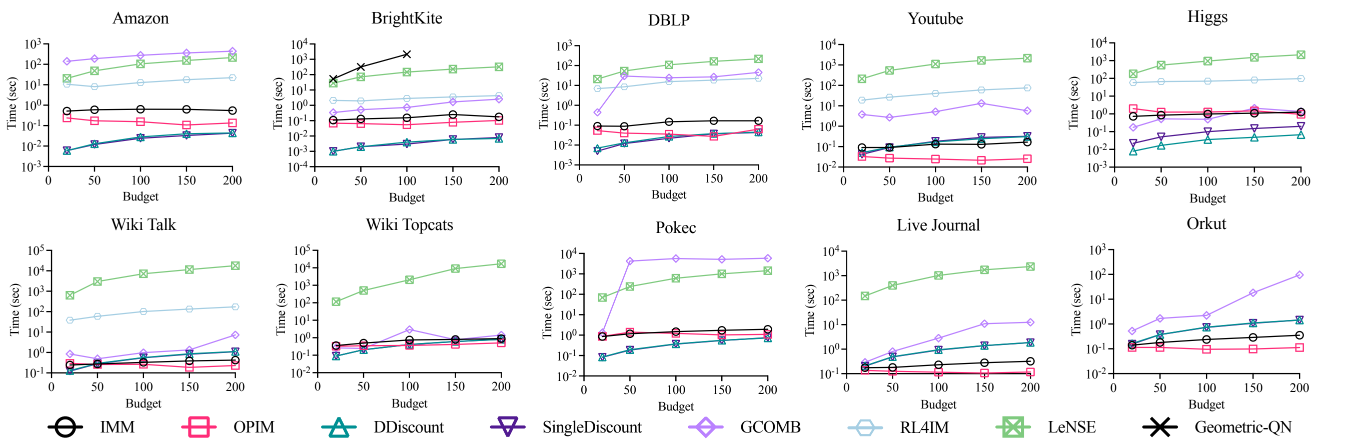


Figure 11: IM: Runtime curve under WC

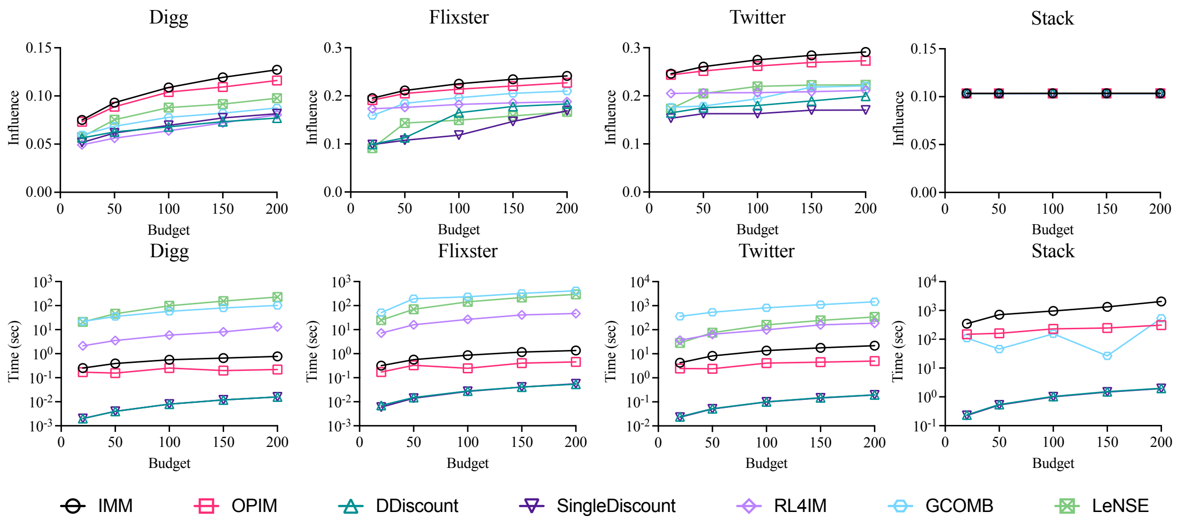


Figure 12: IM: Influence and Runtime curves under LND. Influence represents the proportion of active nodes in the graph.

we examine whether some easy-to-compute statistics of graphs can tell us if two graphs follow the "same distribution".

