A Benchmark Study of Deep-RL Methods for Maximum Coverage Problems over Graphs [Experiment, Analysis & Benchmark]

Zhicheng Liang
zhicliang3-c@my.cityu.edu.hk
City University of Hong Kong
Hong Kong, China

Yu Yang
yuyang@cityu.edu.hk
City University of Hong Kong
Hong Kong, China

Xiangyu Ke
xiangyu.ke@zju.edu.cn
Zhejiang University
Hangzhou, China

Xiaokui Xiao
xkxiao@nus.edu.sg
National University of Singapore
Singapore, Singapore

Yunjun Gao
gaoyj@zju.edu.cn
Zhejiang University
Hangzhou, China

ABSTRACT

Recent years have witnessed a growing trend toward employing deep reinforcement learning (Deep-RL) to derive heuristics for combinatorial optimization (CO) problems on graphs. Maximum Coverage Problem (MCP) and its probabilistic variant on social networks, Influence Maximization (IM), have been particularly prominent in this line of research. In this paper, we present a comprehensive benchmark study that thoroughly investigates the effectiveness and efficiency of five recent Deep-RL methods for MCP and IM. These methods were published in top data science venues, namely S2V-DQN [1], Geometric-QN [2], GCOMB [3], RL4IM [4], and LeNSE [5]. Our findings reveal that, across various scenarios, the Lazy Greedy algorithm *consistently outperforms all* Deep-RL methods for MCP. In the case of IM, theoretically sound algorithms like IMM and OPIM demonstrate superior performance compared to Deep-RL methods in most scenarios. Notably, we observe an *abnormal phenomenon* in IM problem where Deep-RL methods slightly outperform IMM and OPIM when the influence spread nearly does not increase as the budget increases. Furthermore, our experimental results highlight common issues when applying Deep-RL methods to MCP and IM in practical settings. Finally, we provide insights into why Deep-RL may not be a suitable solution for MCP and IM based on our observations. Our benchmark study sheds light on potential challenges in current deep reinforcement learning research for solving combinatorial optimization problems.

PVLDB Reference Format:
Zhicheng Liang, Yu Yang, Xiangyu Ke, Xiaokui Xiao, and Yunjun Gao. A Benchmark Study of Deep-RL Methods for Maximum Coverage Problems over Graphs [Experiment, Analysis & Benchmark]. PVLDB, 14(1): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://anonymous.4open.science/r/MCPBenchmark-B2BF.

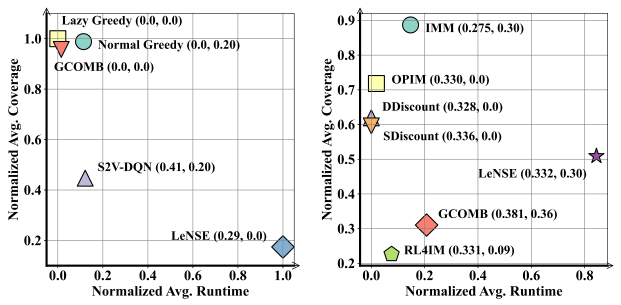
Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Combinatorial optimization problems play a pivotal role in addressing complex challenges across diverse domains. The *Maximum Coverage Problem* (MCP) and its probabilistic variant, *Influence Maximization* (IM), stand out as representative challenges in this realm. The primary goal of MCP is to *identify a set S of k nodes from a given input graph G to maximize node coverage.* On the other hand, IM focuses on maximizing influence spread in social networks by selecting a strategic set of nodes. The significance of MCP and IM transcends theoretical domains, finding practical applications in scheduling [6, 7], facility location [8, 9], recommendation systems [10, 11], viral marketing [12], and sensor placement [13]. Therefore, devising effective and efficient algorithmic techniques for solving MCP and IM has drawn great attention from the data management and data mining research communities.

Existing studies have made creditable strides in understanding the algorithmic complexities of MCP and IM. 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 greedy algorithm's applicability extends to IM, as illustrated by [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 enhanced the practical efficiency of the RIS-based algorithms [16, 17]. Tang et al. utilized the online approximation bound of submodular functions [18] to return online approximation bounds of RIS-based algorithms. As with MCP, any approximation ratio better than $1-\frac{1}{e}$ is implausible unless $\mathbf{P}\neq\mathbf{NP}$.

Despite the substantial theoretical advancements in MCP and IM, recently, there has been a notable surge in the application of *Deep Reinforcement Learning* (Deep-RL) to these combinatorial optimization problems [1–5]. This contemporary approach seeks to harness the power of data-driven learning, aiming to derive heuristics that can surpass the performance of theoretically grounded algorithms in practice. In these studies, *Graph Neural Networks* (GNNs) initially learn node embeddings, followed by the integration of reinforcement learning components like Q-learning to approximate the objective function. Empirical studies of these works reveal instances where they outperform traditionally established algorithms for both MCP and IM.



(a) MCP **(b) IM**

Figure 1: Performance overview of methods on MCP and IM: Values represented as **Method** (Coverage standard deviation, Runtime standard deviation). Geometry-QN was excluded from (b) due to the scalability issues on most graphs.

An aspect of concern in the application of Deep-RL lies in the treatment of training time. Notably, the prevailing practice of excluding training time from computation cost evaluations raises valid questions. Viewing *training as akin to pre-processing*, it is customary in rigorous data management research to consider the pre-processing time in an amortized manner *within query computation cost assessments* [19]. The exclusion of training time might be justifiable only in scenarios where training is performed on a dataset of size independent of subsequent queries, rendering the training time a theoretically constant factor [20, 21]. However, even in such cases, the impact of constant-time training or pre-processing might be limited in boosting the efficiency of a broader range of queries [22]. Consequently, there arises a legitimate concern about the *actual effectiveness and efficiency* of these Deep-RL methods, prompting a closer examination of their claimed benefits.

To this end, this paper conducts a thorough benchmark study on recent Deep-RL methods [1–5]. Fig. 2 presents a comprehensive overview of the benchmarking framework. The pre-processing phase initiates the generation of training data as needed, and propagation probabilities are assigned as edge weights in IM based on diverse edge weight models [3, 12, 23]. Deep-RL solvers, categorized into global and subgraph exploration types, undergo training and validation before their applications. In tandem, traditional solvers encompass both approximate and heuristic approaches. Subsequently, a solution scorer is deployed to calculate coverage. In the case of IM, the scorer estimates the influence spread $F_{\mathcal{R}}(S)$ based on RIS-based simulations. Meanwhile, for MCP, the coverage $F(S)$ is directly calculated on the input graph. We evaluate on 20 commonly-used datasets and test 4 edge weight models. Finally, a thorough analysis of the results is conducted from diverse perspectives, enabling a nuanced understanding of the strengths and limitations of Deep-RL methods compared to traditional solvers.

The contributions of this paper are summarized as follows:

(1) We summarize the general pipeline of recent Deep-RL methods [1–5] for MCP and IM in § 3. Additionally, in § 3.4 and 3.5, we articulate concerns regarding the exclusion of training time and the absence of a strong baseline, Lazy Greedy, for MCP in the Deep-RL studies, which motivate our study.
(2) § 4 details an extensive benchmark study aimed at examining recent Deep-RL methods for MCP and IM. We successfully reproduce some of the experimental results reported in prior studies [1–5] and expand more. Our findings, illustrated in Fig. 1, reveal that, despite reported successes, traditional and well-established MCP and IM algorithms often outperform these Deep-RL methods in terms of both effectiveness and efficiency.
(3) § 5 delves deeper into the practical challenges faced by Deep-RL methods. We highlight common issues, such as the difficulty in determining the suitability of a testing graph for a trained Deep-RL model and the observed performance fluctuations concerning training time and data size.
(4) Our experimental findings prompt a discussion on the challenges that must be addressed for Deep-RL methods to provide effective and efficient solutions for MCP and IM, offering insights for future research in this domain (§ 8).

For conciseness, certain evaluations, including those for GCOMB's noise predictor, strategies to enhance LeSNE's efficiency, and outcomes from more datasets are omitted from the main text. Readers are encouraged to refer to the appendices in our full version[1].

2 PRELIMINARIES

In this section, we formally revisits the *Maximum Coverage Problem* (MCP) and its probabilistic variant in social networks, *Influence Maximization* (IM).

2.1 Problem Statements

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 problem can be viewed as a maximum coverage or reachability problem on a probabilistic graph [24]. 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. 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., $0 \leq p_{uv} = w_{uv} \leq 1$. 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.*

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

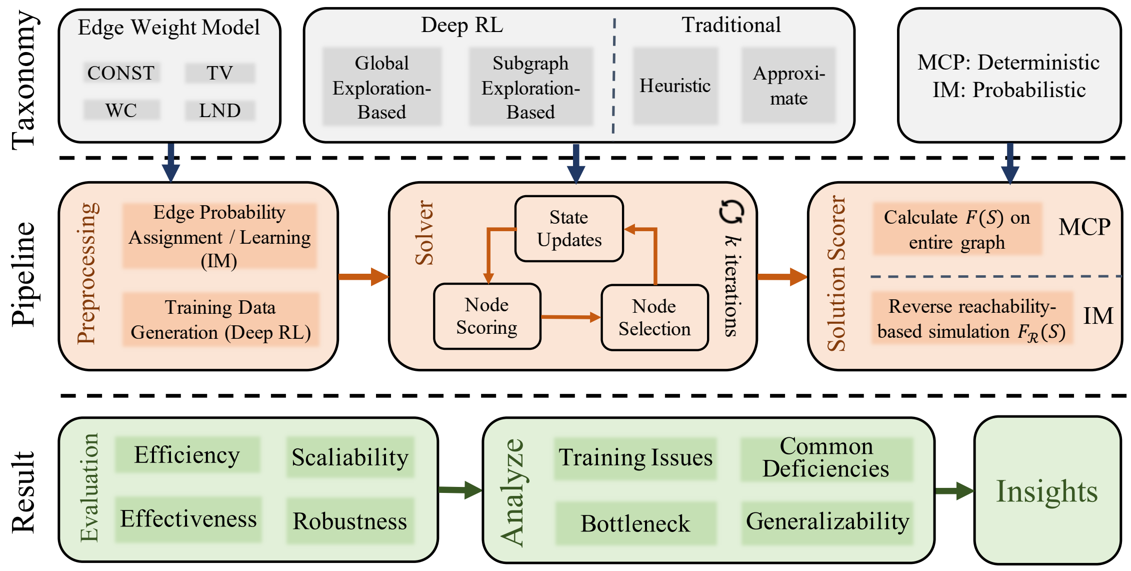


Figure 2: Benchmarking Framework

2.2 Characteristics of MCP and IM

Intractability. Both MCP and IM problems are known to be **NP**-hard [12]. An essential characteristic shared by both is the presence of *monotonicity* and *submodularity* in their objective functions [12]. These properties enable the development of efficient approximation algorithms efficient approximation algorithms, guaranteeing a performance ratio of $1 - \frac{1}{e}$ [12, 17]. The established understanding is that achieving an approximation ratio higher than $1 - \frac{1}{e}$ is implausible unless the complexity classes **P** and **NP** are equal [12].

