Utrecht University

Master Thesis

A Deep Reinforcement Learning Approach for Influence Maximization in Dynamic Non-Progressive Social Networks

Author:
Yunming Hui

Daily Supervisor:
Dr. S. (Shihan) Wang

Second Examiner:
Dr. M.W. (Mel) Chekol

August 22, 2023

Abstract

Influence maximization is pivotal in network analysis, identifying critical individuals for optimal information spread. This thesis introduces a novel dynamic non-progressive diffusion model, extending traditional approaches to address real-world scenarios that evolve over time. To tackle the challenges of dynamic influence maximization, this thesis proposes an innovative framework that integrates dynamic graph embedding with reinforcement learning. Experimental evaluations reveal the effectiveness of this framework, notably excelling on datasets characterized by recurring edges. While showcasing the potential of this integration, this thesis also acknowledges areas for enhancement, especially concerning sparse datasets.

Contents

1 Introduction

Influence maximization is a fundamental problem in social network analysis that focuses on identifying a subset of individuals within a network whose activation would result in the maximum spread of influence across the entire network [1, 2]. The central idea behind influence maximization is to strategically select a limited number of nodes, often referred to as seed nodes, in order to maximize the adoption of a certain behavior, information, or innovation within a social network. This problem finds applications in various fields, including viral marketing, recommendation systems, epidemic modeling, and opinion propagation [3, 4]. The influence maximization problem has attracted significant attention due to its practical implications in harnessing the power of network effects for targeted dissemination.

Influence maximization is intricately tied to diffusion models, forming the foundation of its approach [5]. Diffusion models provide a framework to simulate how information, behaviors, or innovations spread through a network. Influence maximization leverages these diffusion models to identify the most strategic set of initial nodes that would trigger the largest propagation cascade, thus capitalizing on the dynamics of influence propagation to optimize seed node selection.

Diffusion models, discussed in [6], can be classified as progressive and non-progressive (NP) models. Progressive models allow nodes to activate and stay active permanently, whereas NP models enable nodes to switch between active and inactive states [7, 8]. While much attention has been given to progressive models in information diffusion studies, these models fall short in capturing scenarios like declining user interest revitalized by external events, cyclic fashion trends, or disease dynamics [8]. Non-progressive models are essential for accurate representation of such cases.

Moreover, to accurately model influence propagation, diffusion models need to be defined on dynamic graphs [9, 10]. Real-world networks are inherently dynamic, rendering static graphs inadequate in capturing evolving interactions. Incorporating temporal aspects through dynamic graph representations allows diffusion models to better predict influence dynamics in complex social networks. Although traditionally favored for simplicity and efficiency, static graphs disregard temporal dynamics and struggle to depict evolving network interactions.

The dynamic non-progressive model is really important because it brings together two important things from real life. It combines the flexibility of non-progressive models, where nodes can become active and inactive multiple times, with the idea of looking at networks that change over time. This helps us describe situations where people's interest goes up and down, or where things happen in cycles. Regular models can't handle these well. By mixing these two ideas, the dynamic non-progressive model can understand and predict how things spread in networks that are always changing. It's like seeing the bigger picture of how things move in networks that behave in complex ways.

In summary, a dynamic non-progressive diffusion model is of great practical importance. However, to the best of my knowledge, there is no existing dynamic non-progressive diffusion model. Therefore, in this thesis a dynamic non-progressive diffusion model extended for Independent Cascade (IC) model [11], a well-known diffusion model, is proposed. And based on this, the dynamic non-progressive influence maximization problem is defined.

After examining existing research, it becomes evident that reinforcement learning offers a promising avenue for addressing influence maximization (IM) problems [12, 13, 14]. However, the current array of reinforcement learning-based methods faces challenges when confronted with dynamic IM scenarios. In light of this, this thesis introduces an innovative approach that integrates the state of the art dynamic graph embedding, TGNs, with a well-known reinforcement learning algorithm, Dueling Double DQN.

The proposed framework is rigorously evaluated through comprehensive experiments to assess its viability. Moreover, a comparative analysis is conducted between the novel framework and other contemporary methodologies on real-world datasets. The experimental findings demonstrate the framework's strong performance in datasets characterized by frequent recurring edges, while revealing room for improvement in handling sparse datasets.

In conclusion, the integration of dynamic graph embedding and reinforcement learning presents a promising avenue for enhancing the efficacy of influence maximization in dynamic networks. The experimental validation and comparative analysis provide insights into the framework's strengths and potential areas for enhancement, ultimately contributing to the advancement of dynamic influence maximization methodologies.

In the subsequent sections, this article is organized as follows. Section 2 introduce the basic concepts used in this thesis. Section 3 proposed the dynamic non-progressive diffusion model and defined the research focus of this thesis, dynamic non-progressive influence maximization problems. Some existing well-known diffusion models are also introduced. Section 4 provides an review of the existing literature on influence maximization and its associated challenges, shedding light on the gaps that motivate the need for our proposed framework. Section 5 presents the design and formulation of the proposed framework, synthesizing dynamic graph embedding and reinforcement learning for enhanced influence maximization in dynamic networks. Section 6 outlines the comprehensive experiments conducted to validate the framework's effectiveness and present a comparative analysis against state-of-the-art methods using real-world datasets, highlighting the strengths and limitations of the proposed framework. Section 7 concludes the article and suggesting potential avenues for future research in the evolving field of dynamic influence maximization.

2 Preliminaries

In this section, basic concepts used in this thesis will be introduced in case the reader is unfamiliar with them.

2.1 Social Network Models

In the paper [15], it is proposed that the representation of social networks can be characterized predominantly by three models: the static graph model, the snapshot model, and the continuous model. This is followed by this thesis. Science this is written by myself, I used what is in [15] in this subsection.

2.1.1 Static Graph Model

When using a static graph, the social network is represented by a directed graph $G = (V, E)$, where V is the node set and E is the edge set. Each node in V represents an individual which can have attributes that represent the individual, such as age, organizational attributes, etc. Each edge in E represents an interaction between individuals, such as following, sending a message, etc. which can also be attributed to indicate the properties of the interaction. The direction of an edge represents the initiator of an interaction. As the example shown in Fig. 1(a), nodes 1-7 represent users 1-7, respectively. Edge $(1, 4)$ represents an interaction initiated by user 1 to user 4, and the other edges have similar meanings.

2.1.2 Snapshot Model

In the discrete model, a dynamic social network existing between t_s and t_e is represented with a series of snapshots, denoted by $G = \{G_1, \cdots, G_i, \cdots, G_n\}$. The time interval $[t_s, t_e]$ is divided into n sub-intervals and the length of each sub-interval is $l = (t_e - t_s)/n$. For each snapshot $G_i = (V_i, E_i)$, V_i is the node set at time $t_s + i * t$ and E_i is the edge set including all edges within time interval $[t_s + i * t, t_s + (i + 1) * t]$. As the example shown in Fig. 1(c), the 40 time-units are divided into 4 intervals. In the first snapshot, all the nodes exist in the social network at time 10 and the timestamps of all the edges are within the time interval $[0, 10]$.

2.1.3 Continuous Model

In the continuous model, a dynamic social network existing between time t_s and t_e is represented by a directed graph with edges and nodes annotated with timestamps, denoted by $G = (V, E, T)$. V and E are the collections of nodes and edges over time $[t_s, t_e]$. Edges and nodes have the same meaning as in the static diagram and can also be attributed. $T : V, E \rightarrow t \in [t_0, t_0 + h]$ is a function that maps each node and edge to timestamps between time t_s and t_e. The timestamps of nodes represent the time when the node exists in the social network and the timestamps of edges indicate when the interaction took place.

Comparison Between Continuous and Snapshot Model By definition, the continuous model record in detail when each change in a social network occurs. The snapshot model simply records the current network at regular intervals which makes changes that occur during the time interval not recorded. As the example shown in Figure 1, the edges (1, 2), (1, 4) and (2, 3) are sequential. However, this sequential order is not reflected in the snapshot model, meaning that there is a loss of information. This information is crucial, for example node 1 cannot indirectly pass information to node 3 via node 2, because the interaction of nodes 2 and 3 occurs after the interaction of nodes 1 and 2. However, in the snapshot model, such transmission is possible.

The three models contain increasing amounts of evolving information, with the static model containing no evolving information and the continuous model containing all evolving information. But con-

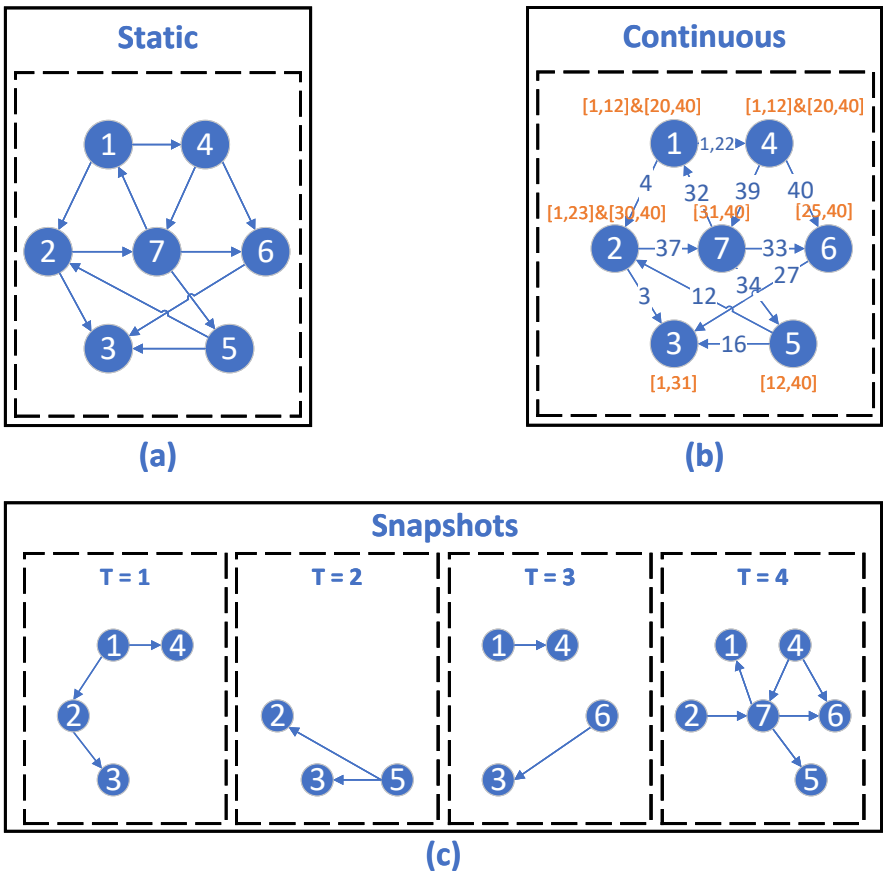

Figure 1: These are three different ways of representing a social network. There are seven users in this small simulated social network, which exists for a total of 40 time-units. Within these 40 time-units, users may leave, join the network or interact with other users.

sequently, the complexity of these three models increases in order and the problems defined on them become more difficult to deal with [16].

2.2 Combinatorial Optimisation

Combinatorial Optimisation (CO) is the study of mathematical methods to find optimal solutions to discrete events and is a classical and important branch of Operations Research. Combinatorial optimization is characterized by the fact that most problems are NP-complete, i.e. there is essentially no algorithm for the problem that gives the desired solution in a tolerable time (i.e. polynomial time) when the sum of the number of variables and constraints for all specific problems is large. The standard form of the combinatorial optimization problem is as follows [17]: **Find** x **to**

$$\begin{aligned} &minimize\ f(x) \\ &subject\ to\ g_i(x) \geq 0 \quad i = 1, ..., m \\ &\qquad\qquad h_j(x) = 0 \quad j = 1, ..., p \end{aligned} \tag{1}$$

where f, g_i and h_j are general functions of $x \in R^n$. Typical combinatorial optimization problems are traveling salesman problem (TSP) [18] and minimal spanning tree (MST) [19] etc.

3 Problem Definition

In this section, a general introduction to the influence maximization (IM) problem will be given first. **Then specific variation of the IM problem that will be addressed in this thesis will be illustrated in Subsection 3.3**.

3.1 General Influence Maximization Problem

The Influence Maximization (IM) Problem centers on the identification of a set of initial nodes within a network, commonly known as seed nodes, that, when activated, propagate influence through the network in a manner that maximizes the spread of this influence [1, 2]. The calculation of the influence of the node(s) is based on how influence diffuses among networks, i.e. diffusion model. The formal definition of the influence of a seed set (influence function) is given below.

Definition 3.1 (Diffusion Model) *A diffusion model, denoted as M, constitutes a formal mathematical framework employed to characterize the intricate process of the dissemination of information, innovation, influence, or any analogous form of contagion across an interconnected network comprising nodes. This model functions as a comprehensive mechanism aimed at describing the dynamics governing the manner in which entities within the network adopt novel states, contingent upon the behaviors exhibited by their interconnected counterparts.*

Definition 3.2 (Influence Function) *Within the context of a social network denoted as $G = (V, E)$, a designated diffusion model represented by M, and a selected seed set denoted as $S \subseteq V$, the influence function, denoted as $\sigma_{G,M}(S)$, is a well-defined mathematical function. This function operates within the realm of social networks and leverages the parameters of the information diffusion model M, the specific social network structure G, and the seed set S. Its primary purpose is to facilitate the calculation of the quantified extent of influence propagated by the members of the seed set S. This quantification pertains to the cumulative count of nodes that undergo activation, including the aggregated temporal duration of their activation, due to the influence emanating from the nodes within the seed set S.*

Based on the definition of influence function, the formal definition of influence maximization problem can be given below:

Definition 3.3 (Influence Maximization (IM) Problem) *[6] Given a social network G, a diffusion model M and the budget (a positive integer) k. The influence maximization problem is to find a k-size seed set $S \subseteq V$ that can maximize the influence function $\sigma_{G,M}(S)$, i.e. $\arg\max_S \sigma_{G,M}(S)$, where $|S|$ is the size of S.*

It is clear from the above definition that the choice of diffusion model can significantly impact the selection of optimal seed nodes. Different influence maximization problems are categorized according to the diffusion model selected. Thus the diffusion models are described in detail next.

3.2 Diffusion Models and Classification of IM Problems

3.2.1 Diffusion Models

There are a large number of existing diffusion models. In this thesis, diffusion models will be categorized in two different dimensions, for a total of four types of diffusion models (see Figure 2). In this subsection, these two dimensions will first be described and their practical implications in real life will be presented. Then typical diffusion models in each type will then be presented.

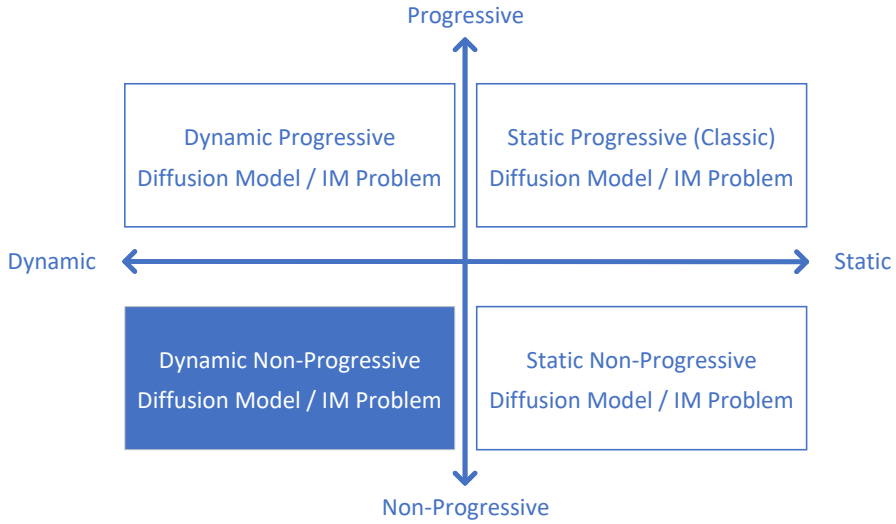

Figure 2: Classification of diffusion models and Influence Maximization (IM) problems. The category indicated in the bottom left corner is the one that this thesis focuses on.

The first dimension is progressive or non-progressive. In progressive diffusion model the nodes that are influenced cannot be deactivated (once a node is influenced it cannot be restored to an uninfluenced state). In contrast, in the non-progressive model, a node is influenced which, after a period of time, will return to the uninfluenced state.

The second dimension is the diffusion model is defined in static or dynamic graphs. In diffusion models defined on static graphs, the influence diffusion process is described independently of time. These models focus on understanding how information, influence, or behavior spreads through a network without considering temporal aspects. The relationships and connections between nodes in the graph remain fixed throughout the diffusion process. In contrast, diffusion models defined on dynamic graphs take into account time-dependent constraints on the diffusion process. These models recognize that the connections between nodes can change over time, and the diffusion process itself can be influenced by temporal dynamics. This allows for a more realistic representation of how information or influence spreads in real-world scenarios where network connections evolve over time.

The following is the description of typical diffusion models among the four categories.

Static Progressive Diffusion Models Static progressive diffusion models are the most classic models, and most existing diffusion models are static progressive diffusion models. Independent Cascade (IC) model, Linear Threshold (LT) model, and Triggering (TR) model are three most famous static progressive diffusion models.

The Independent Cascade Model was originally proposed by Jacob Goldenberg et al. [20] in their study of marketing models. The basic assumption of the model is that the success of an attempt by node u to activate its neighbor node v is an event with a probability $p_{u,v}$. The probability of a node in an inactive state being activated by a neighboring node that has just entered an active state is independent of the activity of neighbors that have previously attempted to activate the node. In addition, the model makes the assumption that any node u in the network has only one chance to try to activate its neighbor node v. Whether it succeeds or not, at subsequent moments u itself remains active but it no longer has influence, and this class of nodes is called uninfluential active nodes. The diffusion process stops when no more inactive nodes can be activated.

The influence probability $p_{u,v}$ can be assigned by a specific model or learning from real-world datasets. The most commonly used model is weighted cascade (WC) [6]. It sets $p_{u,v}$ to be $1/d_v^{in}$, where d_v^{in} is the in-degree of node v. Learning influence probability from data is first proposed by Saito et al. [21]. They used pectation Maximization (EM) algorithm to calculate probabilities of all edges to maximize the total likelihood of all actions. [22] and [23] etc. also proposed methods for learning probabilities from real-world datasets.