Edge Weights Matter in IM. We first show that the edge weights are important for IM. Specifically, we test if trained deep-RL models can generalize to the same graph under edge-weight models different from the edge-weight model in training. For simplicity, let G_M be a graph using the edge weight model M, and $\mathcal{F}_M$ be the deep-RL method $\mathcal{F}$ trained on G_M. We compare the performance of $\mathcal{F}_{CONST}$ and that of $\mathcal{F}_M$ under five graphs using edge weight model M and the budget $k = 50$. Table 3 lists the percentage change of the performance p, where $p = \frac{\mathcal{F}_M(G_M)-\mathcal{F}_{CONST}(G_M)}{\mathcal{F}_M(G_M)}$. Here, $\mathcal{F}_{CONST}(G_M)$ means $\mathcal{F}$ is trained under CONST while tested on G_M, with $M \in \{WC, TV\}$. The larger the absolute value, the more sensitive the method $\mathcal{F}$ is to the edge weight model. The results suggest that these deep-RL methods cannot generalize well across different edge-weight models.

Topology statistics do not help. We then test if some commonly adopted topology statistics of graphs can help us tell whether a testing graph follows the "same distribution" as the training graphs or not. In this part, we use the same edge-weight model for training and testing data. The topology statistics considered include the number of nodes, the number of edges, the density of the graph, the clustering coefficient, the number of triangles, the 90% effective diameter, the number of isolated nodes (those without neighbors), the maximum degree, Sum10 and Sum100, where Sum10 represents the total degree of the top-10 nodes in the graph. For IM, we also consider topology statistics related to edge weights, such as the average weighted degree and the average edge weight. We calculate the Spearman correlation coefficient between the coverage gap $[\delta_0, \delta_1, \cdots \delta_n]$ and the topology statistics of graphs $\{[x_0, x_1, \cdots x_n], [y_0, y_1, \cdots y_n], \cdots\}$. Here, $\delta = \frac{\mathcal{F}(\cdot)-OPT.}{OPT.}$, where $OPT.$ is approximated by the coverage obtained by greedy in MCP

	TV			WC		
	GCOMB	RL4IM	LeNSE	GCOMB	RL4IM	LeNSE
BrightKite	1.29%	-0.85%	-0.19%	30.18%	26.52%	-11.73%
Amazon	21.74%	27.23%	-7.69%	7.64%	65.52%	36.45%
DBLP	86.81%	43.58%	-18.18%	-5.49%	87.61%	43.77%
Wiki Talk	2.46%	-2.26%	7.59%	6.39%	83.10%	10.20%
Youtube	59.22%	13.26%	3.18%	0.61%	2.80%	3.33%

Table 3: Percentage change of the performance

and IMM in IM, and $\mathcal{F}(\cdot)$ is the coverage or influence obtained by a deep-RL method $\mathcal{F}$. Table 4 shows the results. Unfortunately, strong positive correlations (≥ 0.8) are pretty rare. What's worse, even when a statistic has a strong positive correlation with a method, if we change the edge-weight model, such a strong correlation is gone (for example, MAX and GCOMB under TV and WC).

The results suggest that in practice, it may not be easy to decide whether a testing graph follows the "same distribution" as the training graphs. Therefore, probably it is difficult to tell whether a trained deep-RL model can perform well on a testing graph before really running it on the testing graph.

5.2 Impact of Training Time

Deep-RL methods often have convergence issues in practice [26]. We test if the deep-RL methods for MCP and IM [1–5] have such issues which may cause trouble in training the model. We fix the training data but vary the length of the training time (#epochs), and then observe how is the performance of deep-RL methods affected by the training time. Worth noting that we train all models for a much longer time than the training time reported in the original papers, but these deep-RL methods still perform significantly worse than IMM. Fig. 13 show the results. GCOMB first has a weird performance drop but then tends to converge as the number of training epochs increases. LeNSE initially learns effectively but experiences occasional drops in performance, while RL4IM shows a steady increase in performance until about 3,000 epochs, after which no further improvement is observed. However, RL4IM demonstrates overfitting and worsened generalizability with prolonged training, as shown in Figure 13d. Similarly, Geometric-DQN does not efficiently learn as the training steps increase, as indicated in Figure 13c. In summary, increasing the training time for deep-RL methods may not always help, making it challenging to determine when to stop the training process. This makes tuning deep-RL methods difficult in practice.