Influence Spread Estimation. In addition to the **NP**-hardness, computing the influence spread under the IC model for IM problem is known to be **#P**-hard [25]. To address this challenge, the polling method [15] emerges as an efficient solution, ensuring a $1 - \frac{1}{e} - \epsilon$ approximation guarantee for IM with high probability. In the Polling method, the estimation of influence spread involves the sampling of Reverse Reachable (RR) sets [16], outlined 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 practical applications, a large sample size M ensures an accurate estimation of the influence spread $I(S)$ with a high probability.

2.3 Edge Weight Models in IM

The Influence Maximization (IM) problem involves a weighted graph where edge weights p_{uv} denote the direct influence from node u to node v. However, acquiring accurate data for learning these edge weights is often challenging. In the IM literature, predefined models are commonly employed to address this issue:

Tri-valency (TV) Model [3]. The weight of an edge is chosen randomly from a set of weights {0.001, 0.01, 0.1}.

Constant (CONST) Model [3]. In this model, The weight of an edge is set as a constant value, e.g., 0.1.

Weighted Cascade (WC) Model [12]. 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 [3]. 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 [26].

Remark. It's noteworthy that for theoretically sound algorithms [16–18], *the specific choice of edge weight setting doesn't impact the effectiveness of the algorithms.* However, our benchmark study reveals that the choice of edge weight setting can influence the performance of Deep-RL heuristics §4.3.

3 ALGORITHMS REVISITED

In this section, we delve into the foundational concepts of Deep-RL and reflect upon the Deep-RL techniques and other traditional algorithms used for the MCP/IM benchmarks in our experiments.

3.1 Deep-RL with GNNs

GNNs. Graph Neural Networks (GNNs) [27] are designed to process data in graph structures, capturing relationships and features of nodes and edges. Unlike traditional neural networks, GNNs handle irregular structures, making them ideal for graph-structured data like social networks. By propagating information through the graph, GNNs capture both local and global structures.

Deep-RL. Reinforcement Learning (RL) [28] is a prominent branch of machine learning where agents learn optimal strategies through interactions with an environment, receiving feedback in the form of rewards or penalties. The fundamental goal of RL is to deduce a policy that maximizes the expected cumulative reward over time, emphasizing decision-making to achieve specific objectives. In this dynamic process, an agent, situated in a given environmental state, takes actions and receives rewards based on the outcomes, commonly modeled as a Markov Decision Process (MDP). However,

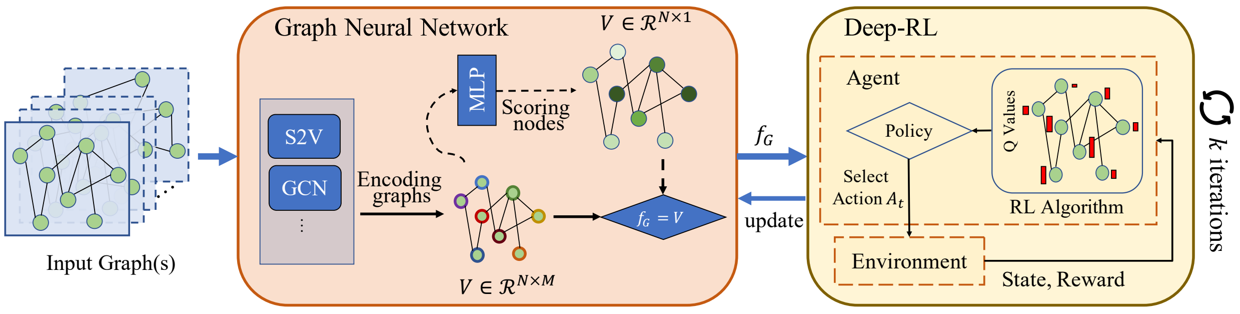


Figure 3: General pipeline of Deep-RL methods for solving MCP/IM over Graphs

dealing with intricate or high-dimensional environments poses challenges. To address this, Deep-RL integrates deep neural networks, enabling the extraction of features from raw data and the approximation of complex functions. This fusion enhances RL's capability to tackle large-scale, high-dimensional problems.

3.2 Deep-RL Methods for MCP/IM

We outline a systematic approach for employing Deep-RL methods to address coverage problems on graphs, illustrated in Fig. 3. The procedure initiates with a Graph Neural Network (GNN) encoder, tasked with learning the graph's embedding. Subsequently, an MLP layer utilizes this embedding to compute scores for each node, represented as either the node embedding matrix or the vector of node scores, denoted as f_G. Using the generated node scores, a series of seed sets are randomly created, serving as training data for the Reinforcement Learning (RL) model. In the RL environment, the feature f_G defines the state, facilitating reward computation. The RL agent, guided by a learned policy, iteratively seeks the optimal solution set S that maximizes the overall reward, with a cardinality of $|S| = k$. Subsequent paragraphs elaborate on how the methods discussed in this paper tailor this general pipeline to address specific aspects of the coverage problem on graphs.

S2V-DQN [1]. It first maps the input graph G into node embeddings via Struc2Vec [29]. A deep Q-network (DQN) [30, 31] 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-QN [2]. Starting from a randomly initialized seed set S, Geometric-QN iteratively enlarges a subgraph g by a random walk over the input graph G. Then DeepWalk [32] is used to generate node features and a GCN [33] encoder excavates the structural information. Lastly, a 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 [34] 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 DQN then finds a solution set that might maximize the cover (influence spread). Node pruning techniques are also adopted to remove noisy nodes to reduce the search space, which makes GCOMB scalable to large-scale 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.

3.3 Traditional Algorithms for MCP/IM

Greedy for MCP. Greedy algorithm [12] sequentially selects a node that covers the most remaining uncovered elements. Leveraging the submodularity of coverage functions, Lazy Greedy (also known as CELF [13]) distinguishes itself by strategically minimizing computational overhead. After initial marginal gain computations for all nodes, it selectively updates and reevaluates only the top contenders in subsequent iterations. This approach retains the $(1 - \frac{1}{e})$ approximation guarantee while delivering a significant speedup over the traditional greedy algorithm.

Degree Discount [35]. The Degree Discount (DDiscount) algorithm selects seed nodes based on their degree. Initially, the node with the highest degree is chosen. After selecting a seed, the degrees of its neighbors are adjusted to account for the influence of the already chosen seed. In each subsequent iteration, the node with the highest adjusted degree is selected as the next seed. A variation of DDiscount, the Single Discount (SDiscount) algorithm, ensures that the connectivity of direct neighbors decreases by one for each chosen seed. This adjustment ensures that a node's influence isn't counted twice, providing a nuanced approach to seed selection in influence maximization.

IMM [17]. IMM distinguishes itself by providing a guaranteed approximation ratio while maintaining efficiency, particularly suited for large-scale networks. The algorithm follows a two-phase approach. Firstly, it utilizes the Reverse Influence Sampling (RIS) method to generate samples, assessing the reachability of nodes. Subsequently, in a greedy fashion, it judiciously selects seed nodes using these samples to maximize influence.

OPIM [18]. Contrary to the offline processing algorithm IMM, OPIM stands out as an interactive online algorithm tailored for influence maximization. Users can seamlessly request a solution along with its approximation guarantee at any stage, and have the

	Dama-scus	Israel	Cond-Mat	Digg	Flixster*	Bright-Kite	Gowalla	Twitter*	DBLP	Amazon	Higgs	You-tube	Pokec	Skitter	Wiki-Topcats	Wiki-Talk	Stack*	Orkut	Live-Journal	Friend-ster
$\lvert V\rvert$	3K	3K	23K	26K	95K	58K	196K	323K	317K	334K	456K	1.1M	1.6M	1.7M	1.8M	2.4M	2.6M	3.1M	4.8M	65.6M
$\lvert E\rvert$	7.7K	8.3K	186K	200K	484K	214K	846K	2.1M	1.0M	925K	14.9M	4.2M	30.6M	11.1M	28.5M	5.0M	36.2M	117M	69M	1.8B
Density	2.54	2.25	4.04	7.5	5.05	3.68	4.83	6.65	3.31	2.76	32.53	2.63	18.75	6.54	15.92	2.1	16.26	38.14	17.26	27.53
Clust. coe.	0.01	<0.01	0.63	0.12	0.11	0.17	0.24	0.21	0.63	0.4	0.19	0.08	0.11	0.26	0.27	0.05	-	0.17	0.28	0.16
Triang. (%)	<0.01	<0.01	10.7	<0.01	<0.01	3.98	0.8	<0.01	12.83	7.92	0.29	0.21	1.61	0.18	0.17	0.11	-	1.41	4.56	0.59
Diameter	13	20	14	13	14	16	14	8	21	44	9	20	11	25	9	9	-	9	17	32
Eff. diameter	7.0	10.0	6.5	5.0	5.7	6.0	5.7	3.4	8.0	15.0	3.7	6.5	5.2	6.0	3.8	4.0	-	4.8	6.5	5.8
Isolated (%)	<0.01	<0.01	<0.01	36.84	38.8	<0.01	<0.01	24.31	40.36	20.58	<0.01	66.98	12.26	43.01	0.0	93.84	26.69	11.36	41.84	<0.01
VCI (%)	21.23	5.25	1.21	2.59	0.43	1.95	7.49	1.09	0.1	0.05	0.28	2.52	0.54	2.09	0.22	4.18	1.25	1.07	0.37	0.01
Sum_{10} (%)	48.75	25.23	8.04	19.25	3.27	13.43	25.84	6.77	0.74	0.43	2.24	8.79	2.84	18.15	1.59	11.5	9.28	8.48	1.79	0.07
Category	Tweets	Tweets	Colla-boration	Social	Social	Social	Social	Tweets	Colla-boration	Ecom-merce	Tweets	Social	Social	Trace-routes	Hyper-links	Commun ication	Q&A	Social	Social	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.

flexibility to resume the algorithm for improved outcomes. OPIM strategically chooses nodes to maximize the anticipated influence spread, all while adhering to a set budget. Leveraging a submodular function, it ensures streamlined optimization and incorporates upper bounds to boost its efficiency.

3.4 Concerns on Training in Deep-RL Methods

In *all* the aforementioned Deep-RL methods [1–5], it's crucial to note that the training time for Deep-RL models is typically not considered in the computation cost. Treating the training time as a form of **pre-processing** might present a concern, as it could potentially affect the fairness of computational performance assessments.

In rigorous database research, it is common to amortize pre-processing time into the computation time of subsequent queries. While one could argue that not counting the training time is justifiable if the training dataset size is fixed and independent of subsequent MCP or IM queries, it's important to recognize that, theoretically, such pre-processing/training step takes **constant** time. Despite this constant pre-processing time, the trained model is expected to provide performance enhancements across various queries. However, it's worth noting that, theoretically, such Deep-RL-based methods may face challenges overcoming the $1 - 1/e$ approximation barrier. The rationale lies in the fact that constant preprocessing time, when added to polynomial query time, still results in polynomial time. According to complexity theory, no polynomial time algorithm can achieve an approximation ratio better than $1 - 1/e$ unless **P** = **NP** [12]. This insight raises considerations about the inherent limitations of these methods in surpassing certain approximation thresholds.