In 1978 Mark Granovetter investigated the potential resistance of users being influenced by those around them involved in a collective activity and thus also participating in that activity, proposing a threshold model of collective behaviour [11]. The linear threshold model assigns a threshold to each node v that represents the ease with which the node can be influenced. Nodes w adjacent to node v are influenced by node v with a non-negative weight $b_{v,w}$ and the sum of $b_{v,w}$ of all neighbors w of v is less than or equal to 1. For a node v in an inactive state, node v will only be activated if the sum of the influence of its active neighbor nodes is greater than or equal to its threshold θ_v, i.e. the decisions of individuals in the network depend on the decisions of all its neighbor nodes. And the active neighboring nodes of node v can participate in activating v multiple times. The diffusion process stops when no more inactive nodes can be activated. The weight $b_{v,w}$ is assigned similarly to influence probability in the IC model.

Triggering (TR) model is proposed by Kempe et al. [6] which combines IC and LT models. Each node v randomly selects a triggering set T_v from a given distribution on the set of incoming neighbor nodes. At timestep t, for each inactive node v, activate v if there is a point in T_v that is active at timestep $t-1$. The diffusion process stops when no more inactive nodes can be activated.

Static Non-Progressive Model The first non-progressive model is proposed by Kempe et al., who proposed the IC and LT model, in the same paper [6]. On the base of IC and LT model, each activate node will turn to inactivate state at each time step.

Another well-known and widely used static non-progressive model is the SIS model [24]. SIS stands for "Susceptible-Infectious-Susceptible, capturing the tran-

sitions individuals undergo between susceptible and infectious states in a population. In the SIS model, individuals within the population are categorized into two main groups, susceptible and infectious. Susceptible individuals have not been infected by the disease but can become infected through contact with infectious individuals. Infectious individuals are infected with the disease and have the potential to transmit it to susceptible individuals. The fundamental assumption of the model is that infectious individuals recover after a period of time and then become susceptible again, creating a cycle: susceptible individuals become infected, become infectious, recover, and then become susceptible once more. This cyclic process continues, simulating the disease's spread within the population. Key parameters of the SIS model often include infection rate, representing the probability of a susceptible individual becoming infected upon contact with an infectious individual, and recovery rate, representing the probability of an infectious individual recovering after a certain period.

The SIS model, originally designed for infectious disease spread, can also be adapted to describe the diffusion of influence or information in social networks. In this context, individuals oscillate between being receptive to influence (susceptible) and actively propagating it (infectious). The transmission dynamics, resembling infection and recovery, mirror the way ideas or behaviors can spread through a population, capturing the ebb and flow of influence as individuals adopt and transmit it within the network.

There are some other static non-progressive diffusion models to model such static non-progressive influence diffusion progress, such as [8]. However, these models have not been widely used since they were proposed, so they will not be described in detail. Compared with classic static progressive models, static non-progressive model gets less attention and therefore there are fewer models.

Dynamic Progressive Model Gayraud et al. [25] defined two variants of the IC and LT mode separately. These four diffusion models are defined based on the discreet dynamic social network model. The two variants of IC model are named tEIC (transient Evolving Independent Cascade) and p(persistent)EIC. In tEIC, a node u can only activate its neighbors immediately after the time instance in which it becomes active. In this case, the diffusion of items and the activation capacity of nodes in the network is transient. Formally, infectious nodes at time $t-1$ are given a single chance to activate their inactive neighbors at time t. The main difference with the IC model is that because the graph is dynamic, the neighboring nodes of a node are different at each timestep. In pEIC, infectious nodes at time $t-1$ are given a single chance to activate their inactive neighbors at time t and beyond. Thus the ability of a node to influence other nodes is persistent. The extensions to the LT model are also divided into two types (persistent and transient influence), named pELT and tELT respectively. In contrast to the LT model, a node accumulates the influence of active nodes it has encountered in the past in pELT. In tELT, the total incoming weight of live edges is computed only over the live edges present in a single graph instance. Liu et al. [26] proposed a variety of LT model. The model is defined based on the discreet model. All activated nodes of an unactivated node will try to activate it with a certain probability. As in the LT model, this node will be activated once a pre-set threshold has been exceeded. There are other works that have proposed dynamic diffusion models

[25, 27]. These models are all based on discrete graph models and extend from well-known static models, such as the LT and IC models.

Dynamic Non-Progressive Model To the best of my knowledge, there is no dynamic non-progressive model so far.

3.2.2 Classification of IM Problems

The definition of Influence Maximization (IM) problems and their corresponding solutions are intricately linked to the underlying diffusion model's dynamics and characteristics [3]. Thus, according to these four categories of diffusion model, IM problems can also be categorized into four corresponding categories. The IM problem defined on static progressive diffusion models are the classic IM problem which is most IM researches focus on. The IM problems defined on static non-progressive and dynamic progressive diffusion models are static non-progressive and dynamic progressive IM problems respectively. The IM problem defined on dynamic non-progressive diffusion models are dynamic non-progressive IM problem. Since there is no existing dynamic non-progressive diffusion models, there is no research on dynamic non-progressive IM problems so far.

3.3 Proposed Problem

Most diffusion models are progressive. In most scenarios like product purchases, the progressive assumption aligns well: buying a product often involves a commitment that isn't easily undone. But there are still some scenarios that can't be described by such progressive models [8]. Take, for instance, the scenario of a user adopting a mobile app. Over time, the initial attraction to the app might wane, leading to reduced usage. However, the user's interest could be revived when a friend informs her about a new and exciting feature added to the app. In response, she might decide to give the app another shot and resume using it. Additionally, the release of new app versions could entice the user to reconsider and potentially start using it again with a certain probability. Another example involves the cyclical nature of fashion trends. Choices that are currently in vogue can fall out of favor only to regain popularity in the future, mirroring the cyclic trends observed in social choices. Consider yet another scenario where users engage actively in a particular activity for a period before eventually disengaging. This can be seen in the adoption of features on content sharing platforms, such as the "like" or "favorite" button for posts, various filters for photo editing, or the act of "checking in" to a location or event.

Unlike the first dimension that both directions have practical applications. In the second dimension, the model defined on dynamic graphs is more in line with reality, since social networks are naturally dynamic. The diffusion models defined on static graphs are more to simplify the model because, as mentioned in Subsection 2.1, static graphs contain no evolving information but are the easiest to deal with. However, given the dynamic nature of social networks, it is the models defined on dynamic graphs that should be of more concern, even if this makes the problem more difficult to deal with.

The non-progressive property is naturally related to the dynamic property. For example, the time length that a node returns from activate state to inactive state is a temporal feature. A static progressive mode cannot capture such non-progressive property well as static models do not contain any evolving information at all. In contrast, the dynamic model, specially the dynamic model defined on the continuous model, can well remain the evolving information of a social network. So, a dynamic non-progressive model can capture such non-progressive property to the maximum extent possible.

Therefore, dynamic non-progressive diffusion models are of great practical importance. Correspondingly, the dynamic non-progressive influence maximization problem defined on such dynamic non-progressive models is research-worthy. Nevertheless, since there is no existing dynamic non-progressive diffusion model nor studies on dynamic non-progressive IM problem.

Hence, a dynamic non-progressive diffusion model, named DNPIC, which is extended from the IC model is first presented. The definition of DNPIC is as follows (A more specific description of DNPIC can refer to 1):

Definition 3.4 (DNPIC) *The DNPIC optimizes diffusion models for evolving information dynamics on a continuous dynamic graph. Initially activated nodes stay active, while others transition from inactive to active state upon successful activation, persisting for $t_{inactive}$ time units. Given a directed edge $e_{ij}(t)$, if node i is active, it strives to activate j. Upon success, j's inactive period resets at time $t + t_{inactive}$. The DNPIC's influence function computes the average active time per node (excluding initial activations). It encapsulates activation dynamics' impact via this equation:*

$$Influence(v) = \frac{1}{t_{total}} \sum_{t=0}^{t_{total}} \mathbb{I}(v \text{ active at } t),$$

where t_{total} is the total simulation time, and $\mathbb{I}$ is the indicator function.

Having given this dynamic non-progressive diffusion model, **the problem that this thesis wants to solve: the dynamic non-progressive IM problem** can also be given.

Definition 3.5 (Dynamic Non-Progressive IM Problem) *Given a dynamic social network $G = (V, E, T)$ modeled with the continuous model, a dynamic non-progressive diffusion model (referred to as DNPIC in this thesis) denoted by M, and a budget k (a positive integer), the Dynamic Non-Progressive Influence Maximization (DNPIM) problem is defined as follows:*

The objective of the DNPIM problem is to identify a k-size seed set $S \subseteq V$ that maximizes the average duration of time each node, excluding the initially activated nodes, remains in the active state. This is formulated as the function $\sigma_{DNP}(S)$, computed by:

$$\sigma_{DNP}(S) = \frac{\sum_{v \in V-S} \tau_v}{|V - S|}$$

Algorithm 1 DNPIC

Require: Dynamic graph with continuous model: $G = (V, E, T)$, initially activated node set: U, transition time from active to inactive state: t_{inactive}
  Initialize dictionary D to store activation times for each node
  Sort edges E by timestamp
  for $e_{ij}(t)$ in E **do**
    if $i \in U$ or $D[i][-1] \leq t \leq D[i][-1] + t_{\text{inactive}}$ **then**
      if random$(0, 1) \leq p_{ij}(t)$ **then**
        $D[t]$.append(t)
      end if
    end if
  end for
  Initialize *total_active_time* as 0
  for v_i in V **do**
    for $D[v_i][j]$ in $D[v_i]$ **do**
      if $D[v_i][j] + t_{\text{inactive}} \geq D[v_i][j+1]$ **then**
        $total\_active\_time+ = D[v_i][j+1] - D[v_i][j]$
      else
        $total\_active\_time+ = t_{\text{inactive}}$
      end if
    end for
  end for
  return *total_active_time*

Here, τ_v represents the cumulative sum of time durations during which node v remains in the active state. The problem aims to select an optimal seed set S that effectively propagates influence over time within the dynamic social network G using the dynamic non-progressive diffusion model M.

3.4 Properties of Proposed Problem

In this subsection, two important properties of the proposed problem, NP-hardness and being combination optimization problem will be discussed.

3.4.1 NP-hardness

The NP-hardness of the influence maximization problem has been proven by [6]. The dynamic non-progressive IM problem as a variant of the IM problem should also be NP-hard. Since this is not the focus of this thesis, a detailed proof is not be given here.

The NP-hard nature of the influence maximization problem has several important implications and considerations:

- **Computational Complexity:** An NP-hard problem implies that finding the exact optimal solution is generally difficult, especially for larger instances of the problem. For large networks, finding the exact optimal solution may require exponential time, which is infeasible in practice.

- **Algorithm Design:** Due to the problem's NP-hardness, researchers often focus on developing approximation algorithms and heuristic methods to find solutions that are close to optimal within a reasonable time. These algorithms do not guarantee finding the true optimal solution, but they can provide good solutions within an acceptable timeframe.

3.4.2 Combination Optimization Problem

Previous researches [1, 2] have proved that classic influence maximization problems is a combination optimization problem for the following reasons:

- Combinatorial Nature: The problem requires selecting a subset of nodes from the entire set of nodes. This selection process involves choosing which individuals to target for initial activation in order to maximize the spread of influence. The number of possible subsets grows combinatorially with the size of the node set V, leading to a combinatorial search space.

- Optimization Objective: The objective is to maximize the expected spread of influence in the network by selecting the most influential nodes. This objective function typically involves a complex and non-linear relationship between the selected nodes and the resulting spread of influence. Finding the optimal subset is not straightforward and often involves solving a maximization problem subject to certain constraints.

- Constraints: There are constraints on the size of the subset to be selected (i.e., $|S| = k$). Additionally, some versions of the problem might introduce constraints to ensure diversity in the selected subset or limit the budget available for initial activations.

- Search Strategy: The problem involves searching through the space of all possible subsets of nodes to find the one that maximizes the influence spread. This requires employing optimization algorithms that explore different combinations of nodes to determine the best subset.

- NP-Hardness: The influence maximization problem is known to be NP-hard, which means that as the size of the network grows, the time needed to find the optimal solution increases exponentially. This characteristic aligns with the behavior of many combination optimization problems.

Since dynamic non-progressive IM problem still fits properties of IM problem mentioned above, dynamic non-progressive IM problem is still a combination optimization problem.

4 Related Work

4.1 Graph Embedding

Graph embedding is a technique used in the field of machine learning and network analysis to transform graph data into continuous vector representations, often in a lower-dimensional space. The primary goal of graph embedding is to capture the

structural information of a graph in a way that facilitates downstream machine learning tasks, such as node classification, link prediction, and clustering. Traditional machine learning algorithms struggle to process graph data directly due to its irregular and discrete nature. Graph embedding addresses this challenge by projecting nodes into a continuous vector space, where algorithms designed for numerical data can be applied more effectively [28].

Depending on the type of graph being processed, graph embedding can be categorized into dynamic and static graph embedding.

4.1.1 Static graph embedding

The embedding results generated by static graph embedding are fixed and do not change over time. For the same node or edge, its embedding vector remains consistent across different time points since this method only considers the graph's structure without accounting for temporal changes. Techniques like DeepWalk [29], node2vec [30], GraphSAGE [31], and graph convolutional networks (GCN) [32] are commonly employed for static graph embedding. Since static graphs are not the main research object of this thesis, these methods are not described in detail. For a detailed survey, please refer [28].

4.1.2 Dynamic graph embedding

Dynamic graph embedding can indeed be classified into two main categories, each addressing different aspects of handling evolving graph data.

Snapshot-Based Dynamic Graph Embedding: In this type of approach, the dynamic graph is treated as a series of snapshots, each representing the graph at a specific time point. The focus is on generating embeddings for individual snapshots, and the embeddings can be quickly updated as the graph evolves. However, the final embeddings do not explicitly encode the temporal evolution information. This method is suitable for scenarios where real-time updates are required, and the primary interest lies in the current state of the graph rather than the temporal patterns of change. EvolveGCN [33] is a representative of this type of approach. Since this type of approach is not involved in this thesis, it is not presented further.

Temporal Dynamic Graph Embedding: This type of approaches aims to capture the temporal evolution of the graph explicitly in the embedding space. It involves learning embeddings that not only represent the graph's structure but also incorporate information about how nodes and edges evolve over time. It typically requires modeling time as a parameter and optimizing embeddings to minimize the difference between node representations across different time points. Temporal dynamic graph embedding is well-suited for analyzing and understanding the changing dynamics and trends in the graph over time. Some methods use random wandering, which differs from random wandering in static graphs by the addition of temporal constraints[34, 35]. More methods use RNNs to update node representations when changes on edge happen.[36, 37].

The best performing and widely used method in this category is the TGNs proposed by Rossi et al. [38]. They designed a novel combination of memory

modules and graph-based operators which makes TGNs can significantly outperform previous approaches while being more computationally efficient.

4.2 Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning concerned with how agents learn to make sequential decisions through interactions with an environment [39]. In RL, an agent learns to take actions in an environment to maximize a cumulative reward signal.

Markov Decision Process A fundamental framework in RL is the Markov Decision Process (MDP) [40], represented by the tuple (S, A, P, R), where:

- S: Set of states
- A: Set of actions
- P: State transition probabilities
- R: Reward function

At each time step t, the agent is in a state $s_t \in S$, selects an action $a_t \in A$, and receives a reward $r_{t+1} \in \mathbb{R}$. The agent then transitions to a new state s_{t+1} according to the probabilities $P(s_{t+1}|s_t, a_t)$.

Policy and Value Functions A policy π is a strategy that defines the agent's actions in different states. It is denoted as $\pi(a|s)$, representing the probability of taking action a in state s. The goal is to find an optimal policy π^* that maximizes the expected cumulative reward. Figure 3 illustrates the action-reward feedback loop of a generic RL model.

The value function $V^\pi(s)$ represents the expected cumulative reward starting from state s and following policy π: $V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s\right]$, where $\gamma \in [0, 1]$ is the discount factor.

The action-value function $Q^\pi(s, a)$ represents the expected cumulative reward starting from state s, taking action a, and then following policy π: $Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a\right]$.

Bellman Equations In reinforcement learning, the Bellman equations [42] play a crucial role in representing the relationships between current and future values of states and actions. These equations are fundamental for solving Markov Decision Processes (MDPs) and finding optimal policies.

The value function $V^\pi(s)$ represents the expected cumulative reward an agent can achieve starting from state s and following policy π. The Bellman equation for the value function is given by:

$$V^\pi(s) = \sum_a \pi(a|s) \left(R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^\pi(s') \right),$$

where:

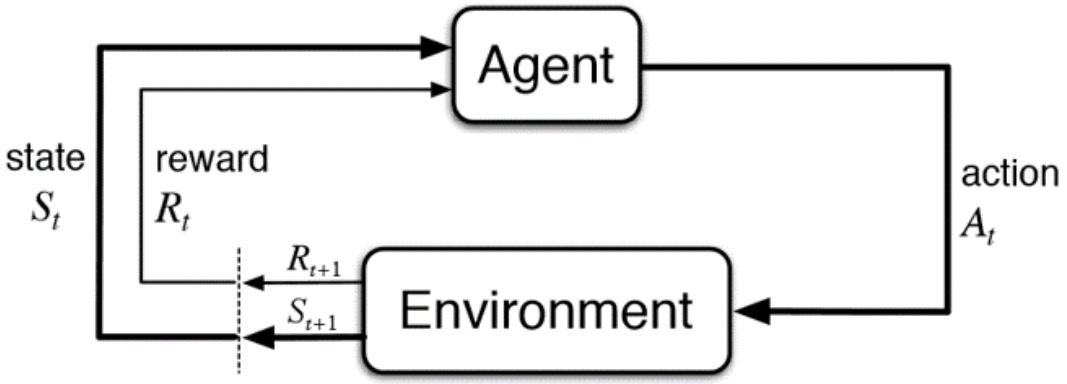

Figure 3: Action-reward feedback loop of a generic RL model[41]

- $V^\pi(s)$ is the value of state s under policy π.
- $\pi(a|s)$ is the probability of taking action a in state s according to policy π.
- $R(s, a)$ is the immediate reward obtained when transitioning from state s to a new state after taking action a.
- γ is the discount factor that trades off the importance of immediate and future rewards.
- $P(s'|s, a)$ is the probability of transitioning to state s' given that action a is taken in state s.
- $V^\pi(s')$ is the value of the next state s' under policy π.