5.3 Impact of Training Dataset Size

Obtaining datasets with high-quality labels can be costly, highlighting the need to investigate the impact of training dataset size on model performance. In this benchmark study, the training approaches for Deep RL methods can be categorized into two types based on the size of the training datasets. GCOMB and LeNSE are trained on fractional subsets of a dataset, while RL4IM and Geometric-DQN are trained on multiple datasets. Consequently, the training dataset size can refer to either the number of nodes in a graph or the number of graphs (i.e., samples). In the experiment, we train GCOMB and LeNSE using a series of subgraphs of varying proportions no more than 30% of Youtube and then test these models on the same graph constructed with 70% of the edges unseen from the training subgraphs. Following the instructions in [2], Geometric-DQN is trained on either a different number of real datasets or synthetic graphs. In the case of RL4IM and Geometric-DQN trained on synthetic graphs, we not only vary the number of samples while keeping the number of nodes fixed but also train the model with a fixed number of samples, each having a different number of nodes.

The experimental results reveal that none of the methods show improved performance with an increased training dataset size More specifically, from the effect of the number of nodes in the training subgraphs on the model's performance, figure 15 shows that GCOMB and LeNSE achieves its best performance when trained on a subgraph of 15% size, but their performance drastically drops as the training dataset size increases, indicating instability. Referring to Fig. 14a, RL4IM is much more stable, and a subtle trend appears that RL4IM tends to perform better as the size of the training subgraphs increases when tested on graphs of similar size. However, there is no significant correlation between generalizability and training dataset size, as RL4IM trained on smaller graphs (n=100) outperforms the model trained on larger graphs when tested on graphs that are 10 or 100 times the size of the training graphs. Examining the impact of the number of training subgraphs, size does not significantly affect RL4IM's performance or generalizability. Furthermore, Fig. 14b indicates that increasing the number of training graphs does not improve the performance of Geometric-DQN either. These findings demonstrate that merely enlarging the training dataset size may harm the model's performance. Thus, it is crucial to make extra efforts to determine an appropriate volume of training data. This also brings challenges to the tuning of deep-RL methods in practice.

6 FINAL THOUGHTS ABOUT DEEP-RL HEURISTICS FOR MCP AND IM

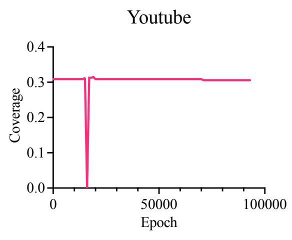
Our benchmark study in Section 4 and Section 5 demonstrate that deep-RL methods for MCP and IM still have many fatal drawbacks. We probably are not ready to trust the "learned heuristics" obtained by these deep-RL methods in practice. We give our thoughts on why deep-RL may not be a suitable solution to MCP or IM.

First, both MCP and IM, though NP-hard, have strong approximation algorithms such as Greedy and IMM. In addition, the approximation guarantees of these strong algorithms are already tight since the ratio $1 - \frac{1}{e}$ is a barrier if $\mathbf{P} \neq \mathbf{NP}$. Note that deep-RL methods essentially are heuristic algorithms. Therefore, the worst-case performance of deep-RL methods probably cannot beat the existing strong approximation algorithms like Greedy and IMM.

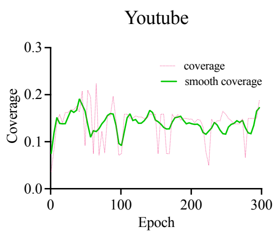
One may argue that learned heuristics exploit the distribution of real data so worst-case performance is not appropriate for evaluating deep-RL methods. However, the spirit of evaluation in IM literature is actually checking the worst-case performance of algorithms. This is due to that in practice, we often do not have enough user action logs to learn edge weights, which have a strong impact on the influence spread. So for IM, it is very hard to learn the "distribution" of real data. As a result, existing studies [12, 17, 18] proposed so many edge-weight models like WC, and comprehensively evaluate IM algorithms under almost all edge-weight models.