One may argue that the approximation ratio is w.r.t. the worst-case performance on all possible query graphs, while 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 [36], 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, 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].

3.5 Concerns on Lacking Strong Baselines

Lazy Greedy, an enhanced version of the conventional greedy algorithm, stands out for its remarkable efficiency and efficacy in MCP. However, this straightforward algorithm was neglected in studies of Deep-RL methods for MCP [1, 5].

4 BENCHMARKING

All the experiments were run on a server with 16 Intel i7-11700KF 3.60GHz cores, 64G RAM, and 1 NVIDIA GeForce RTX 3090 24G GPU. Our source code and datasets can be found at [37].

Datasets. We utilized a set of 20 well-established real-world benchmark datasets [3, 38], which are commonly employed in existing studies, and their topology statistics are comprehensively outlined in Tab. 1. The considered statistics encompass (1) the number of nodes $|V|$, (2) the number of edges $|E|$, (3) the graph density, (4) the average clustering coefficient [39], (5) the fraction of closed triangles [40], (6) the 90-percentile effective diameter [41], (7) the proportion of isolated nodes (those without neighbors), (8) the vertex centralization index, denoting the ratio of the maximum degree to the number of nodes, and (9) the Sum_{10}, representing the total degree of the top-10 nodes in the graph. We examine the strength and direction of association between these statistics and the performances of DeepRL methods in § 5.1.

Our evaluations for MCP were conducted on 17 of these datasets. For the IM experiments, we conducted extensive evaluations across 10 datasets, employing edge weight models such as TV, CONST, and WC. In the case of the Learned (LND) model, influence probabilities were generated using the credit distribution model [26] on Flixster and Twitter datasets. Additionally, the Stack dataset, sourced from [3], was included in our experiments. To address scalability and performance challenges, RL4IM was tested on small synthetic graphs using the power-law model [42]. Furthermore, Geometric-QN underwent evaluation on the small datasets mentioned in [2].

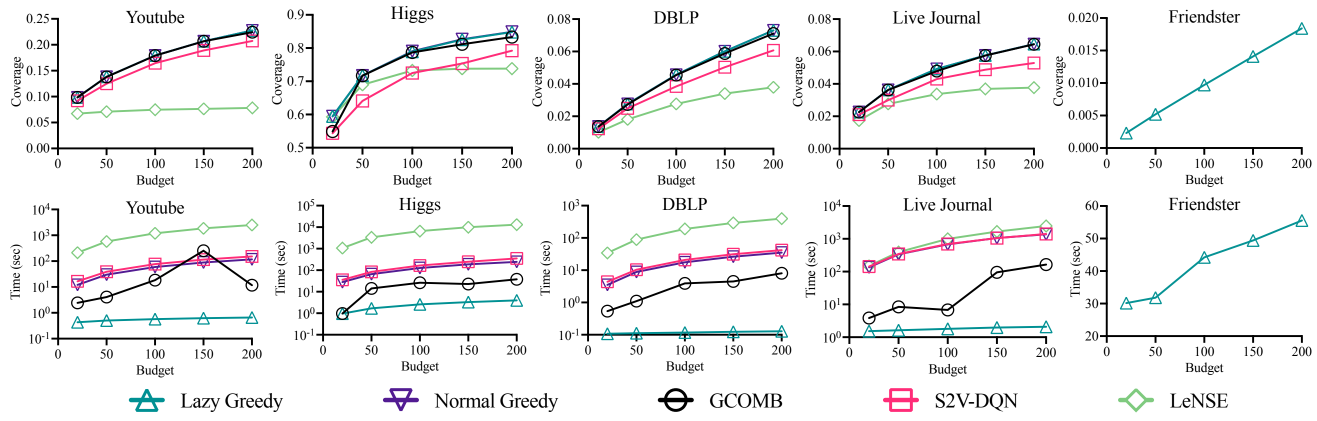


Figure 4: MCP: Coverage and Runtime curve

Implementation and hyperparameters setting. We implemented the Degree Discount heuristic [35] and the Lazy Greedy algorithm [13]. For other methods, we utilized the code provided by the respective authors. All parameters were set consistently with the recommendations in prior studies [1–5, 17]. Specifically, we set ϵ to 0.5 for IMM and 0.1 for OPIM[2], respectively.

We adhered to the methodologies outlined in [1–5] to conduct the training of Deep Reinforcement Learning (Deep-RL) models for both MCP and IM. In the case of MCP, S2V-DQN, GCOMB, and LeNSE were trained using the BrightKite dataset and subsequently tested on other datasets. For IM, GCOMB and LeNSE were trained on a subgraph randomly sampled from Youtube, with 15% of edges selected at random. The model was then tested on various datasets under all edge weight models, excluding Learned (LND) due to the absence of action logs. Beyond training models on real datasets, we extended our approach by training RL4IM on synthetic graphs due to its scalability issue, inspired by the methodology employed by Chen *et al.* [4]. The best result obtained from different training settings was considered for evaluation.

4.1 Training Time for Deep-RL Methods

For fairness, we imposed a 24-hour training cap, selecting the optimal checkpoint based on the validation dataset. The training time required to reach this checkpoint (utilizing the WC model for IM) and the number of queries (k=200) executable by traditional SOTA methods within this timeframe across four datasets are detailed in Tab. 2. The results highlight that, while Deep-RL methods undergo extended training periods, traditional SOTA methods can execute queries numbering in the tens of thousands during the same interval, even on graphs exceeding 100 million nodes.

4.2 Performance on MCP

In this section, we conduct a benchmark of Normal Greedy, Lazy Greedy, S2V-DQN, GCOMB, and LeNSE in the context of the Maximum Coverage Problem (MCP). The selective results are depicted in Fig. 4, with additional results for the remaining 12 datasets provided in the Appendix of our extended version.

Method	Training	#Queries answered within training time			
	Time (mins.)	Pokec	Wiki Talk	Live Journal	Orkut
S2V-DQN	432.4	29522	16111	12506	5750
GCOMB-MCP	22.3	1523	831	645	297
LeNSE-MCP	105.5	7203	3931	3051	1403
GCOMB-IM	312.8	9669	44056	58467	53167
LeNSE-IM[3]	96.6	2986	13606	18056	16419
RL4IM	76.3	2359	10746	14262	12969
Geometric-QN	17.8	550	2507	3327	3025

Table 2: Training time of each method and number of queries answered by traditional methods (Lazy Greedy for MCP and IMM for IM) on four datasets in training time

Effectiveness. Both Normal Greedy and Lazy Greedy yeild an $(1-\frac{1}{e})$-approximation solution. As evidenced by the results in Fig. 4. The performance of these two methods is comparable. GCOMB outperforms S2V-DQN, sometimes approaching or reaching the level of the greedy algorithms, consistent with the findings in [3]. However, other Deep-RL methods exhibit significantly poorer performance compared to Greedy. While GCOMB generally performs closely to Lazy Greedy, there are instances (e.g., on Digg, Skitter, and Higgs) where Lazy Greedy still considerably outshines GCOMB. However, we do not observe any marked distinction in the graph characteristics among these datasets [38], referring to the discussion about the "same graph distribution" claim (§ 5.1).

Efficiency. Fig. 4 illustrates the computational efficiency of GCOMB, which operates 1 to 2 orders of magnitude faster than S2V-DQN and over 2 orders faster than LeNSE in most scenarios. Particularly with a small budget, it surpasses the runtime of Normal Greedy by up to two orders, consistent with the findings in [3]. The runtime of GCOMB exhibits fluctuations rather than a steady increase, primarily attributed to the varying number of good nodes predicted by its node pruner [4]. In contrast, LeNSE takes over 10× longer than Normal Greedy.

Lazy Greedy runs more than one order of magnitude faster than GCOMB when the budget is small, and this gap widens to up to two

[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.
[3]The results for LeNSE are reported after efficiency optimization (refer to Appendix in our full paper).

[4]For a more comprehensive experimental analysis, refer to the detailed results in the Appendix of our full paper

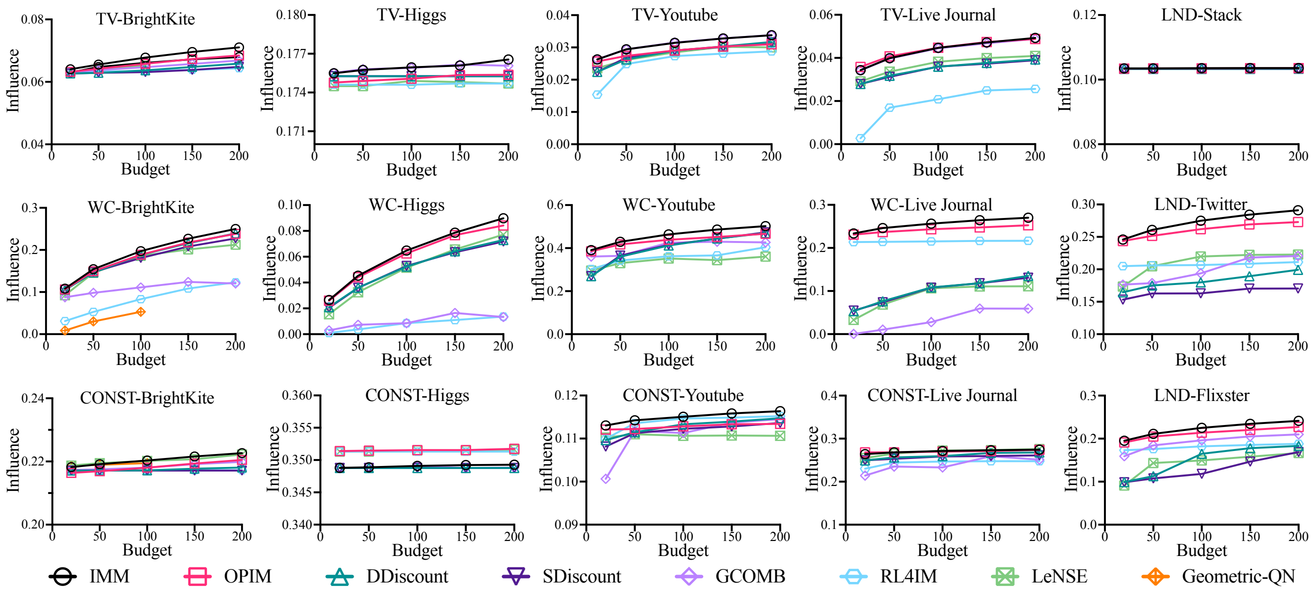


Figure 5: IM: influence curve under different weight models

	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	3.93	3.89
Geometric-QN	0.28	0.3559	/	/	/

Table 3: Memory usage (Gbyte). The upper part of the table records the 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) models

orders of magnitude as the budget increases. Additionally, we conducted tests on a billion-sized graph, Friendster. Lazy Greedy successfully solves the problem within minutes. In contrast, GCOMB crashes in our experimental environment, so we compare it with the result reported in [3], which is two orders of magnitude slower than Lazy Greedy. Note that **the runtime of Deep-RL methods only counts inference time**, excluding the extra time of the preprocessing and training phases. Despite this, even when Deep-RL methods are given some unfair advantages in the comparison, Lazy Greedy consistently outperforms all Deep-RL methods.