The action-value function $Q^\pi(s, a)$ represents the expected cumulative reward an agent can achieve starting from state s, taking action a, and following policy π. The Bellman equation for the action-value function is given by:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) \sum_{a'} \pi(a'|s') Q^\pi(s', a'),$$

where the variables have the same meanings as in the value function Bellman equation.

The Bellman equations provide a recursive way to compute the value of states and actions based on their immediate rewards and the expected future rewards. Solving these equations helps in finding the optimal policy that maximizes the expected cumulative reward over time.

Reinforcement Learning Algorithms There are various algorithms in RL that aim to learn optimal policies and value functions. Some common ones include:

- Q-Learning [43]: A model-free algorithm that learns action-value functions and uses an exploration-exploitation strategy to find optimal policies.
- Policy Gradient Methods [44]: These algorithms directly optimize policy parameters to find policies that maximize expected rewards.

- Deep Q Networks (DQN) [45]: Q-Learning is combined with deep neural networks to handle complex state spaces and achieve state-of-the-art results in many domains.
- Proximal Policy Optimization (PPO) [46]: A family of policy optimization algorithms that ensure stable policy updates.
- Actor-Critic Methods [47]: These algorithms combine policy-based and value-based approaches to improve learning efficiency.

These algorithms are used in various applications like robotics, game playing, and autonomous systems.

4.2.1 Deep Q Networks (DQN)

Deep Q Networks (DQN) [45] is a milestone algorithm that combines Q-learning with deep neural networks to approximate the optimal action-value function $Q^*(s, a)$ (see Figure 4). DQN addresses the limitations of tabular Q-learning in handling large state spaces. The DQN algorithm uses an experience replay buffer, where transitions (s, a, r, s') are stored and randomly sampled for learning. The target Q-network, denoted as Q^-, is used to stabilize learning by decoupling target updates from online updates.

The DQN objective minimizes the mean squared Bellman error:

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q^-(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right],$$

where θ are the network parameters, (s, a, r, s') are the transition elements, γ is the discount factor, and θ^- are the target network parameters.

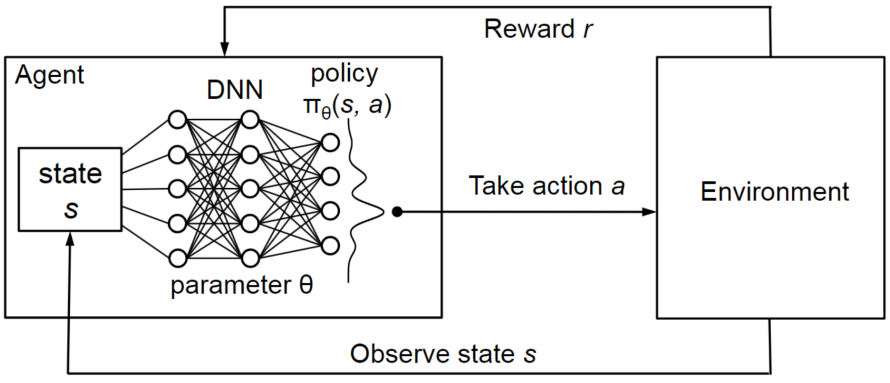

Figure 4: Deep Q Networks (DQN)

Dueling DQN Dueling DQN [48] is an extension of DQN that aims to improve the efficiency of value function estimation. It introduces a separate network architecture to estimate the state value $V(s)$ and action advantages $A(s, a)$ independently (see Figure 5 for a compassion between DQN and Dueling DQN). This separation helps the network learn the value of each state without being influenced

by the specific action taken. The final Q-value is then obtained by combining the state value and action advantages:

$$Q(s, a) = V(s) + A(s, a) \tag{2}$$

This architecture allows the network to focus on learning the general value of states and the impact of actions on those values, leading to more stable value estimates.

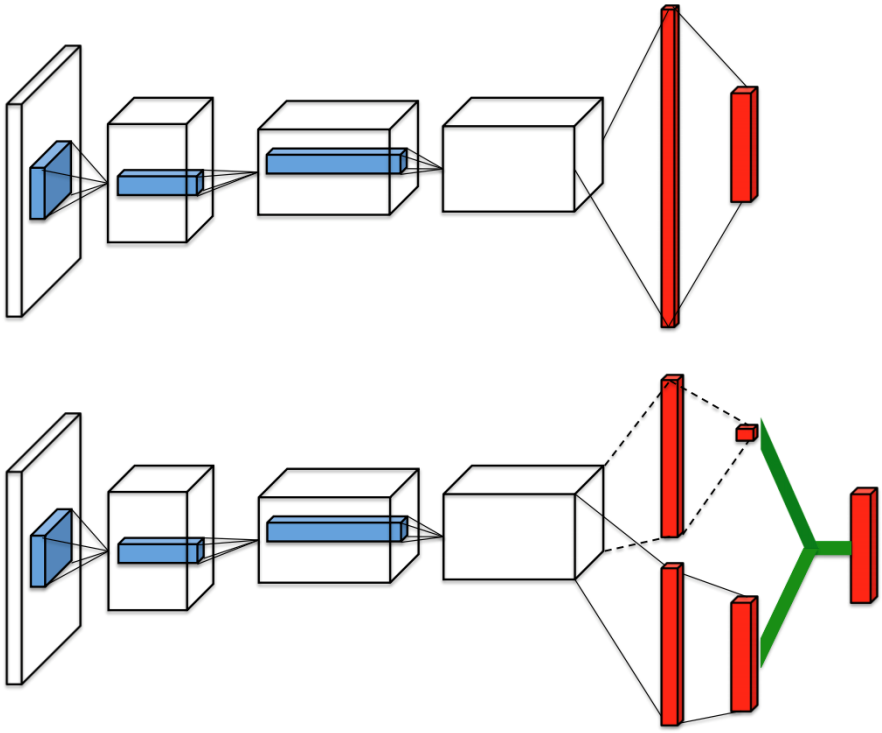

Figure 5: A popular single stream Q-network (top) and the dueling Q-network (bottom) [48]

Double DQN Double Deep Q Networks (Double DQN) [49] is an advancement of the DQN algorithm that aims to address the overestimation bias present in Q-learning and DQN. Double DQN uses two separate networks to mitigate this bias, namely the main network and the target network. The action with the highest Q-value is selected using the main network, but the Q-value itself is evaluated using the target network, resulting in the equation:

$$Q(s, a) = R(s, a) + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-) \tag{3}$$

where θ are the parameters of the main network, θ^- are the parameters of the target network, $R(s, a)$ is the immediate reward, γ is the discount factor, and $\arg\max_{a'} Q(s', a'; \theta)$ is the action that maximizes the Q-value in the target state s'. This separation of action selection and Q-value evaluation helps to reduce the overestimation of Q-values, leading to more accurate learning and improved performance.

4.2.2 CO Meets RL

Traditional algorithms for CO problems include exact, approximate and heuristic approaches. Combinatorial optimization problems are not feasible to solve using brute force due to the excessive number of combinations that exist. These conventional methods can speed up the calculation to some extent. However, according to the survey by Bengio et al. [50] and Mazyavkina et al. [51], traditional algorithms still suffer from relying on manual heuristics to make decisions, being too computationally expensive and mathematically ill-defined.

Machine learning can make such decisions in a more principled and optimized way. And among machine learning methods, reinforcement learning usually leads to the best performance and highest efficiency for following reasons:

- Rapid and Optimal Learning: Reinforcement learning algorithms can learn rapidly and optimize their actions based on rewards or penalties received from the environment [52]. This allows the agent to quickly adapt and improve its performance over time.

- No Expert Knowledge Required: Traditional methods for solving combinatorial optimization problems often rely on hand-crafted heuristics or expert knowledge. In contrast, reinforcement learning obviates the need for expert knowledge or pre-solved instances [51]. The agent learns to make decisions and find solutions through trial and error, without relying on predefined rules or heuristics.

- Scalability and Computation Efficiency: Deep reinforcement learning (DRL) approaches have shown advantages over traditional methods in terms of scalability and computation efficiency [53]. DRL algorithms can handle large-scale combinatorial optimization problems and make efficient use of computational resources.

- Exploration and Exploitation: Reinforcement learning algorithms balance exploration and exploitation. They explore different actions and strategies to discover optimal solutions while also exploiting the knowledge gained from previous experiences [51]. This allows the agent to search for better solutions and avoid getting stuck in suboptimal solutions.

For these reasons, reinforcement learning has been increasingly used to solve CO problems recently.

4.3 Non-machine Learning Influence Maximization Methods

As introduced in Subsection 3.4.2, IM problem is a combination optimization problem. According to the properties of CO problem and its solutions (introduced in Subsection 4.2.2), non-machine learning methods have some unavoidable drawbacks, especially in the face of today's increasingly large scale social networks. Moreover, non-machine learning approaches are not the focus of this thesis, so researches focus on using non-machine learning methods to solve IM problems are not discussed here. If interested, a detailed discussion can be found in the Appendix A.

4.4 Machine Learning Influence Maximization Methods

Different from many traditional non-machine learning influence maximization methods are often designed or tailored specifically to certain diffusion models or assumptions. These methods might rely on simplifications or assumptions that are not applicable across different diffusion models, limiting their generalizability. Machine learning-based influence maximization methods are not tied to specific assumptions about the diffusion process, and they can learn from data how influence propagates through networks in different scenarios. Therefore the categorization of machine learning based IM methods will no longer be based on the type of IM problem targeted, but on the method of machine learning used.

The first thing to face in applying a machine learning to solving IM problems is how to feed graphs into machine learning models because the original graph data is very sparse and large in size. Graph embedding (introduced in Subsection 4.1) provides a good solution. Almost all machine learning based approaches first transform the graph into low-dimensional vectors via (improved) graph embedding methods in order to allow the graph to be fed into the machine learning model. Based on the models of machine learning used, machine learning based approaches can be classified into three categories: supervised, unsupervised, and reinforcement learning.

4.4.1 Unsupervised Learning Methods

Due to the complexity of IM problem, labeled datasets are difficult to obtain which makes unsupervised machine learning to be the dominant approach to solve the IM problem. Based on the output of the machine learning models, unsupervised approaches can be divided into two categories. The first category of methods outputs the optimal seed set directly. The other category of methods only outputs the node embeddings preserving IM-related features and the seed set is generated based on these node embeddings.

The FastCover proposed by Ni et al. [54] is a typical first-category approach. They reduce the IM problem to a budget-constrained d-hop dominating set problem (kdDSP). They designed a multi-layer GNN named graph reversed attention network (GRAT) that captures the diffusion process among neighbors. The key idea of the method is train the model to learn a scoring mechanism for scoring the influence of each node in the range of [0, 1] and selecting the top-k nodes can form the optimal seed set. The parameters of the GRAT are trained by optimizing a differentiable loss function in an unsupervised manner. The authors claim that FastCover can find a better quality solution while achieving a speed increase of more than 1000 times.

The second category of methods focuses more on designing a graph embedding model that preserves IM-related properties. Zhang et al. [55] designed a GCN with adjustable number of layers for different sizes of networks to balance scalability and performance. A self-labeling mechanism was designed to train the network with a more nodal degree. After obtaining these node embeddings, they devised a distance(between difference node embedding vectors)-based heuristic to generate the seed set. Similarly, Li et al. [56] designed a random walk based graph embedding method for heterogeneous graphs. They find a set for each node that contains all relevant nodes based on the cosine similarity of the two node

embeddings and a pre-set similarity threshold. The top-k nodes that appear most frequently in these node sets are selected to generate the optimal set of seeds.

While unsupervised learning methods can solve the problem of lack of labelled data, it is difficult to design objectives or guidelines to guide the training of the model in a comprehensive and reasonable manner. Methods for processing graph embedding results are sometimes designed for part of the diffusion model.

4.4.2 Supervised Learning Methods

In order to address the problem of lack of datasets, which hinders the application of supervised machine learning methods to IM problems, many studies have also attempted to propose some efficient labeling methods. Kumar et al. [57] use the influence of node under SIR and IC model as the label. They transfer the IM problem to a pseudo-deep learning regression problem. They use a big real-world dataset to train a Graph Neural Network (GNN) based regressor. The input of the model is centrality-based feature vectors which are composed of node embedding (generated using an existing graph embedding method sturc2vec [58]) and features of each node. The output of the model is the influence of each node. Then, transfer the trained model is used to predict the influence of nodes on the target network. With the influence of nodes in the new dataset, they simply choose the top-k nodes to form the seed set. Soon after, they proposed another supervised method [59] where they still convert the IM problem to a classical regression task. The difference is that they use graph-based long short-term memory (GLSTM) to solve the problem. The experimental results show that the change in the use of the model significantly improves the performance.

Overall, the approach to data labeling opens up the possibility of applying a wider variety of machine-learning models to IM problems. Compared to traditional methods, machine learning-based approaches can further reduce time complexity and improve reliability. However, this approach requires a high degree of generality in the datasets used for training. As there is little research on the subject, it is still worth exploring whether models trained on a small variety of datasets can be applied to all types of social networks.

4.4.3 Reinforcement Learning Methods

Li et al. [60] first used RL to solve the classic IM problem and proposed the framework named DISCO that composes GNN and DQN. For the IM problem, They use an existing graph embedding method named structure2vec [61] as a basis for graph embedding, and the actual formula used to update the update vectors iteratively is shown as follows:

$$x_v^{(i)} := \mathrm{ReLU}(\alpha_1 \sum_{u \in N(v)} x_u^{(i-1)} + \alpha_2 \sum_{u \in N(v)} \mathrm{ReLU}(\alpha_3 w(v,u)) + \alpha_4 a_v) \tag{4}$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are parameters, ReLU is the Rectified Linear Unit of a neural network, $x_v^{(i)}$ is the node representation of node v in the $i-th$ iteration, $N(v)$ refers to the neighboring nodes of node v, a_v refers to whether the node is selected into the seed set (1 for selected, 0 for unselected). At the start of the iteration, all node representations are initialized to q-dimensional zero vector. After a maximum of about 4 rounds, all node representations will reach their final state.

The modified structure2vec achieves end-to-end learning by combining with DQN [45]. In the RL components, the Q function can predict the marginal influence gain of a node based on the generalized node representation is defined as follows:

$$Q(v, S, \Theta) = \beta_1^T \, \mathrm{ReLU}([\beta_2 \sum_{u \in V} \mu_u^{(I)}, \beta_3 \mu_v^{(I)}]) \tag{5}$$

where $[,]$ is the concatenation operator, $\beta_1, \beta_2, \beta_3$ are parameters, and Θ refers to all the parameters $\alpha_1, ..., \alpha_4, \beta_1, ..., \beta_3$. Using the neural network to train a Q function to predict the marginal influence gain of a node can avoid the high complexity of using traditional methods. In order to improve the efficiency of training, they propose to first use randomly sampling some subgraphs from the graph and train using these subgraphs. They also provide some trained models and claim that these models are applicable to all types of graphs. Shortly after, they proposed the PIANO framework [12], which is an evolved version of DISCO. Its general idea is the same as DISCO, but it refines the algorithm design and theoretical analysis and adds a large number of comparative experiments.

Based on DISCO, Wang et al. [62] proposed the IMGER. They use graph attention networks for graph embedding. Compared to DISCO, additional weights can be learned for the edges, allowing for more realistic scenarios to be applied. In addition, they use Double DQN, which is otherwise similar to DISCO.

To improve the efficiency of RL-based IM algorithms, some approaches integrate network pruning techniques to focus computational resources on more important nodes or sub-networks. Manchanda et al. proposed the GCOMB [63] framework to address the IM problem by introducing a node quality assessment mechanism. GCOMB trains the model using both network embedding and Q-learning. In GCOMB, a graphical neural network named GraphSAGE [31] is used for pruning poor nodes and learning embeddings of good nodes in a supervised manner. Q-learning considers only needs to consider the good nodes left after pruning to generate the seed set.

Chen et al. [64] claim that both GCOMB and PIANO are graph-specific as they are trained on subgraphs, and GCOMB is based on supervised learning which introduced large extra computational overhead and efforts of hand-crafting the learning pipeline, so they provided a new method named ToupleGDD which is a end-to-end reinforcement learning framework. They designed a three coupled graph neural networks (GNNs) for network embedding and double deep Q-networks (DQNs) for parameters learning. Besides, their model is trained on several small randomly generated graphs with a small budget and tested on completely different networks under various large budgets which make it not graph-specific. According to the experiment, ToupleGDD performed similarly to IMM [65] but had better performance in running time and shows strong generalization ability.

Though these reinforcement learning based methods partly address the shortcomings of supervised and unsupervised learning based methods, **one significant disadvantage they all have is using static graph embedding methods. This means they can't (hardly) handle dynamic influence maximization problems as the evolving information is ignored in the whole framework.**

5 Methodology

From the last section, it can be seen that non-machine learning methods for influence maximization often lack adaptability and scalability due to their reliance on rigid assumptions, limiting their effectiveness in capturing real-world complexities and hindering their applicability to diverse diffusion models and network structures. Compared to traditional methods, supervised and unsupervised machine learning-based methods can further reduce time complexity and improve reliability. However, supervised learning methods require labelled datasets and a high degree of generality in the datasets used for training. The unsupervised learning methods may solve the problem of lacking labelled datasets, but they also suffer from the difficulty of designing objectives or guidelines to guide the training of the model in a comprehensive and reasonable manner.

Reinforcement learning offers a promising solution to the aforementioned drawbacks by enabling adaptability and scalability in influence maximization. Unlike non-machine learning methods, reinforcement learning can learn optimal strategies through interactions with the environment, effectively adapting to diverse diffusion models and network structures. Reinforcement learning agents can adapt their strategies based on real-time feedback, mitigating the need for rigid assumptions. Additionally, reinforcement learning methods can handle unlabelled data by learning from rewards obtained through exploration, alleviating the labeling requirement of supervised learning. Moreover, reinforcement learning provides a framework for defining objectives and guidelines for training in a flexible and comprehensive manner, addressing the challenges faced by unsupervised learning.