	MCP			CONST			TV			WC		
	LeNSE	GCOMB	S2V-DQN	LeNSE	GCOMB	RL4IM	LeNSE	GCOMB	RL4IM	LeNSE	GCOMB	RL4IM
$\|V\|$	-0.286	**0.943**	0.771	-0.710	0.143	0.257	0.721	0.714	0.371	0.7	-0.143	0.486
$\|E\|$	-0.238	0.543	0.314	-0.912	0.371	0.543	**0.901**	0.6	0.657	**0.870**	-0.429	0.257
Density	0.024	-0.6	-0.6	0.321	0.6	0.771	0.323	0.143	0.429	0.3	0.029	-0.2
Avg. clustering coefficient	0.5	-0.6	-0.6	-0.728	-0.486	-0.429	-0.712	-0.829	-0.543	-0.712	-0.314	-0.6
Fraction of closed triangles	0.381	-0.714	-0.657	-0.772	-0.543	-0.543	-0.689	-0.886	-0.657	-0.692	-0.086	-0.543
Diameter	0.122	-0.174	-0.174	-0.872	-0.841	-0.696	-0.872	-0.638	-0.986	-0.873	0.203	-0.232
90% effective diameter	0.072	-0.086	-0.086	-0.921	-0.886	-0.771	-0.901	-0.6	-1.0	-0.901	0.257	-0.143
Num. of Isolated	-0.18	0.783	0.522	-0.112	-0.464	-0.261	-0.120	0.464	-0.145	-0.1	0.319	0.754
Max	-0.524	0.486	0.371	0.652	0.429	0.486	0.6	**0.886**	0.6	0.6	0.429	0.771
Sum10	-0.476	-0.029	0.029	0.612	0.6	0.486	0.6	0.6	0.6	0.6	0.486	0.486
Sum100	-0.476	-0.029	0.029	0.612	0.6	0.486	0.6	0.6	0.6	0.6	0.486	0.486
Avg. weighted degree	-	-	-	0.486	0.429	0.371	0.2	-0.371	0.257	0.371	-0.314	-0.6
Avg. edge weight	-	-	-	0.371	0.257	0.371	0.143	0.543	0.257	0.486	**0.829**	0.657

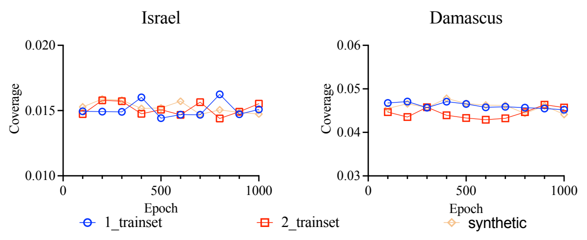
Table 4: Spearman correlation coefficient between graph statistics and coverage gap. Values ≥0.8 are highlighted in bold.



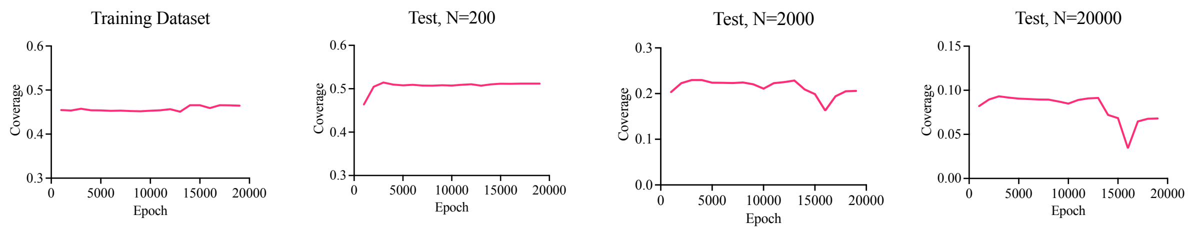
(a) GCOMB



(b) LeNSE



(c) Geometric-DQN



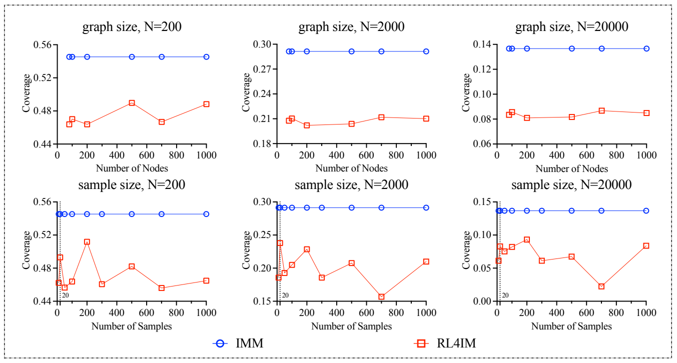
(d) RL4IM

Figure 13: Performance curves of models under different lengths of training time under WC model. The x-axis is the training epoch, and the y-axis is the influence. (a): Training dataset is a subgraph sampled out of Youtube by randomly selecting 15% of edges, and an another 15%-subgraph is used for validation. The test dataset is a subgraph formed by the remaining 70% of the edges (following [3]). (b): The legends represent the model being trained on different datasets, the Copen dataset alone, both Copen and Occupy datasets, and synthetic graphs, respectively(following [2]). (c): Training datasets consist of 200 synthetic graphs of 200 (± 20) nodes. Test datasets are three groups of synthetic graphs of different size (denoted as N).