Memory Usage. Tab. 3 provides insights into the memory consumption of each method across representative datasets. In the inference phase, Deep-RL methods exhibit a memory footprint at least 3× larger than Lazy Greedy and 78× larger than Normal Greedy. Importantly, Deep-RL algorithms typically impose even higher memory demands during the training phase.

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 assess the performance of Deep-RL methods, namely GCOMB, RL4IM, and Geometric-QN, while also benchmarking traditional algorithms like IMM, OPIM, Degree Discount, and Single Discount in our IM experiments. All algorithms underwent testing across four edge weight models: CONST, TV, WC, and LND. It is noteworthy that none of the Deep-RL studies [1–5] have explored their methods under the WC model. This model, arguably the most prevalent in the IC model for IM literature, is included in our evaluation for comprehensive insights.

Effectiveness. As shown in Fig. 5, in line with the findings presented in [3], GCOMB performs comparably to IMM on Youtube under TV and on Stack under LND. However, it slightly lags behind IMM on Youtube under CONST. RL4IM exhibits greater stability than GCOMB and delivers a more effective solution, particularly under the CONST model. Despite being the most effective among the learning methods in various cases, LeNSE still falls short when compared to classical algorithms. Notably, these learning methods *display limited effectiveness across different datasets*, suggesting **poor generalizability**. It is worth highlighting that instances where Deep-RL methods match the effectiveness of IMM are characterized by situations where *the influence spread does not increase with an expanding budget*. In such **atypical cases**, the influence spread is *primarily governed by a few nodes*, making marginal increments subtle and challenging to observe. In such instances, IMM may exhibit inefficiency due to the subtle differences in the marginal gain of nodes, necessitating the generation of numerous RR sets to

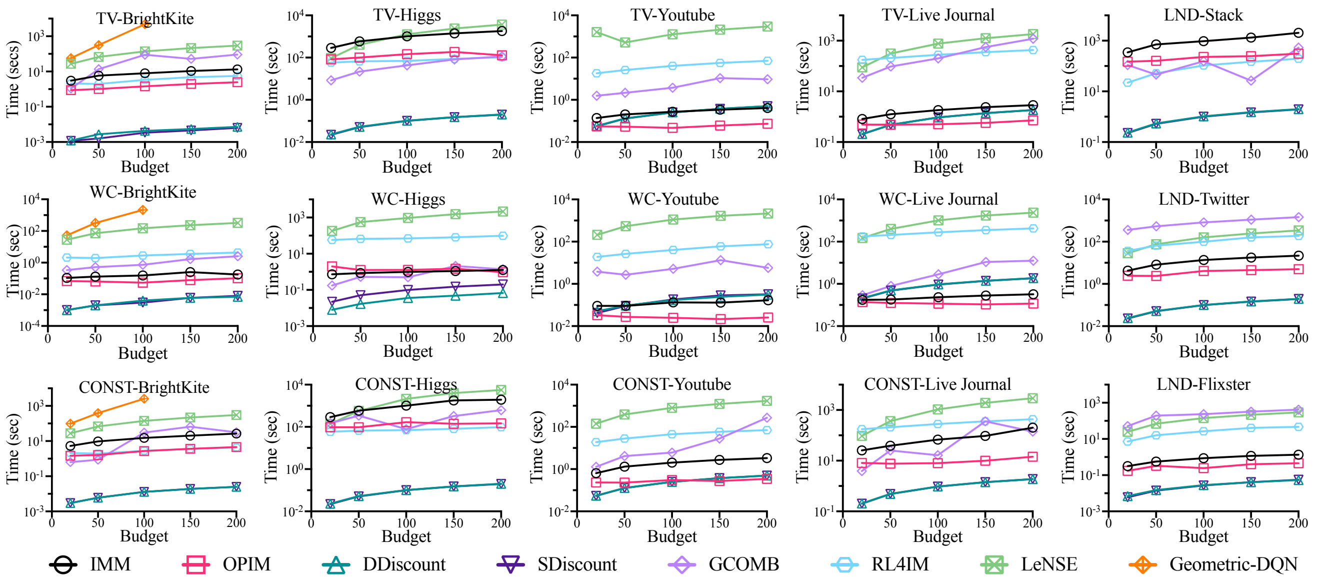


Figure 6: IM: runtime curve under different weight models.

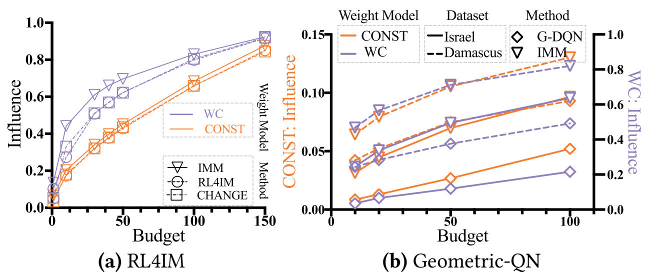


Figure 7: (a) Comparison between RL4IM, CHANGE, and IMM over synthetic graphs under the given weight model. The coverage is an average of the results of 10 repeated experiments. (b) Comparison between Geometric-QN and IMM over small-scale datasets under the given weight model. The coverage is an average of the results of 20 repeated experiments.

distinguish potential candidate nodes. Furthermore, under the WC or LND model, Deep-RL methods consistently underperform IMM.

We observe that IMM remains the most effective algorithm, with OPIM exhibiting similar effectiveness. Especially under WC model and LND model, the performance gap between theoretically sound algorithms (IMM and OPIM) and Deep-RL methods is especially prominent. Surprisingly, even discount algorithms, despite being heuristics, outperform Deep-RL methods across most cases. *These results cast doubt on the effectiveness of Deep-RL methods compared to traditional IM algorithms.*

Additional Evaluation over Synthetic Datasets. Due to the observed poor performance of RL4IM and Geometric-QN on large-scale real-world datasets, we conducted additional experiments to evaluate their effectiveness on smaller datasets as suggested in the respective papers. RL4IM was trained following the instructions in [4], and all methods repeated the query over ten times, calculating the average result. In line with the findings in [4], Fig. 7 illustrates that RL4IM outperforms CHANGE on synthetic graphs with 200 nodes and a small budget. However, extending the experiment to larger graphs with 2000 and 20,000 nodes under CONST and WC models, RL4IM consistently surpasses CHANGE but falls short of IMM. This indicates that *while* RL4IM *performs well in small synthetic graphs, it remains inferior to* IMM *in this context.* Regarding Geometric-QN, [2] reported that it obtains 37.8% and 61.1% of the influence scores of the greedy algorithm for the datasets Israel and Damascus, respectively. As IMM provides the same approximation ratio as the greedy algorithm, we directly compared Geometric-QN with IMM in our experiments. Due to the high variance of Geometric-QN, we repeated the query over 20 times and then calculated the average as the final result. Our experiment showed that Geometric-QN achieves 27.5% and 66.1% of the coverage of IMM in Israel and Damascus, respectively. Though unlike what [2] reported, Geometric-QN unmistakably lags behind IMM.

Efficiency. It is important to note that we provide unfair advantages to Deep-RL methods by excluding their pre-processing or training time. Despite this, as shown in Fig. 6, traditional IM algorithms are still 10× to 10,000× faster than Deep-RL methods in most cases. However, in scenarios where the influence spread hardly increases with the budget, such as in Pokec, Wiki Talk, and Wiki Topcats under the CONST model, there are numerous solution sets with very similar influence spread, i.e., the atypical cases discussed in the effectiveness assessments. Distinguishing these highly similar solution sets requires generating many RR sets, making theoretically sound algorithms like IMM and OPIM slow in such situations.

With the help of node pruning techniques, GCOMB can sometimes achieve speeds that are orders of magnitude faster than IMM. However, the runtime of GCOMB is often non-monotonic concerning the budget, indicating that the node pruner cannot guarantee a smaller search space for a small budget compared to a large budget (e.g., TV-BrightKite, CONST-Higgs, CONST-Live Journal). The instability of the node pruner leads to GCOMB being orders of magnitude slower than IMM in certain cases.(eg. WC-Youtube, TV-Live-Journal, and more cases displayed in our full version).

	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
Clust. coe.	0.5	-0.6	-0.6	-0.728	-0.486	-0.429	-0.712	-0.829	-0.543	-0.712	-0.314	-0.6
Triang. (%)	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
Eff. 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
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
VCI (%)	-0.524	0.486	0.371	0.652	0.429	0.486	0.6	**0.886**	0.6	0.6	0.429	0.771
Sum_{10} (%)	-0.476	-0.029	0.029	0.612	0.6	0.486	0.6	0.6	0.6	0.6	0.486	0.486
weighted degree	-	-	-	0.486	0.429	0.371	0.2	-0.371	0.257	0.371	-0.314	-0.6
edge weight	-	-	-	0.371	0.257	0.371	0.143	0.543	0.257	0.486	**0.829**	0.657
Community Structure	-	-	-	**0.942**	**0.812**	**0.952**	**0.912**	**0.907**	**1.0**	0.643	-0.351	0.398
WL kernel	-	-	-	0.621	**0.882**	0.636	0.515	0.135	0.321	**0.922**	0.0	-0.523
PageRank	-	-	-	-0.653	-0.716	-0.636	-0.475	0.132	-0.334	-0.653	0.366	0.73

Table 4: Correlation of Graph Metrics with Coverage Gap using Spearman's Coefficient: Highlighting Values ≥0.8. The upper section represents unweighted topological metrics, the middle section denotes weighted metrics, and the lower section outlines the complex metrics.

In contrast to 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 slower than other methods and incapable of finding solutions for large datasets like Orkut and Stack within a limited time. Geometric-QN employs a computationally expensive real-time graph exploration policy during the inference process, rendering it even non-scalable to moderate-size Higgs dataset.

Memory Usage. The lower section of Tab. 3 presents the peak memory usage of algorithms in the IM experiments. Memory consumption by Deep-RL methods varies significantly. Geometric-QN is memory-efficient, but it takes considerably more time to find a solution, leading to poor scalability. As discussed in the efficiency analysis, IMM and OPIM need to generate a large number of RR sets in atypical scenarios, resulting in substantial memory usage. Conversely, in other situations, IMM and OPIM tend to consume less memory than Deep-RL methods.

Summary. Based on the above evaluations, 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.*

5 COMMON ISSUES OF DEEP-RL METHODS

The results presented in § 4 contradict the expectations of all the Deep-RL methods [1–5], which aim to efficiently learn better approximations of the coverage function or the influence function, leading to more effective and efficient solutions for MCP or IM. Given the notable performance disparity between Deep-RL methods and Lazy Greedy in MCP, this section shifts its focus to examine the application of Deep-RL in IM. Additional experiments are conducted to unveil prevalent issues within the Deep-RL methods [1–5] that hinder their practical effectiveness.