It is clear that reinforcement learning is by far the best way to solve IM problems. However, existing reinforcement learning based methods suffer from the inability to deal with dynamic IM problems due to the static graph embedding methods used. **The goal of the thesis is to solve this drawback by combining dynamic graph embedding methods with reinforcement learning algorithms and propose a method for dynamic non-progressive IM problem where there is no effective solution yet.** To achieve the goal, a novel framework, named DNPRL, is proposed.

5.1 Framework

Give the large state spaces caused by the large number of nodes in the social network, the reinforcement learning algorithm used in this framework (see Figure 6) is DQN [45]. For better performance the extension made by Dueling DQN [48] and Double DQN [49] (Dueling Double DQN) is included.

The definition of RL components are introduced in Subsection 5.3. The structure of the main Q-network and the target Q-network of double DQN are introduced in Subsection 5.4.

The dynamic social network is first transferred to node representations using the dynamic graph embedding method TGNs which is introduced in Subsection 5.2). For one episode (one episode represents a complete sequence of node additions starting from an empty set set until termination and each addition is named a step), the agent interacts with the environment for a predetermined number of time steps, representing the budget for the seed set. During each step, the

agent selects an action based on Eepsilon-greedy strategy which is introduced in Subsection 5.5. After one episode, the reward of all actions in this episode is calculated according to Equation 7. These experiences, composed of the current state, selected action, reward, and next state, are stored in the replay memory for later use. After each episode, the main Q-network is updated several times by the method introduced in Subsection 5.6 and the exploration rate is updated to gradually shift the agent's focus from exploration to exploitation as training progresses. After a fixed number of episodes, the main Q-network's weights are updated to the target Q-network to stabilize the learning process.

Upon achieving convergence and stability, the refined model becomes amenable to the purpose of optimal seed set selection. Importantly, the singular training of the model effectively obviates the necessity for repetitive retraining to accommodate varying seed set budgets. By virtue of this efficiency, the model's efficacy extends across the spectrum of seed set budgets, with only the selection of a solitary seed set budget (k) sufficing to yield a model that robustly caters to diverse budget constraints. This streamlined approach markedly streamlines the deployment process, affording seamless adaptability to various scenarios without incurring the computational overhead entailed by recurrent retraining endeavors.

The framework is also described specificly by Algorithm 2.

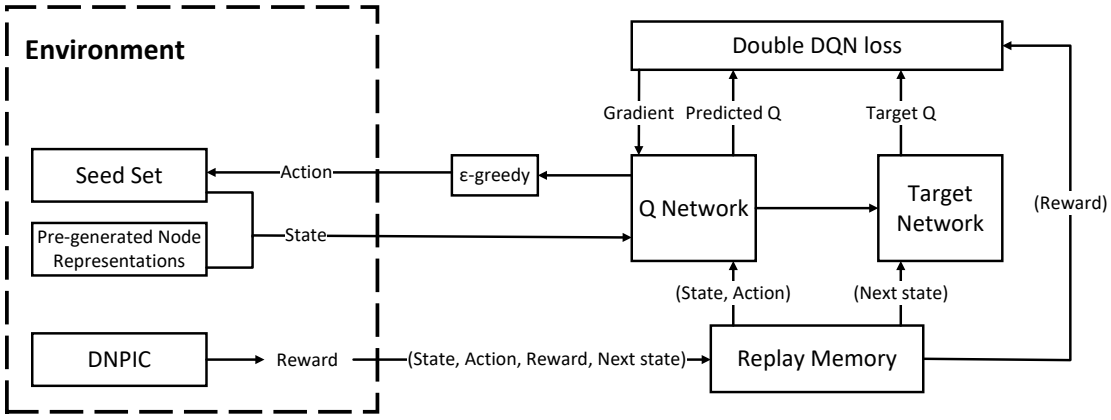

Figure 6: General framework of DNPRL

5.2 Graph Embedding

The graph embedding method we selected is TGNs [38]. The framework of TGNs is shown in Figure 7. TGNs also use the encoder-decoder model. The encoder maps dynamic graph to node embeddings and the decoder uses node embeddings as input to do a task-specific prediction such as link prediction. TGNs models continuous-time dynamic graphs as a sequence of time-stamped events. The addition and deletion of both nodes and edges can be seen as an event.

The core modules of TGNs are as follows:

- **Memory** The memory of the model at time t consists of a vector $s_i(t)$ for each node i the model has seen so far. $s_i(t)$ is a vector and and is initialized to a zero vector. The memory of a node is updated after an event (e.g. interaction with another node or node-wise change), and its purpose is to

Algorithm 2 Training process

Require: Number of episodes M, budget of seed set k, discount factor γ, learning rate α, target Q-network update frequency C, initial exploration rate $\epsilon_{\text{initial}}$, minimum exploration rate $\epsilon_{\min}$, exploration decay rate ϵ_{decay}, size of replay buffer B, number of updates to main Q-network T, $t_{inactive}$

Initialize replay memory D with size B
Initialize main Q-network Q with random weights
Initialize target Q-network $\hat{Q}$ with same weights as Q
Initialize exploration rate $\epsilon_{\text{initial}}$
Pre-generate node representations using TGNs
for episode = 1 to M **do**
  Reset environment state s
  for t = 1 to k **do**
    Calculate $Q(s, a; \theta)$ for each action a
    Set $Q(s, a; \theta)$ for each action a that selects nodes already in the seed set
    With probability ϵ select a random valid action a_t, otherwise select $a_t = \arg\max_a Q(s, a; \theta)$
    Execute action a_t, observe reward r_t and next state s_{t+1}
    Set $s \leftarrow s_{t+1}$
  end for
  Calculate reward r_t of each action a_t according to Equation 7
  for t = 1 to k **do**
    Store transition (s, a_t, r_t, s_{t+1}) in replay memory D
  end for
  for t = 1 to T **do**
    Sample a random mini-batch of transitions (s_i, a_i, r_i, s_{i+1}) from D
    Compute target values: $y_i = r_i + \gamma \hat{Q}(s_{i+1}, \arg\max_a Q(s_{i+1}, a; \theta); \theta^-)$
    Update main Q-network weights using gradient descent according to Equation 3
  end for
end for
Update exploration rate: $\epsilon \leftarrow \max(\epsilon_{\min}, \epsilon - \epsilon_{\text{decay}})$
Every C steps, update target Q-network weights: $\theta^- \leftarrow \theta$

represent the node's history in a compressed format. Due to this specific module, TGNs have the capability to memorize long term dependencies for each node in the graph. When a new node is encountered, its memory is initialized as the zero vector, and it is then updated for each event involving the node.

- **Message Function and Aggregator** For each event involving node i, a message is computed to update i's memory. Experiments show that the following method of generating messages is optimal. For a node-wise event $\mathrm{v}_i(t)$, the message $\mathrm{m}_i(t) = \mathrm{s}_i(t^-)||t||\mathrm{v}_i(t)$ is generated. For a edge-wise event $\mathrm{e}_{ij}(t)$, two messages $\mathrm{m}_i(t) = \mathrm{s}_i(t^-)||\mathrm{s}_j(t^-)||\Delta t||\mathrm{e}_{ij}(t)$ and $\mathrm{m}_j(t) = \mathrm{s}_j(t^-)||\mathrm{s}_i(t^-)||\Delta t||\mathrm{e}_{ij}(t)$ are generated. Here, $\mathrm{s}_i(t^-)$ is the memory of node i just before time t.

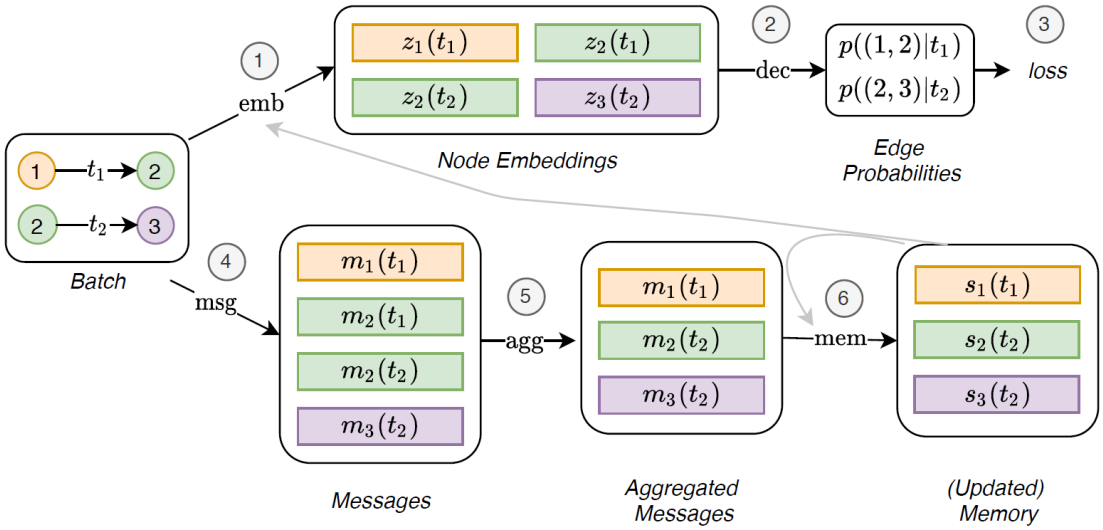

Figure 7: Framework of TGNs[38]

In the same batch, it is high likely that there are multiple events involving the same node i. For efficiency reasons, a mechanism is designed to aggregate these messages. Experiments show that retaining only the most recent messages ($\overline{\mathrm{m}}_i(t)$) is the best strategy.

- **Memory Updater** The memory of a node is updated upon each event involving the node itself which can be represent with $mem(\overline{\mathrm{m}}_i(t), \mathrm{s}_i(t^-))$. mem a learnable memory update function, e.g. a recurrent neural network such as LSTM or GRU.

- **Embedding** The embedding module is used to generate the temporal embedding $\mathrm{z}_i(t)$ of node i at any time t. The main goal of the embedding module is to avoid the so-called memory staleness problem and better aggregation of information of neighbors. The embedding module can be implemented with

$$\mathrm{z}_i(t) = \sum_{j \in n_i^k([0,t])} h(\mathrm{s}_i(t), \mathrm{s}_j(t), \mathrm{e}_{ij}, \mathrm{v}_i(t), \mathrm{v}_j(t)) \tag{6}$$

where h is a learnable function. Experiments show that temporal graph attention network can led to the best performance. Specific information about temporal graph attention network can be found in the original paper.

The main reasons that TGNs is selected are as follows:

- In this framework, the node representations need to be able to contain evolving information of nodes. So, only the temporal dynamic graph embedding methods are applicable instead of snapshot-based dynamic graph embedding methods. And, TGNs is a typical temporal dynamic graph embedding method.

- Among temporal dynamic graph embedding methods, TGNs has the best performance and is the most widely used one.

- It can support multiple tasks to train the model, especially dynamic link prediction. The main goal of dynamic link prediction is to forecast which new connections are likely to form in the network as it evolves. The performance on dynamic link prediction task can well reflect how well the node representations can response the evolving information of nodes. So, this is strongly correlated with the spread of influence. The node representations generated by the TGNs well contain the attributes related to influence maximization, so the generated node representations can be directly used to calculate the marginal gain of adding a node into the seed set without the need to include them in the training process of reinforcement learning.

5.3 Definition of RL Components

The definition of RL components are as follows:

- **State (S_t):** The state at time step t, denoted as S_t, is formed by connecting the node representations generated by Temporal Graph Networks (TGNs) with a one-hot vector X that represents the current seed set. The length of the one-hot vector is equal to the total number of nodes in the graph. Each element of the one-hot vector X is a binary indicator that signifies whether the corresponding node is part of the current seed set. Specifically, a value of 1 indicates that the node is in the seed set, while a value of 0 indicates that the node is not in the seed set. For example, $X_i = 0$ denotes that node i is not included in the seed set.

- **Action (A_t):** An action A_t taken at time step t corresponds to the addition of a node v to the current seed set S_t. This action is chosen from the set of nodes that are not currently present in the seed set, i.e., $v \notin S_t$.

- **Transition:** A transition occurs when a node i is selected to be added to the seed set. This selection causes the corresponding element X_i of the one-hot vector to change from 0 to 1, reflecting the inclusion of node i into the seed set.

- **Reward (R_t):** The reward of an action is calculated after one episode. After one episode, a set of actions $\{A_1, ..., A_i, ..., A_k\}$ that select nodes $\{v_1, ..., v_i, ..., v_k\}$ correspondingly will generate. The reward R_t for an action A_i is defined as

$$\mathcal{A}_i = \sigma_{DNP}(\{v_1, ..., v_i, ..., v_k\}) \frac{single\_inf(i)}{\sum_{j=1}^{k} single\_inf(j)} \tag{7}$$

  where $single\_inf(i)$ is the influence of node i, calculated by

$$single\_inf(i) = \sigma_{DNP}(i) \tag{8}$$

Different from the normal way to define reward $R_t = \sigma_{DNP}(S \cup \{i\}) - \sigma_{DNP}(S)$. The drawback of this normal reward way is that the reward for the first action in one episode will be far greater than all following actions. However, this is not because the node selected by the first action have greater influence than other nodes, but simply because it was the first node to be selected and no node in

the entire network had been influenced before that. In other words, the influence of the first selected node is overestimated.

To avoid this, the reward of an action is defined after a episode. In such way, the reward is calculated based on all the actions in one episode and assigned to each action according to their influence. This will reward each action more in line with the influence of the chosen node. A example is given in Table 3. See this example for a more detailed description.

5.4 Q-network

Since the Dueling Double DQN is used, the Q-network (See Figure 8) is composed of three networks, main network, value network and adventure network.

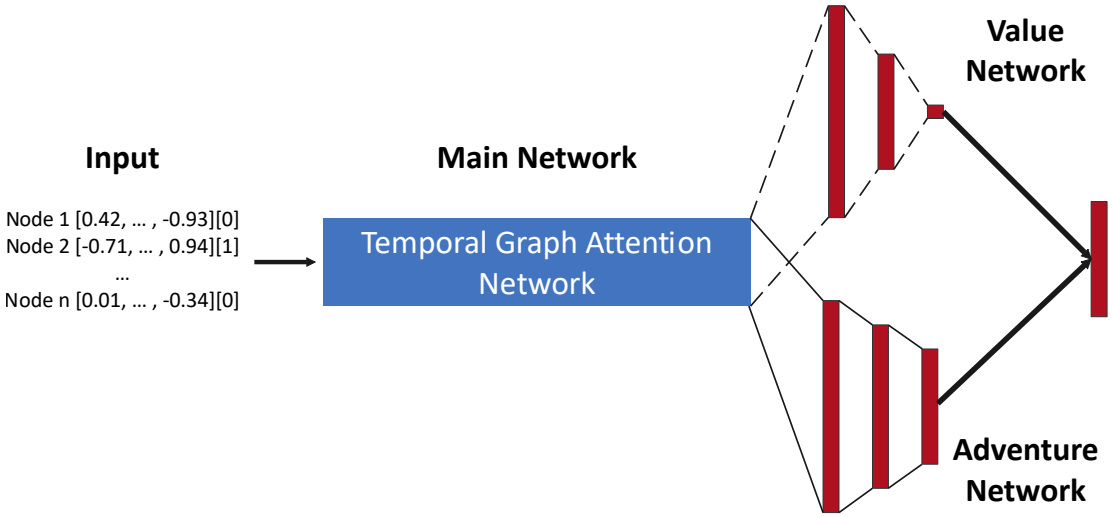

Figure 8: Structure of the Q network

The main network is a multi layer temporal graph attention network proposed by TGNsby Rossi et al. [38] which can aggregate information of neighboring nodes. Specifically,

$$
\begin{aligned}
&h_i^{(l)}(t) = \mathrm{MLP}^{(l)}(h_i^{(l-1)}(t) \| \tilde{h}_i^{(l)}(t)) \\
&\tilde{h}_i^{(l)}(t) = \mathrm{MultiHeadAttention}^{(l)}(q^{(l)}(t), K^{(l)}(t), V^{(l)}(t)) \\
&q^{(l)}(t) = h_i^{(l-1)}(t) \| \phi(0) \\
&K^{(l)}(t) = V^{(l)}(t) = C^{(l)}(t) \\
&C^{(l)}(t) = [h_1^{(l-1)}(t) \| e_{i1}(t_1) \| \phi(t - t_1), \ldots, h_N^{(l-1)}(t) \| e_{iN}(t_N) \| \phi(t - t_N)]
\end{aligned}
\tag{9}
$$

where $\phi(\cdot)$ represents a generic time encoding [66], $\|$ is the concatenation operator.

After passing through the main network, each node is represented with a c-dimensional vector, where c is adjustable. The concatenation of these vectors yields a one-dimensional vector and the length of the vector is $c \times |V|$, where $|V|$ is the number of nodes. This connection formed vector is fed into the value network $V(S; \Theta_1)$ and adventure network and $A(S, v; \Theta_2)$.

Both the value network and adventure network are Multilayer Perceptron (MLP). The number of layers of the MLP and the size of each layer is adjustable.

It is only necessary to ensure that the size of the output of Value Network is $(1, 1)$ and the size of the output of Adventure Network is $(|V|, 1)$.

The final output of the Q-network is calculated with Equation 2.

5.5 Epsilon-greedy Algorithm

The Epsilon-greedy algorithm is a widely used approach in reinforcement learning to balance between exploration and exploitation. It allows the agent to select the action with the highest estimated value (exploitation) most of the time, while occasionally choosing a random action (exploration) to ensure comprehensive exploration of the action space.

At each time step t, the agent chooses an action a_t based on the following policy:

$$a_t = \begin{cases} \text{argmax}_a Q(s, a; \theta), & \text{with probability } 1 - \epsilon, \\ \text{random action}, & \text{with probability } \epsilon, \end{cases} \tag{10}$$

where $Q(s, a; \theta)$ represents the estimated action value for state s and action a, θ denotes the parameters of the Q-network and ϵ is the exploration rate, controlling the probability of selecting a random action.