7 OTHER RELATED WORK

Maximum Coverage and Influence Maximization Problems. The Maximum Coverage Problem (MCP) is a well-known NP-hard problem that has received significant attention. Various variants of MCP, such as the Budgeted Maximum Coverage Problem [27–30] and the Multiple Knapsack Problem [31], have been extensively studied. These problems find applications in diverse fields, including facilities location [8, 32], maximum coverage in the streaming model [33], and information retrieval [34].

The Influence Maximization (IM) problem, as introduced by Kempe et al. [12], has also garnered significant attention. By focusing on the diffusion of influence, IM addresses the concept of coverage from a different perspective, making it a natural and relevant extension of the MCP framework in the context of social networks. Kempe et al. proposed a greedy algorithm that provides a $(1-\frac{1}{e}-\epsilon)$ approximation under the Independent Cascade (IC) model. Subsequent studies have focused on improving the efficiency and scalability of influence maximization algorithms. For instance, CELF +[13] and its improved version CELF++ [35] significantly reduced the Monte Carlo evaluation times while maintaining a $\frac{1}{2}(1-\frac{1}{e})$ approximation. Other heuristic algorithms [25, 36, 37], offer more efficient solutions without relying on Monte Carlo simulations, although they may not provide strong theoretical guarantees. To address the computational challenges of influence maximization, Borgs et al. [15] introduced the Reverse Influence Sampling (RIS)


(a) RL4IM

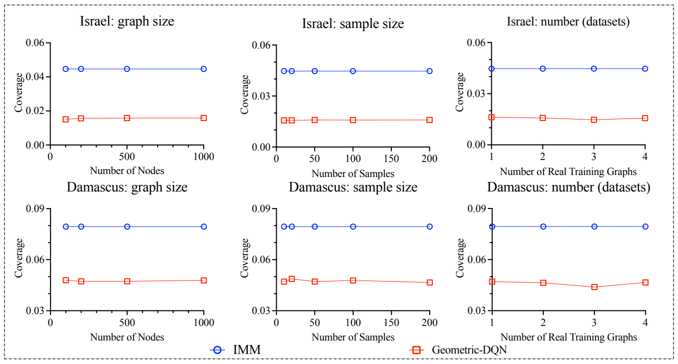

(b) Geometric-DQN

Figure 14: Impact of training data size. (a): In the figures in the first row, the x-axis is the number of nodes n (± 20) in each of the 200 training graphs (samples). In the figures in the second row, the x-axis is the number of training samples/graphs, and n of each sample is 200 (±20). There are 10 testing graphs in each group of different numbers of nodes N, and the y-axis is the average coverage obtained over these 10 test graphs. (b): In the figures in the first column, the x-axis is the number of nodes n in each of the 100 synthetic training graphs (samples). In the figures in the second column, the x-axis is the number of synthetic training samples/graphs, and n of each sample is 1000 (±30). The a-xis in the figures in the third column is the number of real training graphs.

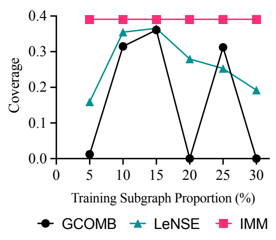


Figure 15: Impact of training data size. The x-axis is the percentage of edges in Youtube used for training. Tested with budget 20 and under WC model.

technique, which achieves nearly linear time complexity relative to the graph size while providing a $(1 - \frac{1}{e} - \epsilon)$-approximation under the IC model. Building upon RIS, Tang et al. proposed TIM/TIM++ [16] and IMM [17], which further improved the empirical efficiency. Tang et al. also introduced OPIM [20], which focuses on influence maximization in online settings.

Criticism on Machine Learning-based Heuristics for Combinatorial Optimization. The remarkable success of machine learning (ML) in diverse fields such as computer vision [38], natural language processing [39], and automatic control [40] has prompted the belief that ML can excel in other domains as well. However, recent studies have shed light on the limitations of deep learning (DL), both from a broad perspective and in specific applications. Camilleri and Chollet [41, 42] have discussed these limitations, providing insights into the challenges faced by DL. Furthermore, Cremer et al. [43] have examined expert opinions on the potential and limitations of DL, summarizing a body of work that uncovers the inadequate performance of DL in various applications.