5.1 Study of Graph Distribution

In § 3.4, we delve into the limitations posed by the worst-case performance of Deep-RL methods, highlighting the *theoretical boundary* of $1 - \frac{1}{e}$. Moreover, we emphasize that the practical performance of

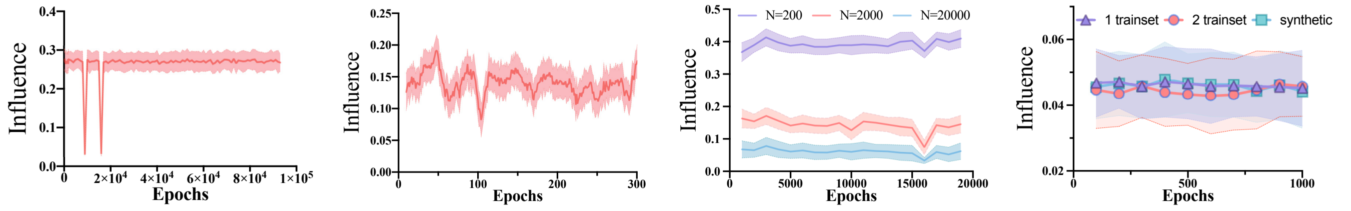
	TV			WC		
	GCOMB	RL4IM	LeNSE	GCOMB	RL4IM	LeNSE
BrightKite	1.29%	-0.85%	-0.19%	30.18%	26.52%	-11.73%
Amazon	21.74%	27.23%	-7.69%	7.64%	65.52%	36.45%
DBLP	86.81%	43.58%	-18.18%	-5.49%	87.61%	43.77%
Wiki Talk	2.46%	-2.26%	7.59%	6.39%	83.10%	10.20%
Youtube	59.22%	13.26%	3.18%	0.61%	2.80%	3.33%

Table 5: Percentage change of the performance

these methods in scenarios beyond the worst-case is contingent on their *generalizability*. Deep-RL studies [1–5] assert that Deep-RL methods can learn heuristics to solve combinatorial problems on graphs and generalize effectively to graphs resembling the training distribution. This assertion prompts an evaluation of the similarity between the distributions of the training and test graphs. However, these studies *lack a rigorous definition of "graph distribution"*. In practical terms, using trained Deep-RL models for MCP and IM requires an efficient method to determine whether a new input graph conforms to the "same distribution" as the training graphs. We explore the feasibility of leveraging *easy-to-compute statistics of graphs* to ascertain if two graphs follow the "same distribution".

Edge Weights Matter in IM. We begin by highlighting the significance of edge weights in IM. Our investigation aims to assess the generalization capability of trained Deep-RL models when applied to *the same graph under edge-weight models different from the model used during training*. To illustrate, 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. Adhering to the experimental setup from baseline papers, we select CONST as the training model. Subsequently, we evaluate the performance of $\mathcal{F}_{CO}$ against $\mathcal{F}_M$ across five graphs, utilizing the edge weight model M with a budget of $k = 50$. Tab. 5 lists the percentage change of the performance p, where $p = \frac{\mathcal{F}_M(G_M) - \mathcal{F}_{CO}(G_M)}{\mathcal{F}_M(G_M)}$. Here, $\mathcal{F}_{CO}(G_M)$ means $\mathcal{F}$ is trained under CONST while tested on G_M, with $M \in \{WC, TV\}$. A larger absolute value indicates greater sensitivity of the method $\mathcal{F}$ to the edge weight model. The results suggest that these Deep-RL methods *struggle to generalize effectively across different edge-weight models.*

Simple topology statistics do not help. Next, we investigate whether commonly adopted topology statistics of graphs can help



(a) GCOMB (b) LeNSE (c) RL4IM (d) Geometric-QN

Figure 8: Performance Curves of Models with Varying Training Durations under the WC Model. (a-b): Training and validation datasets are 15% subgraphs from YouTube, with the test dataset from the remaining 70% edges, as in [3]. (c): Training on 200 synthetic graphs (200 nodes each); testing on synthetic graphs of varying number (N) of nodes. (d): Following [2], models are trained on different datasets, including the Copen dataset (1 trainset), both Copen and Occupy datasets (2 trainset), and synthetic graphs (synthetic)

	DBLP			Wiki Talk		
	CONST	TV	WC	CONST	TV	WC
Community	1625.5	371.7	1730.9	45.4	56.6	1851.5
WL Kernel	258.0	73.3	382.1	4.3	5.5	243.2
PageRank	79.8	22.9	120.8	2.3	2.7	120.5

Table 6: Ratio of Execution Time: Advanced Graph Similarity Calculation Approaches to OPIM Query Resolution with $k = 200$

discern whether a testing graph follows the "same distribution" as the training graphs. In this context, we employ the same edge-weight model for both training and testing data. In addition to unweighted topological statistics (1)-(9) mentioned in the dataset introduction, for IM, we also incorporate topological metrics associated with edge weights, such as (10) the average weighted degree and (11) the average edge weight.

We calculate the Spearman correlation coefficient[5] between the coverage gap $\delta^{n\times 1}$ and m topology statistics of graphs $\mathcal{X}^{n\times m}$ across n datasets. Here, $\delta_i = \frac{\mathcal{F}(\cdot)_i - OPT_i}{OPT_i}$, where OPT_i is approximated by the coverage obtained over dataset i by greedy in MCP and IMM in IM, and $\mathcal{F}(\cdot)$ is the coverage or influence obtained by a Deep-RL method $\mathcal{F}$. Tab. 4 shows the results. Unfortunately, *strong positive correlations (≥ 0.8) are relatively rare.* What's worse, even when a statistic has a strong positive correlation with a particular method, the strong correlation will no longer exist once we change the edge-weight model (e.g., the VCI(%) for GCOMB TV and WC).

Derivation of Complex Topological Statistics is Computationally Intensive. In addition to straightforward and easy-to-compute topological statistics of graphs, we delve into intricate methods like Weisfeiler-Lehman (WL) Graph Kernel [44], PageRank, and Louvain community detection [45] to ascertain similarity for weighted graphs in IM. Insights from Tab. 4 illustrate that the *WL graph kernel and PageRank are markedly inefficient in pinpointing graphs with analogous distributions.* On the other hand, *analogous community structures effectively discern similar distributions in the TV and CONST models.* However, mirroring the outcomes observed in learning approaches, community structures under the WC model fall short of capturing the similarities between graphs. This indicates the potential significance of community structures as pivotal indicators while assessing similar distributions *prior to inference.* Nevertheless, the precision offered by these advanced metrics comes with the trade-off of being impractical for real-time or large-scale applications due to their *substantial computational overhead.* Tab. 6 presenting results from both a small and a large dataset, accentuates that the computation of these metrics far exceeds the time frame required by the state-of-the-art OPIM to resolve a query when $k = 200$. It's noteworthy that, in practical scenarios, *IM queries do not necessarily demand high throughput.* Identifying seed sets for campaigns in a social network, for instance, typically involves a manageable number of candidates rather than thousands or hundreds of thousands.

Summary. These evaluations imply that determining whether a testing graph aligns with the "same distribution" as the training graphs might be challenging in practical applications. Consequently, assessing the performance of a trained Deep-RL model on a testing graph may require actual execution on the specific graph, as predicting its effectiveness beforehand could be a non-trivial task.

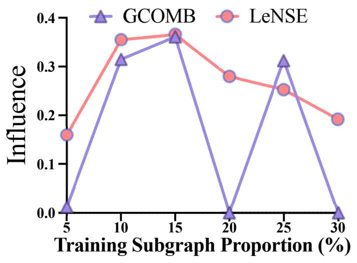
5.2 Impact of Training Time

Deep-RL methods frequently encounter convergence challenges in practical scenarios [46]. To investigate the convergence behavior of Deep-RL methods for MCP and IM [1–5], we conduct experiments by keeping the training data constant while varying the training time (epochs). It is essential to note that we extend the training duration significantly beyond that reported in the original papers and select the best checkpoint based on the validation set. Despite these efforts, the Deep-RL methods consistently exhibit inferior performance compared to IMM. GCOMB (Fig. 8a) initially has an unusual performance: drop but then tends to converge as the number of training epochs increases. LeNSE (Fig. 8b) demonstrates effective learning initially but encounters intermittent performance drops. RL4IM (Fig. 8c) exhibits a steady performance improvement until approximately 3,000 epochs, beyond which no further enhancement is observed, indicating potential overfitting and diminished generalizability with prolonged training. Similarly, Geometric-QN (Fig. 8d) fails to learn efficiently as the training steps increase. In summary, extending the training time for Deep-RL methods may not consistently lead to improved performance, posing challenges in determining an optimal training duration and making the tuning process intricate in practical applications.

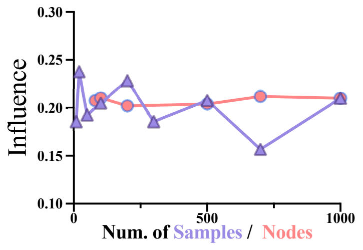
5.3 Impact of Training Dataset Size

In this study, we address the impact of training dataset size on the performance of Deep RL methods, acknowledging the challenges associated with acquiring datasets with high-quality labels.

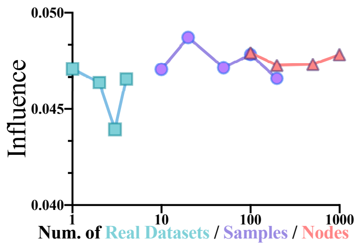
[5]The Spearman correlation coefficient is a widely-adopted metric to measure the strength and direction of association between two ranked variables [43].



(a) GCOMB & LeNSE



(b) RL4IM



(c) Geometric-QN

Figure 9: Performance Curves of Models under the WC Model Based on Training Dataset Size. (a): x-axis represents the percentage of YouTube edges used for training. (b): x-axis denotes the number of samples (200 nodes each) or nodes (200 samples total). (c): x-axis indicates the number of synthetic samples, nodes in a synthetic graph, or the number of real datasets.

The training approaches for Deep RL methods in our investigation fall into two categories based on the size of the training datasets. GCOMB and LeNSE are trained on fractional subsets of a dataset, while RL4IM and Geometric-QN 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). To systematically explore the influence of training dataset size, we adopt distinct strategies for each category. For GCOMB and LeNSE, we train these models using various subgraphs, each comprising no more than 30% of the nodes in the Youtube dataset. Subsequently, we evaluate the models on the same graph, constructed with the remaining 70% of edges unseen during the training phase. Following the methodology outlined in [2], we train Geometric-QN on different numbers of real datasets or synthetic graphs. In the case of RL4IM and Geometric-QN trained on synthetic graphs, we not only vary the number of samples while keeping the number of nodes fixed but also train the models with a fixed number of samples, each having different numbers of nodes.