To avoid the agent selecting the node that has already in the seed set, the estimated action value for state s and action a that choose the node already in the seed set will be set to "-inf" representing negative infinity. When doing a random action, the node selected by the random action will be checked. If it has already been in the seed set, this action will be ignored and do a random action again.

Initially, the exploration rate ϵ is often set to a relatively high value, encouraging the agent to explore different actions. As training progresses, ϵ is gradually reduced using a decay schedule. This allows the agent to rely more on the learned action values, striking a balance between gathering new experiences and exploiting existing knowledge.

5.6 Training the Q-Network

For each training session, a random mini-batch of experiences is sampled from the replay memory. Target Q-values are computed based on the immediate reward and the estimated future value of the next state using the target Q-network. The main Q-network's weights are updated using gradient descent (see Equation 3) to minimize the difference between predicted and target Q-values.

6 Experiments

In this section, several experiments are conducted to validate the design soundness and performance of the proposed DNPRL framework. The code used for experiments can be found at `https://github.com/YunmingHui/Master-Thesis`.

6.1 Datasets

Three different datasets used to carry out experiments are UC-IRV, Bitcoinalpha and Bitcoinotc. All of them can be structured as continuous graph models. The statistics of 3 datasets are shown in Table 1. The detailed information of datasets are as follows. The datasets used are same as those used in [15]. Science this is written by myself, I used what is in [15] in this subsection.

Dataset	Node number	Edge number	Start time	End time	Multiple edges
UC-IRV	1,889	59,835	15/04/2004	26/10/2004	Yes
Bitcoinalpha	3,783	24,186	0	164,246,400	No
Bitcoinotc	5,881	35,592	0	164,442,412	No

Table 1: Statistics of three datasets. Multiple edges indicate weather existing multiple edges whose start and end nodes are same but timestamp is different

- **UC-IRV**[67] UC Irvine messages dataset includes the users that sent or received at least one message in a Facebook-like Social Network originating from an online community for students at the University of California, Irvine. The database contains a total of 59,835 online messages, each message contains the sender, receiver and time of sending. The dataset is used to form a directed dynamic social network. Each edge represents a message which is directed from the sender of the message to the receiver of the message. In this dataset an edge with same start and end node will appear multiple times at different time points.

- **Bitcoinalpha**[1] Bitcoinalpha is a who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin Alpha. The dataset contains a total of 24,186 credit rating records from one user to another. Each record contains the person who made the credit rating and the person who was rated, the rating level and the time the rating was made. The dataset is used to form a directed dynamic social network. Each edge represents a credit rating record which is directed from the rating maker to the rate. In this dataset, any edge will appear only once.

- **Bitcoinotc**[2] Bitcoinotc is similar to Bitcoinalpha, but it is collected from another Bitcoin platform, Bitcoin OTC. The number of records in this dataset is 35,592. The format of records is the same as that of Bitcoinalpha. In this dataset, any edge will appear only once.

6.2 Experimental Settings

The following experiments are carried out:

- **Nodes representation pre-generation** The node representations of three datasets are pre-generated by TGNs and tested with dynamic link prediction task

[1] `http://snap.stanford.edu/data/soc-sign-bitcoin-alpha.html`
[2] `https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html`

- **Parameters turning** Several experiments are carried out to find the optimal parameters.
- **Ablation study** Several experiments are carried out to evaluate the necessarily of different parts of DNPRL.
- **Modele training** The models for the three datasets are trained with the optimal parameters and the training processes are analysed.
- **Framework evaluation** The framework is evaluated by comparing with several state of the art methods.

6.3 Metrics

The metrics used in experiments are as follows:

- **Dynamic link prediction** The dynamic link prediction experiment is a common practice in dynamic graph embedding research, aiming to evaluate the model's capacity to forecast the formation of links between nodes over time. By employing a dynamic graph dataset with distinct time snapshots, the model's ability to capture evolving relationships is assessed. The evaluation hinges on the Area Under the ROC Curve (AUC) and Average Precision (AP) metrics. AUC measures the model's proficiency in distinguishing actual linkages from non-existent ones, while AP gauges the trade-off between precision and recall. Elevated AUC and AP values signify enhanced predictive performance, indicative of the model's adeptness at identifying links within the dynamic graph's temporal evolution.
- **Activation time** The activation time is average length of time each node, except the initially activated nodes, is in active state (calculated by DNPIC). The larger the activation time is, the higher the influence of the seed set is.
- **Episode score** The episode score refers to the activation time caused by the seed set formed from the actions the agent done in one episode. As the higher activation time indicates the larger influence of the seed set, the higher episode score means the actions made by the agent is better.
- **training curve** The training curve reflects the **multiple learning processes** of a RL model. Taking Figure 16 as an example curve, the line indicate the average episode score of each episode and the shaded areas show extreme values that deviate from the mean. Taking the last episode as an example. The average episode score of the last episode of all learning processes is around 205,000, the biggest episode score of the last episode among the learning processed is around 221,000, the smallest episode score of the last episode among the learning processed is around 201,000.

6.4 Dynamic Graph Embedding

6.4.1 Parameters for Dynamic Graph Embedding

The graph embedding method TGNs is implied using code[3] provided by authors. The parameters of TGNs used in experiments are as follows: the dimension of node embedding is 64, the learning rate is 0.001, the drop out rate is 0.1, the batch seize is 200, the number of epochs is 50, the number of neighbours to sample is 20, the embedding model us graph attention network whose number of layer is 1 and number of heads used in attention layer is 2. The parameters of all three datasets are the same.

6.4.2 Node Representations Pre-generation and Evaluation

Since the node representations are the basis of the entire framework and all subsequent processing are based on them, the quality of the generated node representations needs to be ensured. As motivated in Subsection 5.2, as a commonly used metric, the dynamic link prediction task is used to evaluate the pre-trained node reprsentations.

Since there is randomness in the process of node representation generation. The TGNs are run 5 times for each dataset and the node representations with the best performance on dynamic link prediction task is used for the follow-up experiments. 80% of the data in the dataset is used for training and evaluation, 20% of the data in the dataset is used for testing. The performance of dynamic link prediction task on the three dataset is shown in Table 2.

Dataset	UC-IRV	Bitcoinalpha	Bitcoinotc
AUC	0.904	0.855	0.846
AP	0.906	0.856	0.862

Table 2: Link prediction task results on the three datasets

To the UC-IRV dataset, the model exhibits strong performance with an AUC of 0.904 and an AP of 0.906. These high values indicate that the node representations effectively capture the evolving relationships between nodes, enabling accurate link prediction. Moving to the Bitcoinalpha dataset, the model maintains solid performance with an AUC of 0.855 and an AP of 0.856. Although slightly lower than the UC-IRV dataset, these results suggest that the node representations continue to provide meaningful insights into the changing link patterns within this dataset. Finally, the Bitcoinotc dataset also demonstrates commendable results, featuring an AUC of 0.846 and an AP of 0.862. Notably, the high AP score suggests that the node representations excel in predicting dynamic link changes within this dataset.

The experiment results show that the node representations in the model exhibit strong performance across all three dynamic link prediction datasets. The consistently high AUC and AP scores indicate that the representations effectively capture the evolving nature of node relationships, making them valuable assets for accurately predicting changes in link connectivity within dynamic networks.

[3] `https://github.com/twitter-research/tgn`

This performance underscores the significance of the employed node representation method for dynamic network analysis.

6.5 Parameter Turing and Setting

In this subsection, the results of experiments carried out for parameter turning purpose are shown and discussed. The determined optimal parameters can be found in Subsection 6.5.4.

All the experiments in this subsection are carried out 5 layers to avoid the randomness and analysis the robustness of different settings.

6.5.1 Network Structure

To determine the best structure of the Q-network, two different networks with different number of layers are designed (see Figure 9). In both networks, the size of the temporal graph attention network is fixed for 2 layers but sthe dimension of vectors representing each node is different. For the large network, the dimension of vectors representing each node is 2, making the output of the temporal graph attention network is a $2|v|$ vector. Both the value and adventure networks both have two layers. To the value network, the output of the first layer is $1.5|V|$, and the output of the second layer is $1.5|V|$. For the small network, the dimension of vectors representing each node is 2, making the output of the temporal graph attention network is a $|v|$ vector. Both the value and adventure networks both only have a fully connected layer.

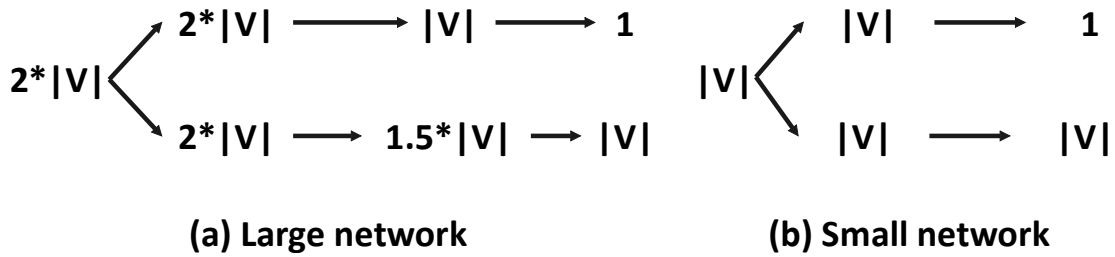

Figure 9: Two networks with different size

Two different models using these two different network structures are trained for the UC-IRV network. When training the model, the parameters are kept the same except for the size of the network used which is different. The training curves of the two models are shown in Figure 10

The experiments show that although the smaller network converge faster than the larger network, the average episode score is worse after both two models are stable. Therefore, this large network is chosen.

6.5.2 Frequency to Update Target Q-network

In this subsection, different update frequencies of target Q-network are tested to get the optimal frequency to update the target Q-network. The tested update frequencies includes every 200, 150, 100 and 50 episodes.

Four different models are trained for UC-IRV with these four different updating frequencies. The training curves of the four different models are shown in Figure

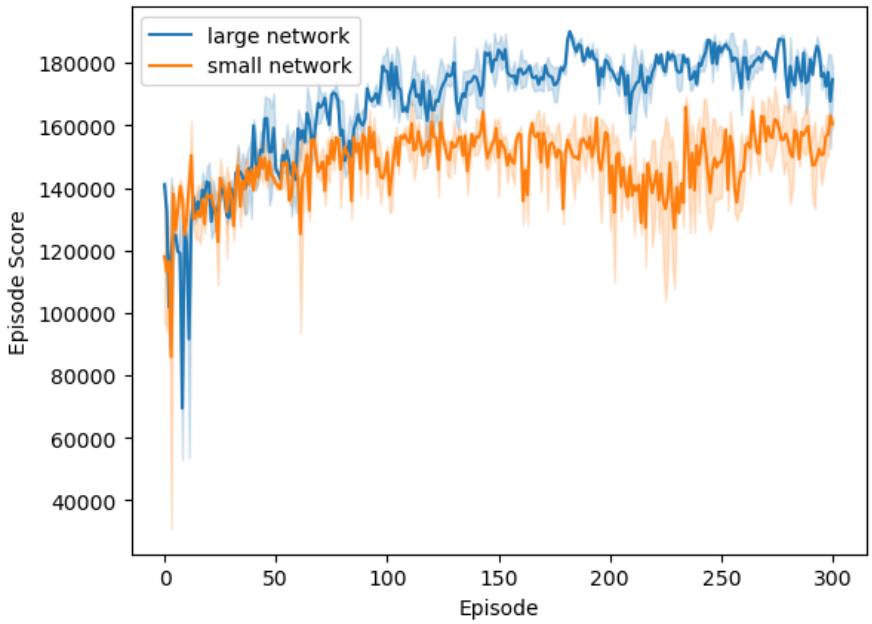

Figure 10: Training curve of two models with different network size

11. The experiments show that updating the target Q-network every 200 episodes can lead to the highest episode score with the convergence time (training time) similar to other settings. Therefore, the frequency that update the target q-network every 200 episodes is chosen.

6.5.3 Learning Rate

In this subsection, different learning rates are tested to get the optimal learning rate. The tested learning rates includes $1e-2$, $1e-3$ and $1e-4$. Three different models are trained for Bitcoinalpha with these three different learning rates.

The training curve of the three different models are shown in Figure 12. Experiments show that learning rate $1e-4$ is too low which make the model get stuck in local minima and struggle to escape. The small steps taken during updates are not sufficient to overcome barriers and reach a better global minimum. This makes the performance of the trained model very poor. Learning rate $1e-2$ is too low making the training process very unstable and leading the model to fail to converge. This learning rate $1e-3$ is more appropriate and allows the model to obtain a good performance. Thus, the learning rate $1e-3$ is chosen.

6.5.4 Parameters for RL

Based on the parameters turning experiments carried out before. The parameters are set as follows: the learning rate is $1e-3$, the dimension of node representations is 64, the size of replay buffer is 10000, the discount factor is 0.95, the batch size is 32, initial exploration rate is 1, minimum exploration rate is 0.2, exploration decay rate is 0.98, the frequency to update Target Q-network is every 200 episodes, number of updates to main Q-network is 5, the budget of seed set used for training is 30.

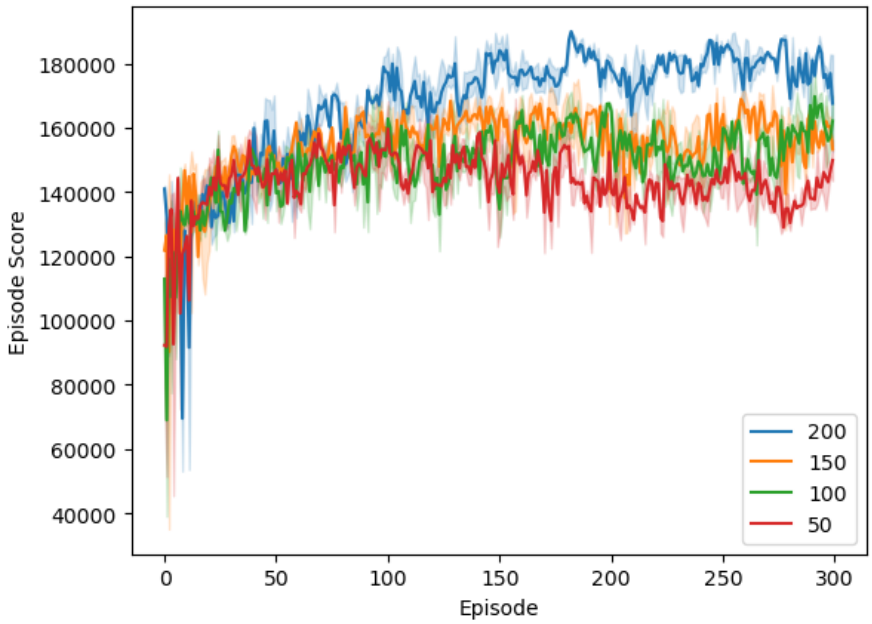

Figure 11: Training curve of four models with different frequencies to update Target Q-network

6.6 Ablation Study

In this subsection, several ablation studies are carried out to validate the necessity of each part of in the proposed framework.

6.6.1 Dueling Double DQN

As described in Subsection 4.2, the dueling double DQN is an extension to DQN which fuses the improvements to DQN made by dueling DQN and double DQN. In this experiment, it will be verified that both two extensions contribute to the final result.

To verify the necessity of using Dueling Double DQN, four models are trained for UC-IRV using the DQN, Dueling DQN, Double DQN, and dueling double DQN respectively. The training curves of the four different models are shown in Figure 13.

From the experimental results, it is clear that Dueling Double DQN has the best performance for the following three reasons.

- **Average episode score** On average episode score, Dueling Double DQN is the highest. Average episode score intuitive indication of how well the final trained model performs on the dynamic non-progressive IM problem. Obviously, models trained by Dueling Double DQN has the highest average episode score.

- **Robustness** Ensuring robustness, characterized by the presence of randomness across multiple experiments, holds significant importance. Given the considerable time investment required for training, a low degree of model randomness (i.e., high robustness) alleviates the necessity for repetitive training iterations to attain the optimal model. This concept is illustrated in Figure

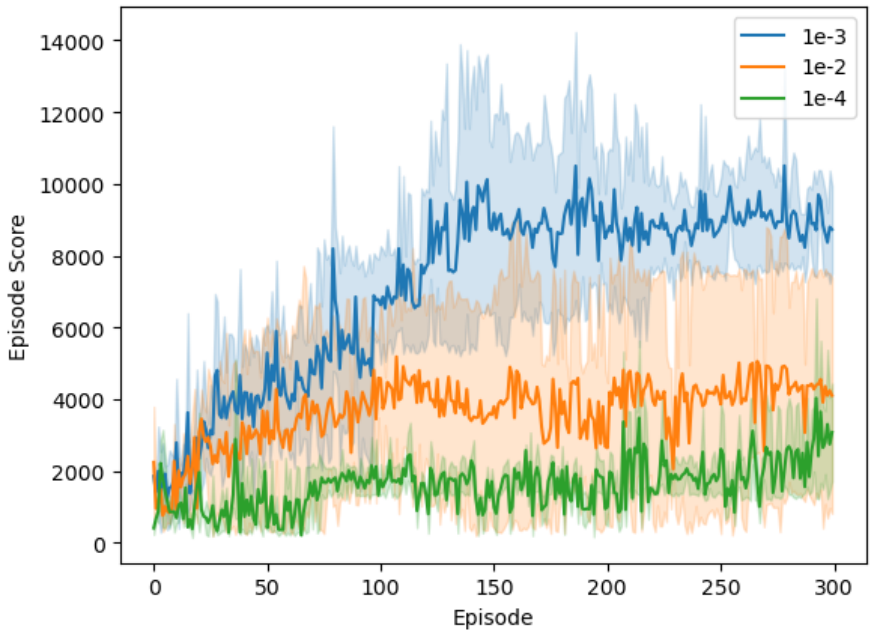

Figure 12: Training curve of three models with three different learning rates

13, where the influence area of Dueling Double DQN is minimized, signifying the robust nature of the model training process facilitated by Dueling Double DQN.

- **Convergence speed** Dueling Double DQN converges slightly slower than DQN and Doubel DQN, but faster than Dueling DQN. Although the convergence rate is slightly lower, given that Dueling Double DQN leads to far better average episode score. Therefore, it is worthwhile to sacrifice some of the convergence speed to bring about greater gains in average episode score.