In a recent contribution, Angelini et al. [44] proposed a benchmark study that highlights the disparity between DL solvers and a simple greedy algorithm in solving the maximum independent set (MIS) problem. Their findings demonstrate that the greedy algorithm not only provides better quality solutions but also exhibits significantly faster computation, surpassing Deep RL solvers by a factor of 10^4. This work emphasizes the need to carefully evaluate the performance of DL techniques in specific problem domains, as there may be alternative approaches that outperform DL in terms of both solution quality and computational efficiency.

These discussions and benchmark studies serve as important reminders that while DL has achieved remarkable success in numerous areas, it is not a one-size-fits-all solution. Understanding the limitations and exploring alternative approaches can help researchers and practitioners make informed decisions regarding the most suitable techniques for solving specific problems.

8 CONCLUSIONS

In this benchmark study, we comprehensively examine the effectiveness and efficiency of five recent deep-RL methods for MCP and IM. Our experimental results demonstrate that for both MCP and IM, traditional algorithms like Lazy Greedy and IMM can still outperform deep-RL methods in most cases or in even all cases. We also reveal some common issues of deep-RL methods for MCP and IM, such as it is difficult to tell if a trained deep-RL model can work well on a testing graph before really running it. Finally, we discuss some potential reasons why deep-RL may not be a good solution to MCP and IM. Our benchmark study sheds light on potential problems in current deep reinforcement learning research for solving combinatorial optimization problems.

REFERENCES

[1] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[2] Harshavardhan Kamarthi, Priyesh Vijayan, Bryan Wilder, Balaraman Ravindran, and Milind Tambe. Influence maximization in unknown social networks: Learning policies for effective graph sampling. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 575–583, 2020.

[3] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems*, 33:20000–20011, 2020.

[4] Haipeng Chen, Wei Qiu, Han-Ching Ou, Bo An, and Milind Tambe. Contingency-aware influence maximization: A reinforcement learning approach. In *Uncertainty in Artificial Intelligence*, pages 1535–1545. PMLR, 2021.

[5] David Ireland and Giovanni Montana. Lense: Learning to navigate subgraph embeddings for large-scale combinatorial optimisation. In *International Conference on Machine Learning*, pages 9622–9638. PMLR, 2022.

[6] Valérie Guihaire and Jin-Kao Hao. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice*, 42(10):1251–1273, 2008.

[7] Güneş Erdoğan, Erhan Erkut, Armann Ingolfsson, and Gilbert Laporte. Scheduling ambulance crews for maximum coverage. *Journal of the Operational Research Society*, 61:543–550, 2010.

[8] Darshan Chauhan, Avinash Unnikrishnan, and Miguel Figliozzi. Maximum coverage capacitated facility location problem with range constrained drones. *Transportation Research Part C: Emerging Technologies*, 99:1–18, 2019.

[9] Chawis Boonmee, Mikiharu Arimura, and Takumi Asada. Facility location optimization model for emergency humanitarian logistics. *International Journal of Disaster Risk Reduction*, 24:485–498, 2017.

[10] Maryam Habibi and Andrei Popescu-Belis. Keyword extraction and clustering for document recommendation in conversations. *IEEE/ACM Transactions on audio, speech, and language processing*, 23(4):746–759, 2015.

[11] Xuefeng Chen, Yifeng Zeng, Gao Cong, Shengchao Qin, Yanping Xiang, and Yuanshun Dai. On information coverage for location category based point-of-interest recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[12] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

[13] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429, 2007.

[14] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[15] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing social influence in nearly optimal time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 946–957. SIAM, 2014.

[16] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 75–86, 2014.

[17] Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1539–1554, 2015.

[18] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1029–1038, 2010.

[19] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. A data-based approach to social influence maximization. *arXiv preprint arXiv:1109.6886*, 2011.

[20] Jing Tang, Xueyan Tang, Xiaokui Xiao, and Junsong Yuan. Online processing algorithms for influence maximization. In *Proceedings of the 2018 International Conference on Management of Data*, pages 991–1005, 2018.

[21] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *International conference on machine learning*, pages 2702–2711. PMLR, 2016.

[22] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[23] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[24] J-P Onnela, Jari Saramäki, Jorkki Hyvönen, György Szabó, David Lazer, Kimmo Kaski, János Kertész, and A-L Barabási. Structure and tie strengths in mobile communication networks. *Proceedings of the national academy of sciences*, 104(18):7332–7336, 2007.