The experimental results reveal that *none of the methods consistently 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, Fig. 9(a) shows that GCOMB and LeNSE achieves their optimal performances when trained on a subgraph of 15% size, but their performance drastically drops as the training dataset size increases, indicating instability. In contrast, Fig.9(c) reveals that RL4IM displays more stability, with a subtle trend suggesting 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. 9(d) indicates that increasing the number of training graphs does not improve the performance of Geometric-QN either. These results underscore the critical insight that *simply enlarging the training dataset size may have detrimental effects on the model's performance.* Therefore, determining *an appropriate volume of training data* poses a significant challenge.

6 RATING SCALE FOR EACH SOLVER

Fig. 1 in § 1 illustrates the relationship between the average normalized coverage or influence (y-axis) and the average normalized runtime (x-axis) across 16 datasets in MCP and 8 datasets under three weight models in IM. A position closer to the top-left corner indicates a more desired performance – faster and more effective. Higher robustness is suggested by a lower standard deviation value in effectiveness, while a higher value indicates the opposite. Among Deep-RL methods in MCP, GCOMB offers coverage comparable to greedy algorithms and outpaces the standard greedy approach. However, it consistently falls short when compared to Lazy Greedy. In IM, Deep-RL methods even lag behind simple heuristic methods like SDiscount and DDiscount. It's noteworthy that the presented runtimes exclude the preprocessing and training time of these Deep-RL methods. Despite this, they remain significantly less efficient than traditional algorithms.

Based on the observed results across a wide range of datasets and settings, a rating scale is summarized in Tab. 7. Here, the metric *Quality* for method f is defined as the average of $\{\frac{c_d^{(f)}}{\text{Max}(c_d)}|d \in \text{D}\}$, where c represents the coverage value, and d denotes a selected dataset within the entire set of datasets D. Similarly, the metric *Efficiency* is defined as the average of $\{\frac{\text{Max}(t_d)}{t_d^{(f)}}|d \in \text{D}\}$, where t signifies the runtime. Furthermore, we delineate the metric *Robustness* for method f as the normalized reciprocal standard deviation of its quality. Please note that the rating for Geometric-QN, due to its limited scalability, is not derived from a direct comparison with others across D. Instead, it is derived by comparing it specifically with IMM on the smaller datasets, as detailed in Kamarthi et al. [2].

Method	Quality(%)	Memory(%)	Efficiency(%)	Robustness(%)
MCP				
Normal Greedy	99.73	<u>100</u>	0.97	98.53
Lazy Greedy (2007)	<u>100</u>	37.78	<u>100</u>	<u>100</u>
S2V-QN (2017)	87.40	18.49	0.86	9.53
GCOMB (2020)	99.11	13.27	7.43	91.58
LeNSE (2022)	71.91	14.70	0.04	2.50
IM				
Degree Discount (2009)	89.77	96.32	82.59	8.54
Single Discount (2009)	89.15	<u>96.88</u>	<u>84.09</u>	8.33
IMM (2015)	<u>99.44</u>	60.20	12.86	<u>100</u>
OPIM (2018)	96.36	73.85	35.69	43.54
Geometric-QN (2020)*	44.66	33.98	<0.01	5.08
GCOMB (2020)	67.71	15.15	3.06	3.41
RL4IM (2021)	66.12	57.60	0.26	3.64
LeNSE (2022)	78.44	29.90	0.02	5.35

Table 7: Rating scale for each solver: In each metric, higher values indicate better performance. The highest values are underlined.

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 [47–50] and the Multiple Knapsack Problem [51], have been extensively studied, finding applications in facilities location [8, 52], maximum coverage in streams [53], and information retrieval [54], etc.

The Influence Maximization (IM) problem [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++ [55] significantly reduced the Monte Carlo evaluation times while maintaining a $(1 - \frac{1}{e})$ approximation. Other heuristic algorithms [35, 56, 57], 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) 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 [18], 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 [58], natural language processing [59], and automatic control [31] 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. [60, 61] have discussed these limitations, providing insights into the challenges faced by DL. Furthermore, [62] has 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. A recent contribution [63] proposes 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 POTENTIAL DIRECTIONS FOR IMPROVING DEEP-RL METHODS

Deep-RL methods currently demonstrate excellence primarily in abnormal scenarios with limited practical optimization value, emphasizing their current limitations. We briefly outline challenges and limitations that require addressing when applying Deep-RL methods to IM in this section.

Identify similar distribution effectively. As discussed in § 3.4, the upper bound of the worst-case performance for Deep Reinforcement Learning (Deep-RL) methods on MCP and IM is constrained, i.e., not exceeding $1 - \frac{1}{e}$. Furthermore, the effectiveness of these methods in scenarios beyond this worst-case heavily depends on their generalizability. This necessitates a thorough assessment of the alignment between the distributions of the training and test graphs. Evidence from experimental results and analyses presented in § 5.1 suggests that this issue remains unresolved. One critical open question is to identify a few easy-to-compute measures to quantify the learned graph distribution.

Extract high-quality query subspace. The Deep-RL methods often struggle with inefficiency, primarily due to the vastness of the search space they need to explore. To mitigate this, certain Deep-RL approaches aim to enhance efficiency without compromising solution quality by identifying a subgraph within the entire graph that contains the high-quality solution, effectively reducing the search space. However, this strategy necessitates a delicate balance between the time consumption in identifying the quality of this subspace and the precision of such identification. Methods like Geometric-QN and LeNSE tend to spend more time extracting this subspace compared to the time saved during the querying process. Conversely, while GCOMB improves query efficiency by efficiently pruning the search space, it struggles to consistently maintain solution quality. Overall, a substantial challenge persists in effectively extracting a high-quality search subspace.

Initialization with prior knowledge. Initializing the state with prior knowledge, as opposed to random initialization, holds the potential to enhance performance. Geometric-QN, for instance, constructs a subgraph based on random initialization, leading to a high variance in the final performance. Similarly, LeNSE, by initializing a subgraph randomly, incurs a time-consuming process for subgraph exploration. This suggests that adopting prior knowledge for initialization could potentially streamline exploration processes and yield more stable performance.

9 CONCLUSIONS

In this benchmark study, we conducted a comprehensive examination of the effectiveness and efficiency of five recent Deep-RL methods for MCP and IM. The experimental results reveal that, for both MCP and IM, traditional algorithms such as Lazy Greedy and IMM consistently outperform Deep-RL methods in the majority of cases, if not all. Additionally, we highlight common issues observed in Deep-RL methods for MCP and IM, emphasizing the challenge of predicting the performance of a trained Deep-RL model on a testing graph before actual execution. Finally, we discuss potential reasons why Deep-RL may not be an optimal solution for MCP and IM. Our benchmark study sheds light on possible challenges and limitations in current deep reinforcement learning research aimed at 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, 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*, 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] 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.

[19] Jasmina Malicevic, Baptiste Lepers, and Willy Zwaenepoel. Everything you always wanted to know about multicore graph processing but were afraid to ask. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 631–643, 2017.

[20] Noga Alon and Baruch Schieber. *Optimal preprocessing for answering on-line product queries.* Citeseer, 1987.

[21] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms.* MIT press, 2022.

[22] Michael J Kearns and Umesh Vazirani. *An introduction to computational learning theory.* MIT press, 1994.

[23] Akhil Arora, Sainyam Galhotra, and Sayan Ranu. Debunking the myths of influence maximization: An in-depth benchmarking study. In *Proceedings of the 2017 ACM international conference on management of data*, pages 651–666, 2017.

[24] Xiangyu Ke, Arijit Khan, and Leroy Lim Hong Quan. An in-depth comparison of s-t reliability algorithms over uncertain graphs. *Proc. VLDB Endow.*, 12(8):864–876, 2019.

[25] 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.

[26] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. A data-based approach to social influence maximization. *Proc. VLDB Endow.*, 5(1):73–84, sep 2011.

[27] Wenlong Liao, Birgitte Bak-Jensen, Jayakrishnan Radhakrishna Pillai, Yuelong Wang, and Yusen Wang. A review of graph neural networks and their applications in power systems. *Journal of Modern Power Systems and Clean Energy*, 10(2):345–360, 2021.

[28] Martijn Van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement learning: State-of-the-art*, pages 3–42. Springer, 2012.

[29] 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.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[31] 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.

[32] 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.

[33] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

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

[35] 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.

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

[37] Zhicheng Liang, Yu Yang, Xiangyu Ke, Xiaokui Xiao, and Yunjun Gao. Our source code and dataset repository. https://anonymous.4open.science/r/MCPBenchmark-B2BF, 2024.

[38] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[39] R Duncan Luce and Albert D Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.

[40] Duncan J Watts. *Six degrees: The science of a connected age.* WW Norton & Company, 2004.

[41] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[42] 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.

[43] Charles Spearman. The proof and measurement of association between two things. 1961.

[44] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[45] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[46] 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.

[47] 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.

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

[49] 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.

[50] 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.

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

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

[53] 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.
[54] 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.
[55] 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.
[56] 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.
[57] 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*, pages 127–132, 2011.
[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.
[59] 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.
[60] 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.
[61] Francois Chollet. The limitations of deep learning. *Deep learning with Python*, 2017.
[62] Carla Zoe Cremer. Deep limitations? examining expert disagreement over deep learning. *Progress in Artificial Intelligence*, 10:449–464, 2021.
[63] 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.

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 8: Training Time of Noise Predictor (Sec.)

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%

Table 9: Number of Non-Noisy Nodes

A 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.

A.1 Training time of Noise Predictor

The noise predictor training method proposed in [3] isn't consistently feasible. In our tests, the noise predictor didn't generalize well to unseen budgets. Consequently, we followed the author's experimental approach, training a distinct noise predictor for each budget. Table 8 details the time required to train a noise predictor for each budget in MCP. Remarkably, this process takes thousands of times longer than solving a problem using Lazy Greedy.

A.2 Proportion of Good Nodes

Table 9 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 where 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, Pokec, Flixster and Twitter 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.

B 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 utilizing reverse influence sampling techniques for calculating influence spread, and dynamically generating training subgraphs of varying quality. As for MCP, we replace the normal greedy with Lazy Greedy to boost the efficiency in preprocessing and inference phase.

C ADDITIONAL RESULTS

In the MCP experiment, we present supplementary results for twelve datasets. Additionally, in the IM experiment, we display further results for six datasets under the edge weight models WC, TV, and CONST, respectively. Due to scalability issues, LeNSE and RL4IM crash on the dataset Orkut.