In conclusion, based on the comparison of four different DQN algorithms (DQN, Dueling DQN, Double DQN, and Dueling Double DQN) in addressing the UC-IRV problem, the experimental results unequivocally demonstrate the necessity of employing the Dueling Double DQN approach. With the highest average episode score and proven stability across multiple experiments, Dueling Double DQN not only preserves valuable time and resources during practical training but also justifies its slightly slower convergence rate. Considering the substantial performance advantage it offers in terms of average episode scores, opting for the Dueling Double DQN algorithm emerges as a prudent choice for tackling the dynamic non-progressive IM problem.

6.6.2 Reward Strategy

In this subsection, the proposed strategy will be compared with the normal reward strategy introduced in Subsection 5.3 to validate the rationality and necessity of the proposed methodology.

Two different models are trained for UC-IRV with the proposed reward strategy and the proposed reward strategy. The training curves of the two different models are shown in Figure 14.

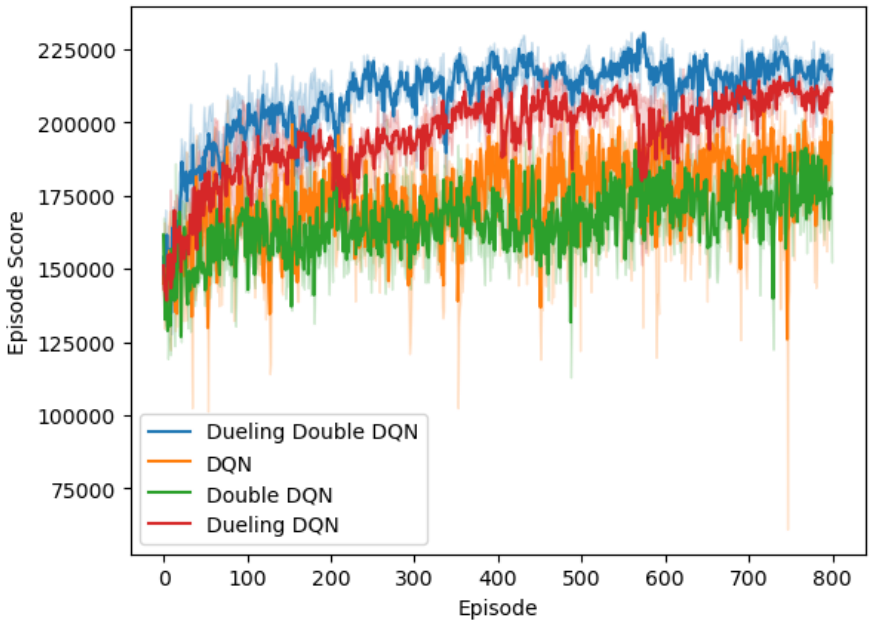

Figure 13: Training curve of four models with different DQN algorithms

From the experiments, it is clear that the proposed reward strategy is far superior to the normal reward strategy, from average episode score, robustness and convergence speed.

To explain the reasons for this advantage, ten points will be randomized. These ten points will be used as the actions made by the agent in an episode. Then the reward of these ten actions will be calculated using two different strategies. The results are shown in Table 3.

Derived from the conducted experiments, a salient observation becomes evident: within the rewards computed using the conventional reward strategy, the initial action stands out with a markedly high reward. Conversely, subsequent actions exhibit considerably lower rewards in contrast to the first action. Interestingly, despite the augmented reward for the first action, the actual influence of its corresponding node remains limited. This incongruity highlights a significant concern related to the conventional reward strategy, wherein an overestimation of the initial action is apparent, consequently introducing complexities to the training process.

In contrast, the introduced reward strategy rectifies this phenomenon. The recalibrated approach mitigates the pronounced overestimation associated with the first action and yields more precise reward allocations. Consequently, the training process substantially benefits from this corrective measure. Based on these empirical insights, the adoption of the proposed reward strategy emerges as both an imperative and well-justified improvement.

To conclude, this subsection effectively establishes the validity and necessity of the proposed novel reward strategy. By comparing it with the conventional approach, it becomes evident that the new strategy surpasses it in terms of performance metrics such as average episode score, robustness, and convergence speed. Additionally, through an experiment involving ten specific actions, the shortcomings of the conventional strategy are highlighted. The proposed strategy rectifies

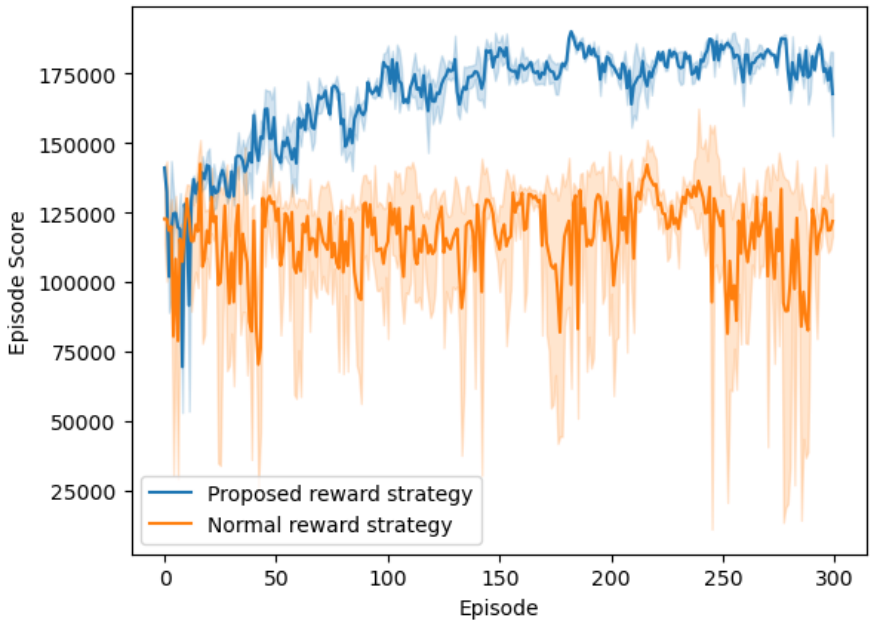

Figure 14: Training curve of two models with proposed reward strategy and the proposed reward strategy

Action	399	8	41	104	522	31	2	102	18	40
Normal Reward Strategy	0.6247	0.1605	0.0192	0.0556	0.0138	0.0202	0.0290	0.0298	0.0156	0.0157
Proposed Reward Strategy	0.0235	0.0057	0.1124	0.0471	0.0590	0.0412	3.6410	0.0471	5.446	0.4201
Influence	91	22	434	182	228	159	14049	182	21017	1621

Table 3: Reward of 10 random actions calculated with proposed reward strategy and normal reward strategy. The first row is the node selected by each action. The second and third rows are the rewards of corresponding actions calculated by the two reward strategies. The fourth row is the influence of corresponding node calculated with the method introduced in Subsection 5.3.

the overestimation of the initial action's importance, thus addressing training difficulties associated with skewed rewards.

6.6.3 Necessity of Pre-generating Node Representations

In the framework, the node representations are pre-generated. In fact, the network used for graph embedding can also be incorporated into the training process. Two different models are trained for UC-IRV with and without pre-generating the node representations.

For better comparison, the results of only one training session are shown. The model trained with pre-generating the node representations is trained for 8 hours and the model trained without pre-generating the node representations is trained for 24 hours.

The experiments (see Figure 15) show that adding the network used for graph embedding into the training process will extremely increase the complexity of training. The model without pre-generating the node representations is only trained for around 100 episodes for 24 hours compared with the model with pre-

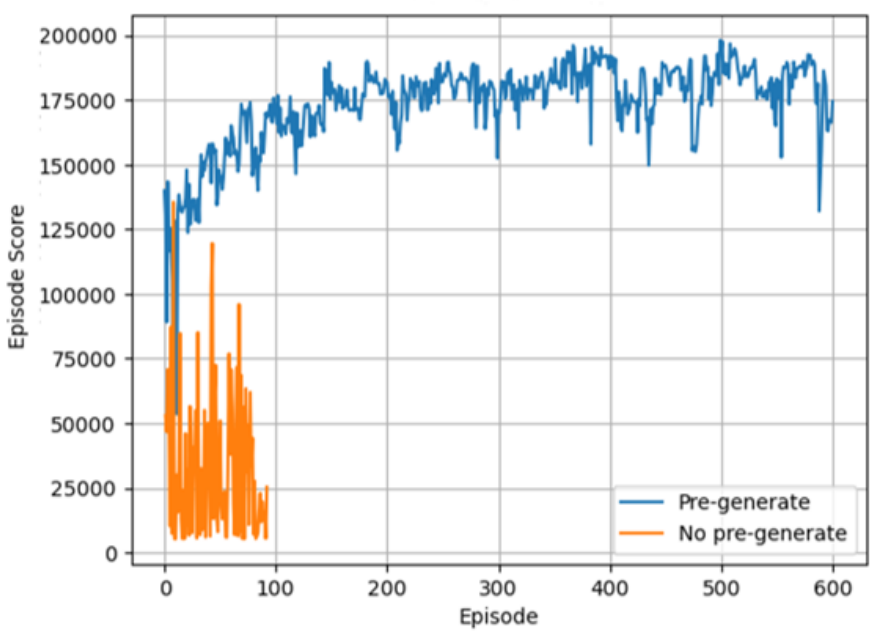

Figure 15: Training curve of two models with and without pre-generating the node representations

generating the node representations is trained for around 600 episodes for 8 hours. What's worse, the model without pre-generating the node representations cannot converge within 24 hours.

In short, pre-generated node representations speed up training and convergence and lead to better performance.

6.7 Model Training

In this subsection, the training process of the models for the three datasets using the optimal parameters is demonstrated and evaluated. The training of the three models is performed over ten repetitions.

The training process of the model for UC-IRV, shown in Figure 16), exhibits a remarkable trait of relatively swift convergence, typically reaching convergence around the 300-epoch mark. This convergence behavior is notably consistent across multiple training runs, as evidenced by the comparatively narrow shaded region on the graph. In stark contrast, the training processes of the models designed for Bitcoinalpha and Bitcoinotc present a distinctly different profile (as shown in Figure 16 and Figure 18 respectively), characterized by a slower convergence trajectory. Notably, the model tailored for Bitcoinalpha demands a significantly more extended training period, often necessitating approximately 800 epochs to achieve convergence.

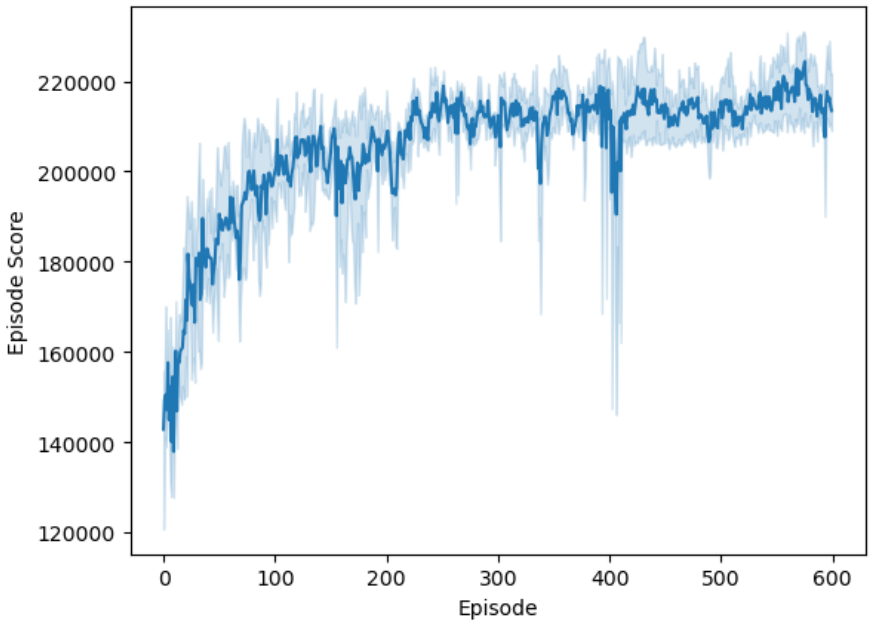

Figure 16: Training curve of final model for UC-IRV

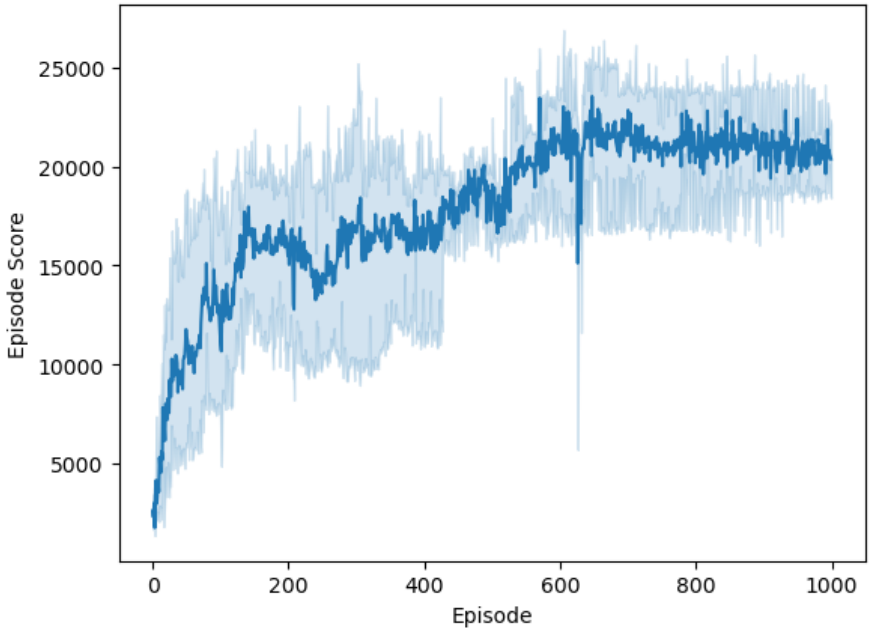

Figure 17: Training curve of final model for Bitcoinalpha

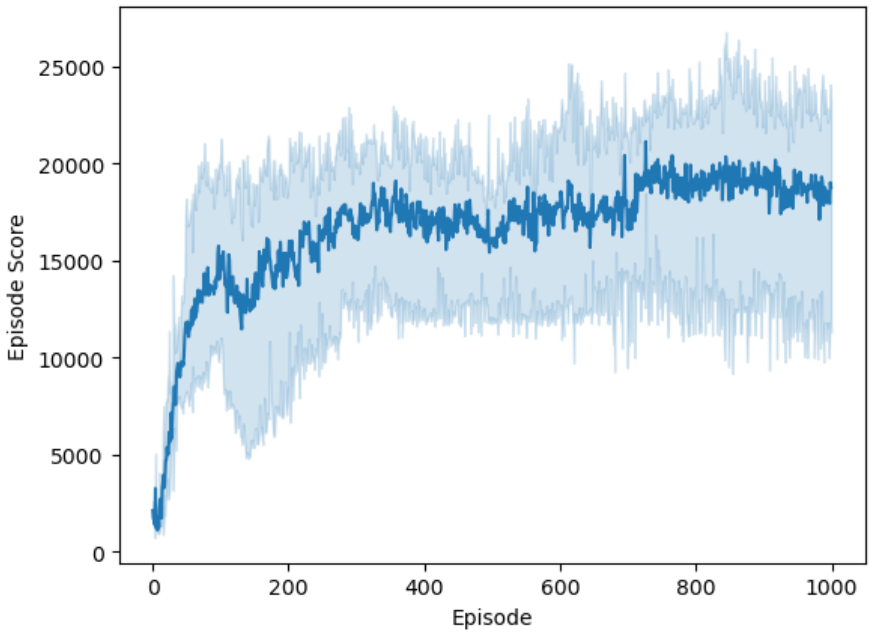

Figure 18: Training curve of final model for Bitcoinotc

For each dataset, the repeatability between the nodes in seed sets generated by the 10 models obtained from 10 training sessions is tested. Taking the seed sets with 10 as an example, the results are shown in Table 4. Similar to the conclusions drawn from the analysis of the training process, the repetition rate of seed sets generated from the model trained for UC-IRV is higher, the repetition rate of seed sets generated from the model trained for UC-IRV are relatively low.

Dataset	UC-IRV	Bitcoinalpha	Bitcoinotc
Repeatability	68.9%	50.2%	41.9%

Table 4: Repetition rate between nodes in seed sets (taking size of seed set = 10 as an example) generated by models trained multiple times

Overall the training process of the model on UC-IRV is more undetermined and the generated model is more stable. The robustness on UC-IRV is high. Lower robustness on Bitcoinalpha and Bitcoinotc may require multiple repetitions of the experiment to select a better model.

6.8 Framework Evaluation

In this subsection, the quality of seed set generated is evaluated by comparing with that generated by other state of the art influence maximization methods.

6.8.1 Baselines

To evaluate the performance of the framework, four state of the art methods are chosen as the baselines. As there are no existing dynamic non-progressive influence maximization methods. One greedy method (CELF), a static reinforcement learning method (ToupleGDD), a dynamic progressive method (INDDSN), and a

static non-progressive method (TSGC) are chosen as the baselines. As all four baselines have already introduced in Section , here are just the reasons given for choosing them.

- **CELF**[68] The CELF algorithm is often selected as a baseline in influence maximization due to its computational efficiency and simplicity. Its ability to efficiently identify influential seed nodes makes it a practical benchmark for evaluating the performance of other algorithms. CELF's established reputation in the field and its widespread use as a comparison point enable consistent and meaningful comparisons between different studies, while also reflecting the trade-off between algorithm complexity and influence spread performance.

- **ToupleGDD**[64] ToupleGDD is the latest method that employ reinforcement learning to process IM problem and outperforms other similar reinforcement learning-based algorithms. The proposed framework is also based on reinforcement learning, so ToupleGDD is chosen as the baseline. The code and model used for evaluation is provided by the authors, available on `https://github.com/Dtrycode/ToupleGDD`.

- **INDDSN**[10] INDDSN is the most recognized method among the approaches to solving dynamic progressive IM problems. The reason for choosing INDDSN as baseline is to verify that the progressive approach cannot be applied to the non-progressive problem as the progressive IM problem is significantly different from the non-progressive problem.