[25] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 199–208, 2009.

[26] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[27] Binayak Kar, Eric Hsiao-Kuang Wu, and Ying-Dar Lin. The budgeted maximum coverage problem in partially deployed software defined networks. *IEEE Transactions on Network and Service Management*, 13(3):394–406, 2016.

[28] Breno Piva. Approximations for restrictions of the budgeted and generalized maximum coverage problems. *Electronic Notes in Theoretical Computer Science*, 346:667–676, 2019.

[29] Liwen Li, Zequn Wei, Jin-Kao Hao, and Kun He. Probability learning based tabu search for the budgeted maximum coverage problem. *Expert Systems with Applications*, 183:115310, 2021.

[30] Jianrong Zhou, Jiongzhi Zheng, and Kun He. Effective variable depth local search for the budgeted maximum coverage problem. *International Journal of Computational Intelligence Systems*, 15(1):43, 2022.

[31] Hans Kellerer, Ulrich Pferschy, and David Pisinger. Multidimensional knapsack problems. In *Knapsack problems*, pages 235–283. Springer, 2004.

[32] Nimrod Megiddo, Eitan Zemel, and S Louis Hakimi. The maximum coverage location problem. *SIAM Journal on Algebraic Discrete Methods*, 4(2):253–261, 1983.

[33] Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *Proceedings of the 2009 siam international conference on data mining*, pages 697–708. SIAM, 2009.

[34] Aris Anagnostopoulos, Luca Becchetti, Ilaria Bordino, Stefano Leonardi, Ida Mele, and Piotr Sankowski. Stochastic query covering for fast approximate document retrieval. *ACM Transactions on Information Systems (TOIS)*, 33(3):1–35, 2015.

[35] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Celf++ optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*, pages 47–48, 2011.

[36] Chi Wang, Wei Chen, and Yajun Wang. Scalable influence maximization for independent cascade model in large-scale social networks. *Data Mining and Knowledge Discovery*, 25:545–576, 2012.

[37] Qingye Jiang, Guojie Song, Cong Gao, Yu Wang, Wenjun Si, and Kunqing Xie. Simulated annealing based influence maximization in social networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 25, pages 127–132, 2011.

[38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[41] Daniel Camilleri and Tony Prescott. Analysing the limitations of deep learning for developmental robotics. In *conference on Biomimetic and Biohybrid Systems*, pages 86–94. Springer, 2017.

[42] Francois Chollet. The limitations of deep learning. *Deep learning with Python*, 2017.

[43] Carla Zoe Cremer. Deep limitations? examining expert disagreement over deep learning. *Progress in Artificial Intelligence*, 10:449–464, 2021.

[44] Maria Chiara Angelini and Federico Ricci-Tersenghi. Cracking nuts with a sledgehammer: when modern graph neural networks do worse than classical greedy algorithms. *arXiv preprint arXiv:2206.13211*, 2022.

A REPRODUCED RESULTS OF RL4IM & GEOMETRIC-DQN

RL4IM and Geometric-DQN perform badly in real large-scale datasets, and then we carry out more experiments on validating their effectiveness on small datasets mentioned in the accordingly paper.

We train RL4IM following instructions in [4], and all methods repeat the query over ten times and then take an average as the final result. RL4IM performs better than CHANGE when the budget is small on synthetic graphs of 200 nodes, the same as reported in [4]. Moreover, we extend the experiment to the larger size of graphs of 2000 and 20,000 nodes under CONST and WC models, and RL4IM is always superior to CHANGE. This indicates that we play the best of RL4IM, but it is still inferior to IMM in all these small synthetic graphs.

[2] shows that Geometric-DQN obtains 37.8% and 61.1% of the influence scores of the greedy algorithm for the datasets Israel and Damascus, respectively. Since IMM could provide the same approximation ratio as the greedy algorithm, we directly compare Geometric-DQN with IMM in our experiments. Due to the high variance of Geometric-DQN, we repeat the query over 20 times and then take an average as the final result. Our experiment shows that Geometric-DQN has 27.5% and 66.1% of the coverage of IMM in Israel and Damascus, respectively. Though unlike what [2] reported, Geometric-DQN lags behind IMM undoubtedly. Meanwhile, the gap widens in the WC model. The runtime curve, which is not reported in [2], indicates that Geometric-DQN takes considerable time to infer, even on small graphs, making it unscalable to large-scale graphs.