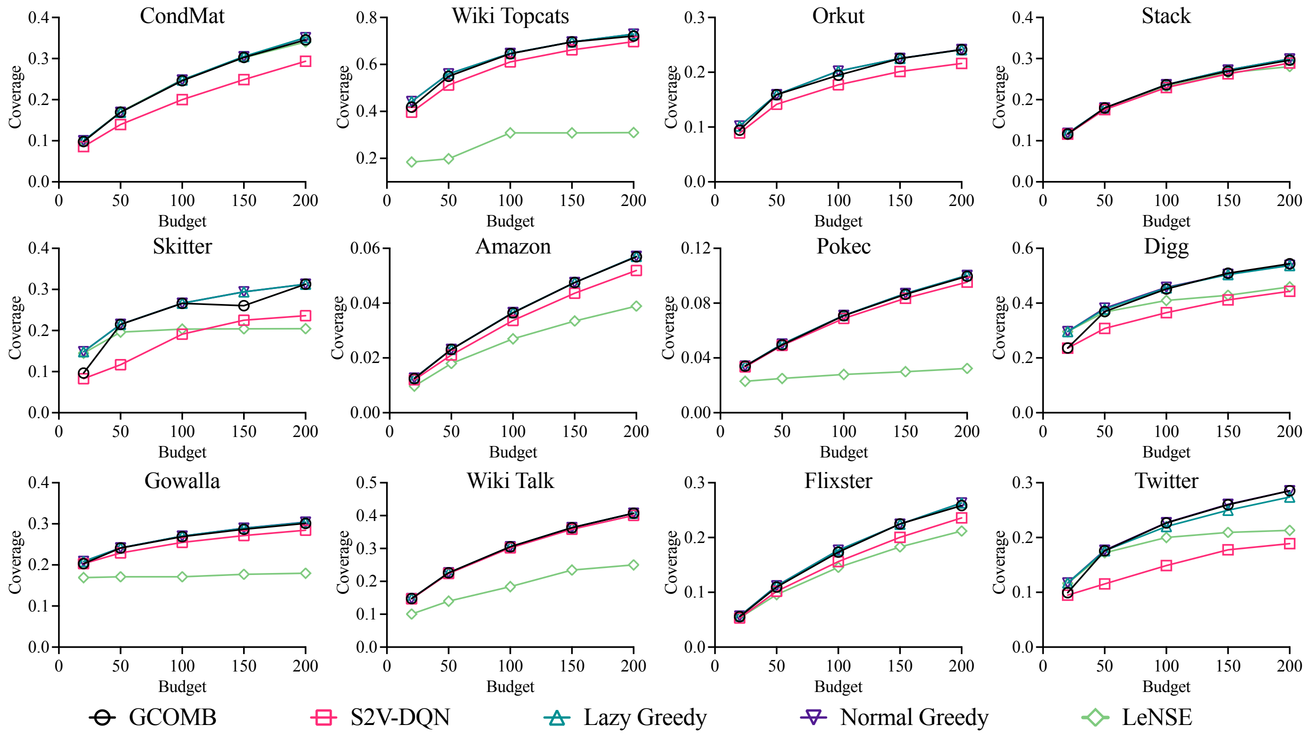


Figure 10: Additional results for MCP: coverage curve

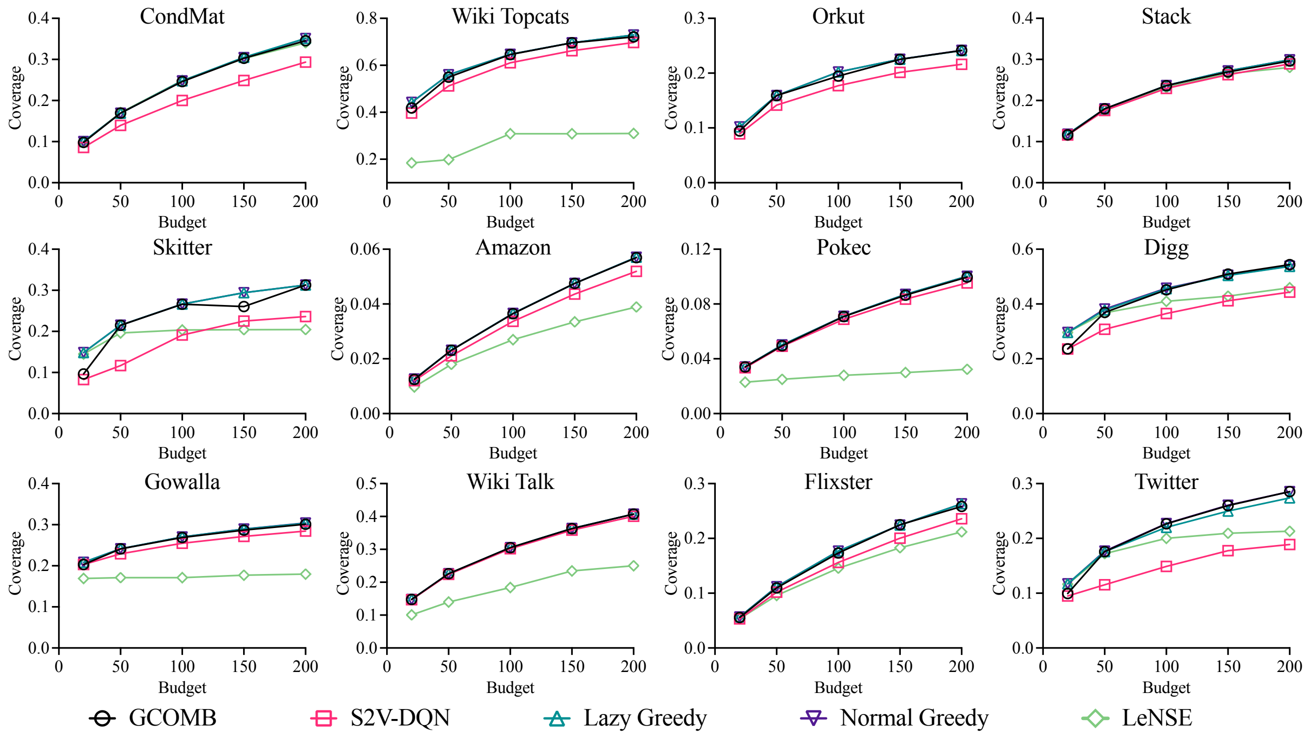


Figure 11: Additional results for MCP: runtime curve

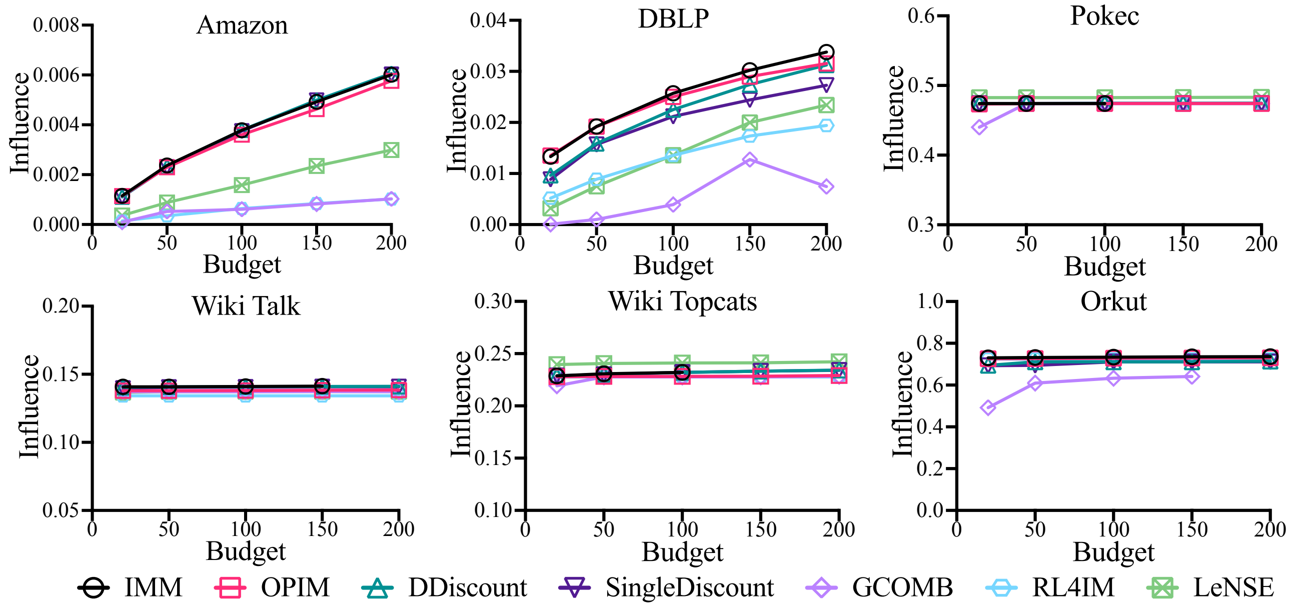


Figure 12: Additional results for IM: Influence curve under CONST

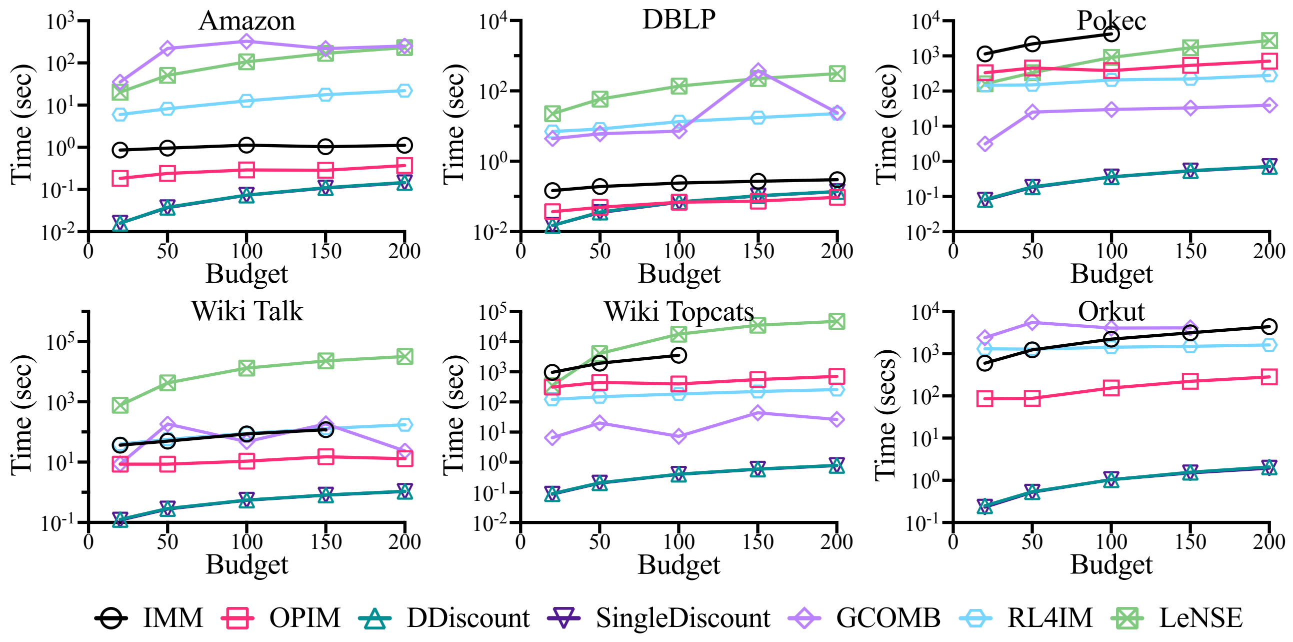


Figure 13: Additional results for IM: Runtime curve under CONST