- **TSGC** [69] TSGC is the most recognized method among the approaches to solving static non-progressive IM problems. The reason for choosing INDDSN as baseline is to verify that the static non-progressive approach cannot be applied to the dynamic non-progressive IM problem as the the dynamic nature of the network will make a difference to the problem.

Except the ToupleGDD, the code for the other three baselines methods is reproduced by myself according to the original paper.

6.8.2 Influence Spread

This subsection performs quantitative and qualitative analysis of the nodes selected by various methods to evaluate the performance of the proposed method. To quantitative analysis, the metrics for measuring the performance of methods is average of the length of time each node, except the initially activated nodes, is in active state, i.e. $\sigma_{DNP}(S) = \frac{\sum_{v \in V-S} \tau_v}{|V-S|}$, where τ_v is the sum of time length for which node v is active. In order to reflect the generalizability of the training-generated model to corresponding dataset, the quality of seed sets is tested on DNPIC (proposed diffusion model) with different time to return to inactive state $t_{inactive}$ and budget of seed set k.

The tested different time to return to inactive state $t_{inactive}$ are 1,440 (1 days), 10,080 (7 days), and 86,400s (60 days). The best performing models are chosen due to the high level of randomness in the model training process. The results are shown in Figure 19, Figure 20, and Figure 21.

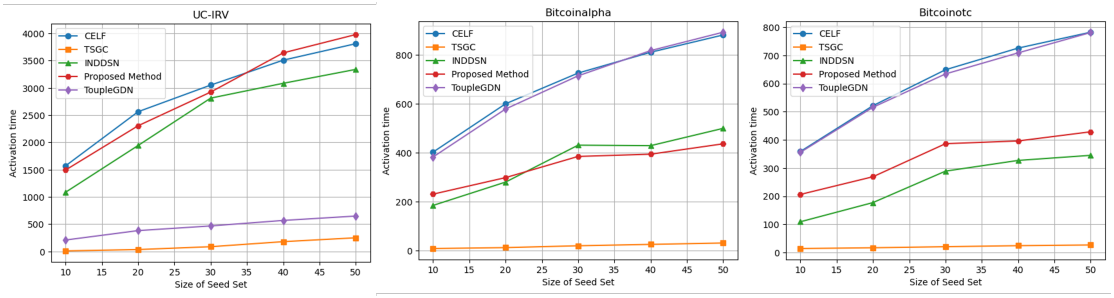

Figure 19: Influence of seed sets with different size on diffusion with $t_{inactive} = 1,440$

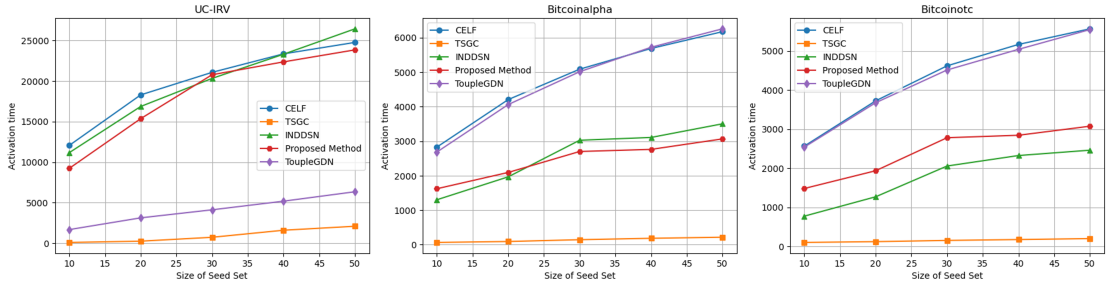

Figure 20: Influence of seed sets with different size on diffusion with $t_{inactive} = 10,080$

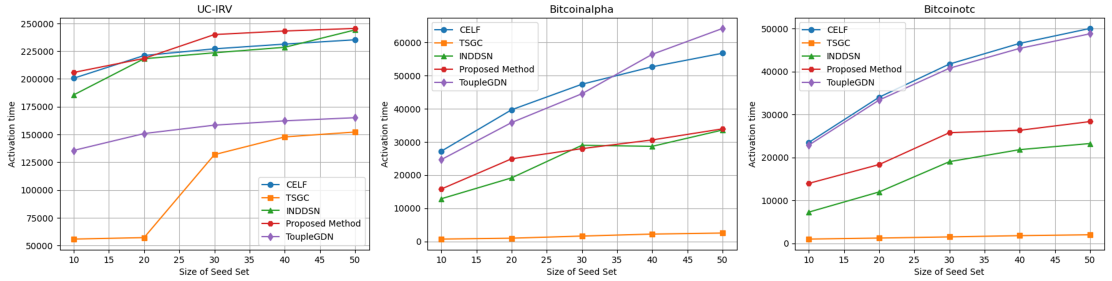

Figure 21: Influence of seed sets with different size on diffusion with $t_{inactive} = 86,400$

First, the proposed method is compared with TSGC and INDDSN comprehensively.

The results unmistakably showcase the proposed method's remarkable superiority over TSGC across all evaluated cases. This stark contrast gains significance from the fact that TSGC operates as a static, non-progressive method within the realm of influence maximization (IM). This demonstration firmly substantiates the pivotal role of network dynamics in shaping the IM landscape. The limitations of methods grounded solely in static IM paradigms become apparent when confronted with the complexities of dynamic IM challenges. Evidently, the exploration of dynamic, non-progressive IM problems holds profound implications.

Furthermore, the proposed method exhibits competitive performance against INDDSN in the majority of scenarios. Notably, INDDSN represents a dynamic

progressive approach, implying that the transition from a progressive to a non-progressive stance inherently introduces shifts within the IM framework. However, while the progressive to non-progressive shift is perceptible, it pales in comparison to the dynamic-to-static transformation in terms of its observable impact.

The proposed method is next compared to CELF and ToupleGDD. What is clear is that the proposed method has a good performance on UC-IRV, but is less effective on Bitcoinalpha and Bitcoinotc. Therefore the analysis will first focus on UC-IRV.

On the UC-IRV dataset, the proposed method performs comparably to CELF. Considering CELF's nature as a greedy algorithm, this indicates that the proposed method demonstrates strong performance across different models and various sizes of seed sets. It's important to note that the model is solely trained on a diffusion model with a fixed seed set size. This highlights the generalization capability of the proposed method within a single dataset. This is particularly significant, as scenarios involving distinct recovery times from an activated state to an inactivated state hold practical importance and may be encountered.

On the Bitcoinalpha and Bitcoinotc dataset, the proposed methods performs worse than CELF and ToupleGDD. I hypothesize that the main reason behind this is there are large number of repeated edges in UC-IRV. A shown in Tabble 1, there are only 1,899 nodes in UC-IRV, however 59,835 edges. In comparison, Bitcoinalpha and Bitcoinotc, with 3,783 and 5,881 nodes, respectively, exceed UC-IRV, but the number of edges is only 24,186 and 35,592, respectively, which is far below UC-IRV. This phenomenon makes the probability of a node being activated again once it reverts from an activated state to an inactive state relatively low in Bitcoinalpha and Bitcoinotc. In other words, the properties of non-progressive and dynamic is weaker in these two datasets. This is corroborated by the better performance on UC-IRV in the dynamic link prediction task. It is quite possible that TGNs themselves do not handle this type of dataset well, making the method end up performing poorly. This is why ToupleGDD does not perform well on UC-IRV. Since ToupleGDD uses static graph embedding method, when the dynamic and non-progressive nature of the dataset is strong, the static graph embedding method is unable to deal with the dynamic nature well, which makes the performance worse.

The above analysis illustrates the better performance of the proposed method on dense and dynamic social networks. In addition, the proposed method has a good generalization to a single dataset, eliminating the need to repeat training the model several times when the size of the set and the parameters in the diffusion model change.

6.8.3 Running Time

The running time of different methods are shown in Table 5. Although the proposed method requires a separate model to be trained for each dataset, the time taken is much less than that of ToupleGDD, which is also a reinforcement learning method. Although ToupleGDD can obtain a model for all datasets, the model needs to be retrained when the diffusion model is changed. As noted in chapter 3.2, there are a large number of existing diffusion models. In applications where multiple diffusion models need to be processed, the running time will become

uncontrollable. However, it has to be recognized that if only a single diffusion model is considered and multiple datasets need to be processed, ToupleGDD has a definite advantage.

Although both TSGC and INDDSN have relatively short run times, they are not relevant for discussion given that they are not as effective as the other methods. It is worth mentioning that CELF, as the most classical method, still has an advantage in terms of runtime, although it sometimes performs slightly worse than the proposed method or ToupleGDD.

Method	CELF	TSGC	INDDSN	Proposed Method		ToupleGDD	
				Training	Execution	Training	Execution
UC-IRV	4 hour	3 min	0.5-1.5 hour	3 hour	<1 min		<1 min
Bitcoinalpha	3 hour	5 min	0.5-1 hour	8 hour	<1 min	52 hour	<1 min
Bitcoinotc	7 hour	7 min	1-2.5 hour	16 hour	<1 min		<1 min

Table 5: Running time of different methods

6.9 Qualitative Analysis

In this subsection, experiments are carried out to seed if the framework is working as pre-designed. More specifically, whether the agent can make optimal decisions according to different states. Taking the final model trained for the UC-IRV as an example, the 10 actions with highest q-value at the first 5 states are shown in Table 6.

Actions with highest q-value	1	2	3	4	5	6	7	8	9	10
State 1	8	26	232	94	31	430	43	11	473	142
State 2	8	26	232	94	31	430	43	473	11	142
State 3	8	26	94	232	31	430	43	473	142	18
State 4	8	26	94	232	31	430	43	473	142	18
State 5	8	26	94	31	430	232	473	43	142	18

Table 6: The nodes selected by 10 actions with highest q-value at the first 5 states.

It is obvious that for different states, the actions with the highest q-value choose almost the same nodes only slightly different in order. This is different from what was pre-designed. And the ordering of the q-values of these actions is very close to the ordering of the influence of the selected nodes.

In response to this phenomenon, I see two possible reasons. The first is that the difference between each state (see Figure 22) is very small. After each action, only one number changes in the one-hot vector used to represent the seed set. With such a large input, it is difficult for such a small difference to make a significant difference to the output of the neural network.

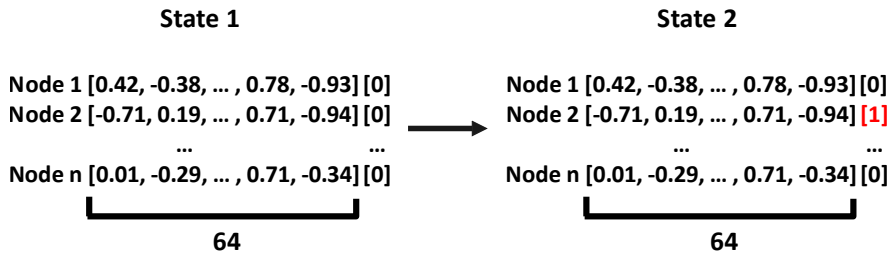

Figure 22: Difference between two states. The difference is marked in red.

The second possible reason is during the training process in order to keep the process running smoothly. The selection of nodes that already exist in the seed set is artificially set as unselectable, rather than learning so that the agent no longer selects these nodes. This makes the agent not focus on learning to make the best decision based on different states, but rather on learning the influence of each node. The decision is then made directly based on the level of influence.

7 Conclusion and Future Work

7.1 Conclusion

This master thesis focus on the dynamic non-progressive IM problem which is a variant of the IM problem that has not yet been studied before. This thesis carefully scrutinizes the relevant researches on the IM problem and illustrates the practical significance of this variant of dynamic non-progressive. Given the absence of a dynamic non-progressive diffusion model, a dynamic non-progressive diffusion model extended from the IC model, named DNPIC is proposed. Through the analysis of related researches, it is found that reinforcement learning is the most suitable for solving IM problems, but the existing methods based on reinforcement learning can't deal with dynamic IM problems well. Therefore, this thesis proposed a framework that combines dynamic graph embedding and reinforcement learning. Experiments are conducted to analyze the plausibility of this framework. And this framework is compared with other state of the art methods on real datasets. The experiment results show that the framework performs well on the dataset with a large number of repeated edges, but performs relatively poorly on sparse datasets.

7.2 Future work

The poor performance on sparse datasets is partly due to the lower quality of node representations. This is due to the fact that existing graph embedding methods, TGNs, is used directly without being adapted accordingly for the IM problem, and TGNs itself performs poorly on sparse datasets. Therefore, a dynamic graph embedding method more suitable for IM problems is an interesting research direction.

It is also worth thinking about how to incorporate influence maximization related attributes (e.g., the current seed set) into the state. In the current framework influence maximization related attributes are only included in the state with the current seed set and the percentage in the state is quite low. This makes the

influence of influence maximization related attributes on the agent's action policy is quite low.

In order to allow the training process to continue without being interrupted by invalid actions (actions choose the node already in the seed set), these invalid actions are set unselectable, instead of letting the agent learn such strategy. In turn, the node representations are invariant, which makes the agent according easy to ignore the differences between different states. How to change this defect is also worth exploring. One possible approach is to give a larger negative reward when the agent chooses an illegal action. However the size of the negative reward needs to be tested in more experiments.

In addition, the use of Dueling Double DQN leads to unavoidable need for training a model for each dataset. Although the use of pre-trained node representations somewhat mitigates the resulting time-consuming problem, it still doesn't meet practical usage requirements. Therefore, it is also worth exploring the use of methods such as transfer learning to make it possible to obtain a model that works for all datasets. think transfer leaning is a very promising technique for solving this problem.

References

[1] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018.

[2] Suman Banerjee, Mamata Jenamani, and Dilip Kumar Pratihar. A survey on influence maximization in a social network. *Knowledge and Information Systems*, 62:3417–3455, 2020.

[3] Yuxin Ye, Yunliang Chen, and Wei Han. Influence maximization in social networks: theories, methods and challenges. *Array*, page 100264, 2022.

[4] 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.

[5] Kristo Radion Purba, David Asirvatham, and Raja Kumar Murugesan. Influence maximization diffusion models based on engagement and activeness on instagram. *Journal of King Saud University-Computer and Information Sciences*, 34(6):2831–2839, 2022.

[6] 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.

[7] T-H Hubert Chan, Li Ning, and Yong Zhang. Influence maximization under the non-progressive linear threshold model. In *International Workshop on Frontiers in Algorithmics*, pages 37–48. Springer, 2020.

[8] Vincent Yun Lou, Smriti Bhagat, Laks VS Lakshmanan, and Sharan Vaswani. Modeling non-progressive phenomena for influence propagation. In *Proceedings of the second ACM conference on Online social networks*, pages 131–138, 2014.

[9] Honglei Zhuang, Yihan Sun, Jie Tang, Jialin Zhang, and Xiaoming Sun. Influence maximization in dynamic social networks. In *2013 IEEE 13th International Conference on Data Mining*, pages 1313–1318. IEEE, 2013.

[10] Charu C Aggarwal, Shuyang Lin, and Philip S Yu. On influential node discovery in dynamic social networks. In *Proceedings of the 2012 SIAM International Conference on Data Mining*, pages 636–647. SIAM, 2012.

[11] Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6):1420–1443, 1978.

[12] Hui Li, Mengting Xu, Sourav S Bhowmick, Joty Shafiq Rayhan, Changsheng Sun, and Jiangtao Cui. Piano: influence maximization meets deep reinforcement learning. *IEEE Transactions on Computational Social Systems*, 2022.

[13] Shan Tian, Songsong Mo, Liwei Wang, and Zhiyong Peng. Deep reinforcement learning-based approach to tackle topic-aware influence maximization. *Data Science and Engineering*, 5:1–11, 2020.

[14] Khurshed Ali, Chih-Yu Wang, and Yi-Shin Chen. Boosting reinforcement learning in competitive influence maximization with transfer learning. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 395–400. IEEE, 2018.

[15] Shihan Wang Yunming Hui, Melisachew Wudage Chekol. Leveraging graph embedding for opinion leader detection in dynamic social networks. In *The 2nd edition of the Semantic Data Mining (SEDAMI) Workshop*, 2023.

[16] Nesrine Hafiene, Wafa Karoui, and Lotfi Ben Romdhane. Influential nodes detection in dynamic social networks: A survey. *Expert Systems with Applications*, 159:113642, 2020.

[17] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[18] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[19] John C Gower and Gavin JS Ross. Minimum spanning trees and single linkage cluster analysis. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 18(1):54–64, 1969.

[20] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3):211–223, 2001.

[21] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. Prediction of information diffusion probabilities for independent cascade model. In *International conference on knowledge-based and intelligent information and engineering systems*, pages 67–75. Springer, 2008.

[22] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. Sparsification of influence networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 529–537, 2011.

[23] Konstantin Kutzkov, Albert Bifet, Francesco Bonchi, and Aristides Gionis. Strip: stream learning of influence probabilities. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 275–283, 2013.

[24] Roy M Anderson and Robert M May. *Infectious diseases of humans: dynamics and control*. Oxford university press, 1991.

[25] Nathalie TH Gayraud, Evaggelia Pitoura, and Panayiotis Tsaparas. Diffusion maximization in evolving social networks. In *Proceedings of the 2015 acm on conference on online social networks*, pages 125–135, 2015.

[26] Zekun Liu, Jianyong Yu, Yuqi Liu, and Hangyu Zhu. Maximizing the influence in dynamic social networks: An entropy-based linear threshold model. In *2022 IEEE 9th International Conference on Cyber Security and Cloud Computing (CSCloud)/2022 IEEE 8th International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pages 111–116. IEEE, 2022.

[27] Guangmo Tong, Weili Wu, Shaojie Tang, and Ding-Zhu Du. Adaptive influence maximization in dynamic social networks. *IEEE/ACM Transactions on Networking*, 25(1):112–125, 2016.

[28] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[29] 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.

[30] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

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

[32] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[33] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5363–5370, 2020.

[34] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *Companion proceedings of the the web conference 2018*, pages 969–976, 2018.

[35] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. Dynamic network embedding: An extended approach for skip-gram based network embedding. In *IJCAI*, volume 2018, pages 2086–2092, 2018.