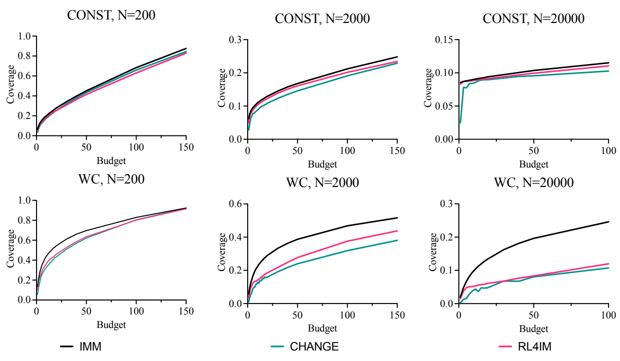


Figure 16: test RL4IM over synthetic graphs of different N under the given weight model.The coverage is an average of the results of 10 repeated experiments.

B NOISE PREDICTOR IN GCOMB

GCOMB runs up to two orders of magnitude faster than the simple greedy algorithm and S2V-DQN in MCP is partly credited with the noise predictor. Concretely, GCOMB reduces the search space with aid of filtering out a large number of noisy nodes by noise predictor. Therefore, GCOMB's performance relies heavily on the performance of the noise predictor. Unfortunately, our result demonstrates that the noise predictor, a linear interpolation method in essence, can't learn how to distinguish noisy nodes well.

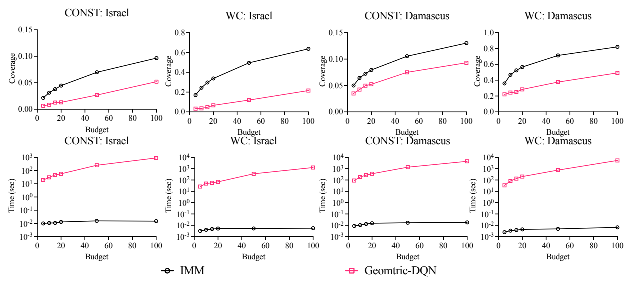


Figure 17: test Geometric-DQN over small-scale datasets under the given weight model. The coverage is an average of the results of 20 repeated experiments.

Table 5: Training Time of Noise Predictor (sec)

Budget	DBLP	Youtube	Live Journal
20	13.2	33.2	384.4
50	18.8	45.2	515.8
100	27.9	66.5	728.9
150	36.8	88.5	951.0
200	46.1	109.0	1181.5

Table 6: Number of Non-Noisy Nodes

Budget	DBLP	Youtube	Live Journal
20	0.030%	0.002%	0.01%
50	0.025%	0.010%	0.04%
100	0.415%	0.013%	0.013%
150	0.304%	0.020%	0.022%
200	0.327%	0.049%	0.061%

B.1 Training time of Noise Predictor

The way to train a Noise predictor proposed in [3]is not always feasible. In our experiments, the noise predictor can not be generalized to unseen budgets well so we follow the way how the author did in the experiments, train a noise predictor specifically for every budget. Table 5 shows how much time is needed to train a noise predictor in MCP for each budget. This process even takes thousands of times longer to run than Lazy Adaptive Greedy solves a problem.

B.2 Proportion of Good Nodes

Table 6 shows the proportion of good nodes in the entire graph in MCP. It shows that the proportion is not monotone increasing and even varies dramatically, which accounts for the cases that the runtime of GCOMB fluctuates wildly.

What's worse, the noise predictor is unfeasible in some cases. We can find that GCOMB takes much more time than expected to find the solution on Amazon and Stack in the IM problem. Simply because the number of good nodes derived from the noise predictor is larger than the total number of nodes in the entire graph, in this case, GCOMB has to find the solution set from the entire graph, which takes a significant amount of time.

C IMPROVE LENSE EFFICIENCY

The original implementation of LeNSE is slow performance, requiring several days to generate a training dataset even for small graphs. In our optimization efforts, we have significantly improved the implementation, reducing the preprocessing runtime from days (> 72 hours) to just a few minutes (11.3 minutes), while preserving the underlying logic. In IM problem, our optimizations include replacing the Python-based IMM with a faster C++ version, utilizing reverse influence sampling techniques for calculating influence spread, and dynamically generating training subgraphs of varying quality. As for MCP, we replace the heuristic method normal greedy with our proposed Lazy Adaptive Greedy to boost the efficiency in preprocessing and inference phase.