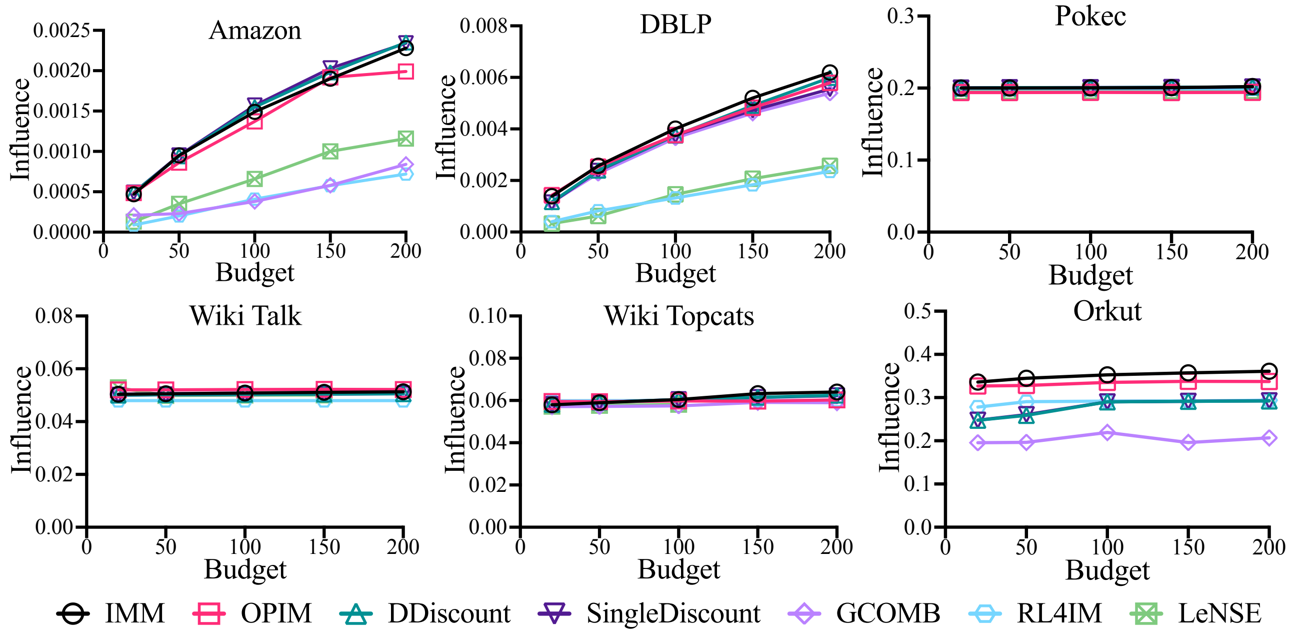


Figure 14: Additional results for IM: Influence curve under TV

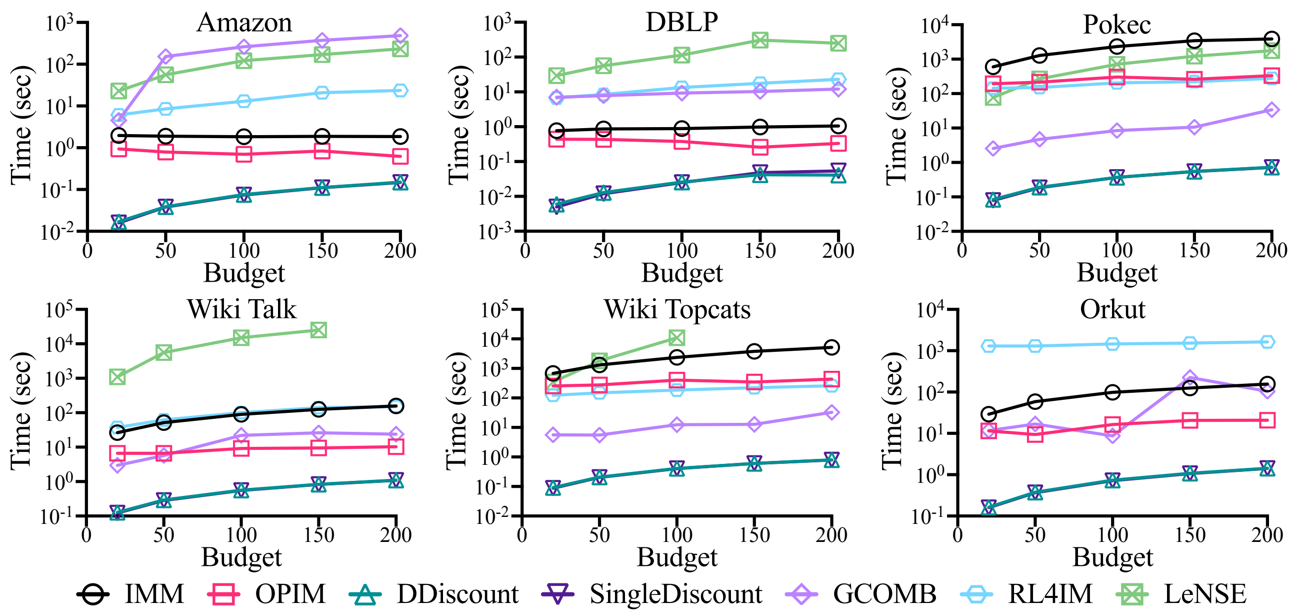


Figure 15: Additional results for IM: Runtime curve under TV

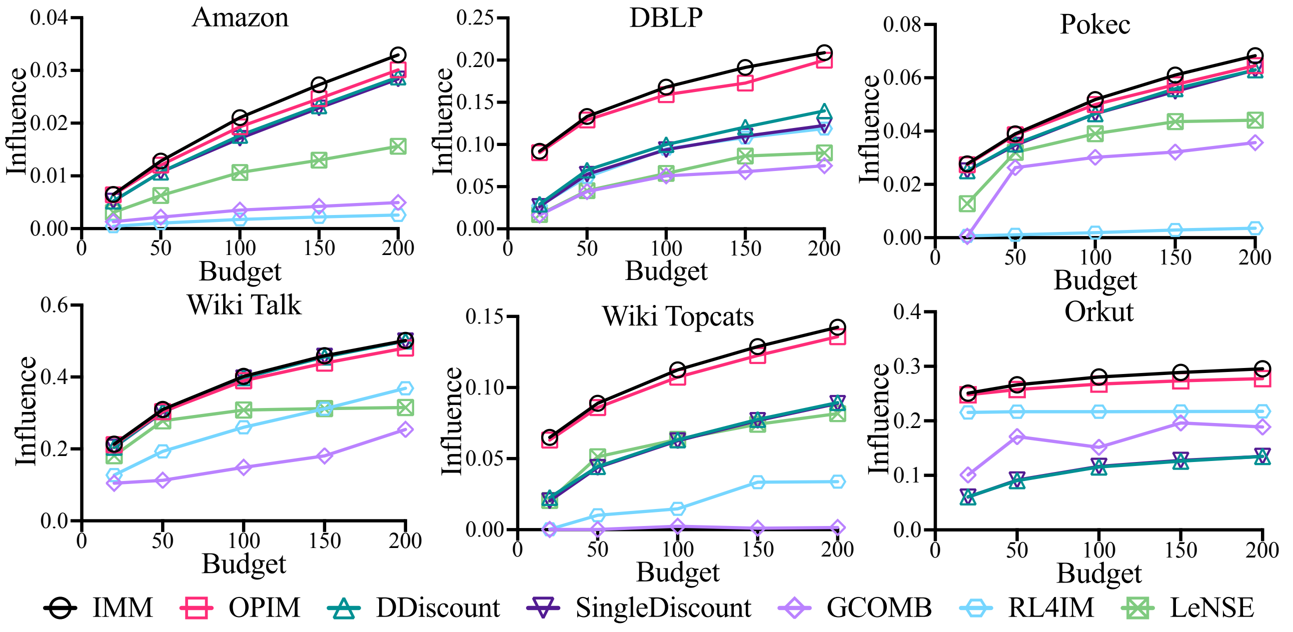


Figure 16: Additional results for IM: Influence curve under WC

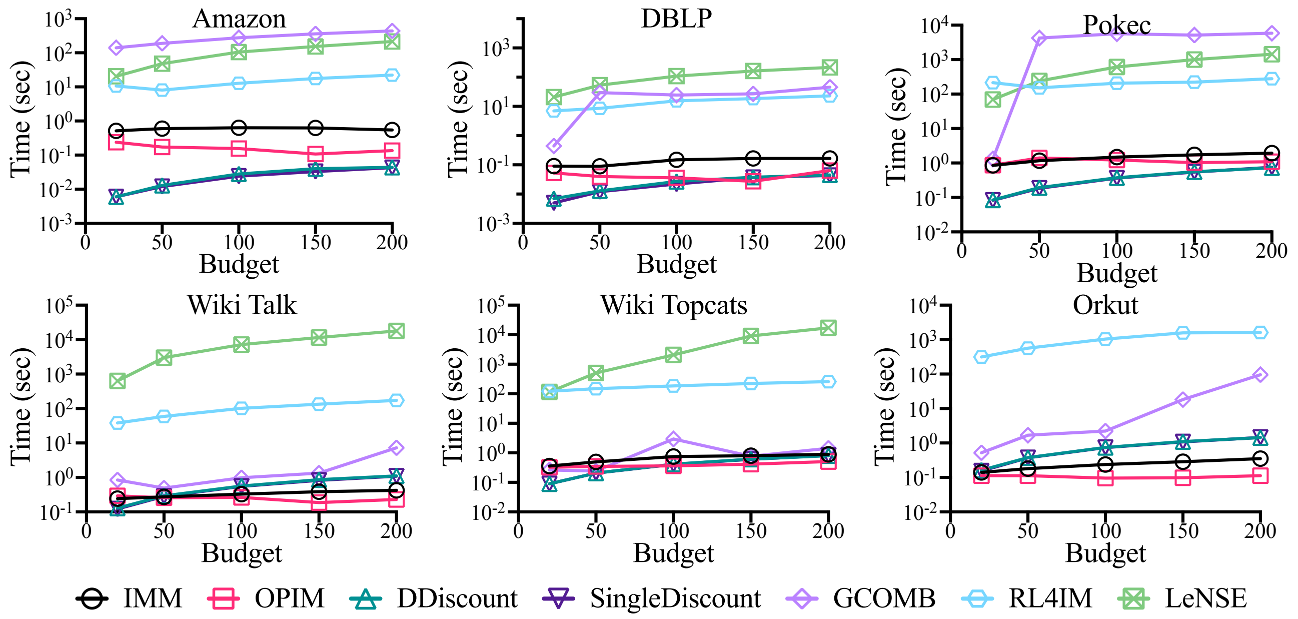


Figure 17: Additional results for IM: Runtime curve under WC