[36] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–728, 2020.

[37] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. Dyrep: Learning representations over dynamic graphs. In *International conference on learning representations*, 2019.

[38] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

[39] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[40] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[41] Shweta Bhatt. Reinforcement Learning 101, 2018.

[42] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[43] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[44] Jan Peters. Policy gradient methods. *Scholarpedia*, 5(11):3698, 2010.

[45] 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.

[46] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[47] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[48] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[49] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[50] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[51] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

[52] Hiroki Nakajima and Hitoshi Iima. The solution of combinatorial optimization problems based on reinforcement learning. In *Proceedings of the 2017 International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*, pages 78–82, 2017.

[53] Dong Yan, Jiayi Weng, Shiyu Huang, Chongxuan Li, Yichi Zhou, Hang Su, and Jun Zhu. Deep reinforcement learning with credit assignment for combinatorial optimization. *Pattern Recognition*, 124:108466, 2022.

[54] Runbo Ni, Xueyan Li, Fangqi Li, Xiaofeng Gao, and Guihai Chen. Fastcover: An unsupervised learning framework for multi-hop influence maximization in social networks. *arXiv preprint arXiv:2111.00463*, 2021.

[55] Cai Zhang, Weimin Li, Dingmei Wei, Yanxia Liu, and Zheng Li. Network dynamic gcn influence maximization algorithm with leader fake labeling mechanism. *IEEE Transactions on Computational Social Systems*, 2022.

[56] Ying Li, Linlin Li, Yijun Liu, and Qianqian Li. Mahe-im: multiple aggregation of heterogeneous relation embedding for influence maximization on heterogeneous information network. *Expert Systems with Applications*, 202:117289, 2022.

[57] Sanjay Kumar, Abhishek Mallik, Anavi Khetarpal, and BS Panda. Influence maximization in social networks using graph embedding and graph neural network. *Information Sciences*, 607:1617–1636, 2022.

[58] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 385–394, 2017.

[59] Sanjay Kumar, Abhishek Mallik, and BS Panda. Influence maximization in social networks using transfer learning via graph-based lstm. *Expert Systems with Applications*, 212:118770, 2023.

[60] Hui Li, Mengting Xu, Sourav S Bhowmick, Changsheng Sun, Zhongyuan Jiang, and Jiangtao Cui. Disco: Influence maximization meets network embedding and deep learning. *arXiv preprint arXiv:1906.07378*, 2019.

[61] 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.

[62] Chao Wang, Yiming Liu, Xiaofeng Gao, and Guihai Chen. A reinforcement learning model for influence maximization in social networks. In *Database Systems for Advanced Applications: 26th International Conference, DASFAA 2021, Taipei, Taiwan, April 11–14, 2021, Proceedings, Part II*, pages 701–709. Springer, 2021.

[63] 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.

[64] Tiantian Chen, Siwen Yan, Jianxiong Guo, and Weili Wu. Touplegdd: A fine-designed solution of influence maximization by deep reinforcement learning. *IEEE Transactions on Computational Social Systems*, 2023.

[65] 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.

[66] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

[67] Pietro Panzarasa, Tore Opsahl, and Kathleen M Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009.

[68] 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.

[69] Kalyanee Devi and Rohit Tripathi. Identification of best social media influencers using icirs model. *Computing*, pages 1–23, 2023.

[70] 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.

[71] Chuan Zhou, Peng Zhang, Wenyu Zang, and Li Guo. On the upper bounds of spread for greedy algorithms in social network influence maximization. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2770–2783, 2015.

[72] Yu Wang, Gao Cong, Guojie Song, and Kunqing Xie. Community-based greedy algorithm for mining top-k influential nodes in mobile social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1039–1048, 2010.

[73] Qingye Jiang, Guojie Song, Cong Gao, Yu Wang, Wenjun Si, and Kunqing Xie. Simulated annealing based influence maximization in social networks. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[74] Yandi Li, Haobo Gao, Yunxuan Gao, Jianxiong Guo, and Weili Wu. A survey on influence maximization: From an ml-based combinatorial optimization. *arXiv preprint arXiv:2211.03074*, 2022.

[75] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *Knowledge Discovery in Databases: PKDD 2006: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases Berlin, Germany, September 18-22, 2006 Proceedings 10*, pages 259–271. Springer, 2006.

[76] 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.

[77] Jinha Kim, Seung-Keol Kim, and Hwanjo Yu. Scalable and parallelizable processing of influence maximization for large-scale social networks? In *2013 IEEE 29th international conference on data engineering (ICDE)*, pages 266–277. IEEE, 2013.

[78] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *2011 IEEE 11th international conference on data mining*, pages 211–220. IEEE, 2011.

[79] Sainyam Galhotra, Akhil Arora, and Shourya Roy. Holistic influence maximization: Combining scalability and efficiency with opinion-aware models. In *Proceedings of the 2016 international conference on management of data*, pages 743–758, 2016.

[80] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[81] Linyuan Lü, Yi-Cheng Zhang, Chi Ho Yeung, and Tao Zhou. Leaders in social networks, the delicious case. *PloS one*, 6(6):e21202, 2011.

[82] Qi Liu, Biao Xiang, Enhong Chen, Hui Xiong, Fangshuang Tang, and Jeffrey Xu Yu. Influence maximization over large-scale social networks: A bounded linear approach. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, pages 171–180, 2014.

[83] Kyomin Jung, Wooram Heo, and Wei Chen. Irie: Scalable and robust influence maximization in social networks. In *2012 IEEE 12th international conference on data mining*, pages 918–923. IEEE, 2012.

[84] Ramasuri Narayanam and Yadati Narahari. A shapley value-based approach to discover influential nodes in social networks. *IEEE Transactions on Automation Science and Engineering*, 8(1):130–147, 2010.

[85] Suqi Cheng, Huawei Shen, Junming Huang, Wei Chen, and Xueqi Cheng. Imrank: influence maximization via finding self-consistent ranking. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 475–484, 2014.

[86] Suqi Cheng, Huawei Shen, Junming Huang, Guoqing Zhang, and Xueqi Cheng. Staticgreedy: solving the scalability-accuracy dilemma in influence maximization. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 509–518, 2013.

[87] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Dynamic and historical shortest-path distance queries on large evolving networks by pruned landmark labeling. In *Proceedings of the 23rd international conference on World wide web*, pages 237–248, 2014.

[88] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, pages 629–638, 2014.

[89] 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.

[90] 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.

[91] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Chen Chen. Bring order into the samples: A novel scalable method for influence maximization. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):243–256, 2016.

[92] Radosław Michalski, Jarosław Jankowski, and Piotr Bródka. Effective influence spreading in temporal networks with sequential seeding. *IEEE Access*, 8:151208–151218, 2020.

[93] Yuying Zhao, Yunfei Hu, Pingpeng Yuan, and Hai Jin. Maximizing influence over streaming graphs with query sequence. *Data Science and Engineering*, 6(3):339–357, 2021.

A Non-machine Learning Influence Maximization Methods

The non-machine learning influence maximization methods are categorized according to the type of IM problem being solved. Since there is no research related to dynamic non-progressive IM problem yet, there are a total of three categories.

A.1 Classic Static Progressive IM Problem

In this section, non-machine learning approaches for solving classic IM problems will be reviewed. The main non-machine learning methods include simulation-based approaches, proxy-based approaches, and sampling-based approaches.

The classic IM problem under IC, LT, TR, and CT diffusion models has been proven to be NP-hard which makes related studies try to find approximate solutions with theoretical guarantees.

Most approaches proposed to solve the classic IM problem are based on the greedy framework as shown in Algorithm 3. The greedy framework iteratively puts a node u that can provide the maximum marginal gain to the influence function to the seed set S. According to the diffusion model, violently computing the influence function would be time-consuming and infeasible for any real dataset, so how to estimate the influence function efficiently has become a concern for almost all studies.

Algorithm 3 Greedy

Input: graph: G, size of seed set: k, diffusion model: M, influence function: σ
Output: seed set: S

```
1 begin
2     S ← ∅  for i ← 1 to k do
3         u ← arg max_{u∈V−S} (σ_{G,M}(S ∪ {u}) − σ_{G,M}(S))   S ← S ∪ {u}
```

A.1.1 Simulation-based Approaches

Most simulation-based approaches use Monte Carlo (MC) simulation to estimate it. Kempe et al. [6] first proposed a framework that integrates the MC simulation into the greedy algorithm to solve the classic IM problem. They basically follow the greedy framework shown in Algorithm 3. The difference is that they estimate the influence function using MC simulation. To an influence function $\sigma_{G,M}(U)$, MC simulation is used to simulate the activation process and outputs the estimated number of nodes that can be activated. To each influence function, the MC simulation is run r times and the final results are averaged.

This algorithm has the $1 - \frac{1}{e} - \epsilon$ approximation guarantee and runs in time $\mathcal{O}(\epsilon^{-2}k^3n^2m\log n)$. The approximation ratio is acceptable, however, the running time is too large for practical applications. For a simple real-world dataset, it may take days to get the result. So, many follow-up studies are proposed to reduce running time. There are two main ways to reduce the complexity, which are reducing the usage of MC simulations and reducing the complexity of MC simulation.

Reducing Usage of MC Simulation Leskovec et al. [68] proposed CELF which reduces the usage of MC simulations by designing an early stop mechanism that identifies and prunes unimportant nodes. In an unoptimised MC simulation, each node would need to be traversed, which is very time-consuming. In CELF, each node has an upper bound which is its maximum marginal gain in the past iteration. The upper bound makes it unnecessary to calculate its marginal gain when the maximum marginal gain of an already visited node is already greater than its upper bound. In traversing the nodes, the nodes will be ordered in descending order by their upper bounds, and then the nodes will be traversed in turn. Once this mechanism is triggered, the current iteration will be terminated and the upper bound of each visited node will be updated to the latest marginal gain. This mechanism has greatly improved efficiency, allowing the greedy framework to run 700 times faster. But, it does not improve the worst-case time complexity. On the basis of CELF, Goyal et al. [70] proposed CELF++ which further pruning unimportant nodes by including the maximum marginal gain of the nodes that have been visited before. Such improvements can further increase the execution efficiency of the CELF algorithm by 35-55%.

Although CELF algorithms (including the CELF++ algorithm) can effectively improve the efficiency, their execution efficiency on large-scale networks is still unsatisfactory. In particular, at the initial stage, the CELF algorithm needs to establish an initial upper bound for each node in the social network, and the algorithm can execute very slowly when the number of nodes in the network is high. To speed up the establishment of the upper bound, Zhou et al. [71] proposed UBLF that can quickly establish the upper bound of each node by matrix analysis. Nevertheless, UBLF can only be applied to the IM problem based on IC and LT diffusion models.

Reducing MC Completely Approaches to reduce the complexity of MC simulations are less common and not mainstream. Wang et al. [72] proposed CGA which divides the graph into communities, and then it uses the influence of each

node within its community to determine which nodes are selected as seeds. Jiang et al. [73] proposed a method based on Simulated Annealing (SA) to reduce the complexity of MC simulations.

In summary, the advantage of simulation-based approaches is that the vast majority of them are not specific to a particular diffusion model, as they are based on simulations in a diffusion model. In addition, simulation-based approaches usually can provide any theoretical guarantees. However, the disadvantage is that even with the pruning technique applied, the number of MC simulations that need to be performed is still high and the complexity of the simulations can hardly be reduced [74].

A.1.2 Proxy-based Approaches

To avoid the high computation completely of estimating the influence function using MC simulation, some studies proposed methods that use proxy models to estimate the influence function. According to the survey by Li et al. [1], there are two different types of proxy models. There are two different types of models. The first, which transforms the IM problem into a problem of ranking the influence of users, is called the influence ranking proxy. The second, which simplifies the diffusion model to reduce the complexity of the algorithm according to its characteristics, is called the diffusion model reduction model.

Diffusion Model Reduction Proxy The main idea of the diffusion model reduction proxy is to generate a proxy model to replace the original diffusion model to reduce the complexity of MC simulation. There are two main ways to build a proxy model. There are two main ways to build a proxy model. the first way is to reduce the stochastic diffusion model to a deterministic model. The second way is to limit the range of influence of a single node to a certain number of hops. The reduction of the model is generally performed on the IC and LT models.

The first method proposed to reduce IC model is proposed by Kimura et al. named Shortest-Path Model (SPM) [75]. The core idea of the Shortest-Path Model is that node u only influences node v by the shortest path. In the same paper, they also proposed SP1M, where the restrictions on paths have been relaxed by a jump. Among all the proxy models of IC model, the most well-known one is MIA/PMIA model [76]. MIA limits the diffusion of influence of a node u to a local tree structure rooted at u which changes the diffusion process from random to deterministic which can avoid the MC simulation. The tree construction process is as follows. They first proposed the maximum influence path $MIP(u, v)$ which is the path with the maximum influence probability among all paths from u to v. Then to a node u, both an in-arborescence subgraph ending with u and an out-arborescence subgraph starting from u are built using the Dijkstra shortest-path algorithm. The maximum influence in-arborescence $MIIA(u, \theta)$ containing all MIPs ended with u with propagation probabilities larger than a given threshold θ and the maximum influence out-arborescence $MIOA(u, \theta)$ containing all MIPs started from u with propagation probabilities larger than θ. Based on the pruned trees, all nodes that can be affected by node u can be identified. PMIA improves MIA by updating the arborescence subgraph after adding a node to the seed set

which stops existing seeds from blocking the influence of future seeds. Kim et al. [77] further reduce the IC model using a similar way to MIA named IPA. IPA additionally prunes paths with propagation probabilities less than a given threshold.

Chen et al. [76] proposed a proxy model of LT model named LDAG which is similar to PMIA. Instead of arborescence subgraph, LDAG used directed acyclic graphs (DAGs) to reduce the LT model. Each node is attached to a DAG using the Dijkstra SP algorithm. Similarly, they also set a threshold θ to prune the paths with low probability. The influence of nodes is estimated based on DAGs. However, the computation of DAGs is time-consuming. They propose LDAG at the cost of reliability but with a significant reduction in computational complexity i.e. LDAG is sub-optimal. Goyal et al. [78] proposed SIMPATH algorithm where the influence of the seed set can be estimated by counting paths starting from all the nodes in the seed set which is NP-hard. In contrast to LDAG, they accelerate counting by retaining edges with likelihood below a threshold θ.

The main disadvantage of diffusion model reduction proxy is that methods usually can only be applied to a single diffusion model and have no expand ability, although some methods such as [79] can be applied to both IC and LT models. But the advantage is that they are designed to suit the characteristics of the model making them achieve good performance.

Influence Ranking Proxy The general idea of influence ranking proxy is ranking nodes according to a metric similar to the influence of users. The seed set is generated from the ranking which makes the design of an appropriate metric central to this type of approach.

There are a number of algorithms such as PageRank [80] and LeaderRank [81] that already provide metrics for ranking users in social networks. The design ideas for these metrics, such as centrality, are similar to what is needed in the IM problem. But using these indicators directly would present two main problems [1]. The first is the problem of influence overlap. For example, two nodes both score high in PageRank, but there may be many duplicate users that both nodes can influence. Since the size of the seed set is limited, it is clear that both nodes should not be selected for the seed set. The second is that a higher level of these metrics does not mean a higher level of impact [4, 6].

Therefore, based on these traditional metrics, many studies have proposed new metrics that take influence into account. Chen et al. [4] proposed a degree proxy named DEGDIS. DEGDIS solves the influence overlap problem by reducing the influence of all neighboring nodes of a node with a certain factor when it is selected into the seed set. Such a strategy solves the problem of overlapping influence between neighboring nodes, but the overlap of influence between all non-directly connected nodes remains unaddressed. Liu et al. [82] proposed a quantitative metric, named Group-PageRank, that can efficiently estimate the upper bound of the influence of a group of nodes by extending PageRank from a single node to a group of nodes. They still follow the greedy framework, but instead of using MC simulations to estimate influence, they use their proposed Group-PageRank, which effectively improves the efficiency of their approach. The Group-PageRank only works for IC model. Jung et al. proposed IRIE [83] which derived a system of linear equations with n variables to estimate the influence of each node. There

are many other studies that proposed this kind of proxy, such as SPIN [84] and IMRANK [85] etc.

The biggest problem with influence ranking proxy is that it's hard to completely solve the problem of influence overlap. But such a transformation can reduce the computational complexity to a large extent.

In summary, the proxy-based approaches are effective in reducing the computational complexity and in most cases give acceptable performance results. However, these approaches do not give theoretical guarantees and they mostly work only on specific models and are less scalable.

A.1.3 Sampling-based Approaches

Sampling-based approaches can also be named sketch-based approaches. The general idea of sampling-based approaches is to sample enough sketches (realizations) from the graph and then solve the IM problem based on these sketches. This effectively avoids the considerable time-consuming MC simulations required in simulation-based approaches. Based on the way in which sampling is carried out, such approaches can be divided into two categories, forward influence sampling (FIS) and reverse influence sampling (RIS).

In FIS, the sketch is constructed by extracting a subgraph. The extraction is guided by an instance of the influence process. Then, the influence of a seed set is estimated by these sketches. There are many typical FIS-based approaches such as: NEWGREIC [4], StaticGreedy [86], StaticGreedyDU [86], PRUNEDMC [87], SKIM [88] etc. The efficiency of FIS is significantly higher than that of simulation-based approaches and preserves approximation guarantee. However, the sampling of FIS is based on the whole graph, the time complexity is still too high to run on large graphs. As a result, the FIS is now no longer in use. However, inspired by the FIS, a more efficient RIS were proposed and has successfully replaced the RIS. Therefore, in this subsection the focus will be on the more efficient RIS.

RIS is first proposed by Borgs et al. [89]. The core idea of RIS is random reverse reachable (RR) set. A random RR set is generated by the following ways: (1) randomly select a node $u \in V$, (2) randomly sample a subgraph g from G, (3) the RR set contains all the node that can reach u through a path in g. Roughly speaking, a random RR set contains nodes that can influence u. In application, multiple random RR sets need to be constructed. The more a node appears in these random RR sets, the greater its ability to influence others. For IM problem, the more nodes in a node set appear in random RR sets, the more likely it is the optimal seed set. The general framwwork of RIS-based approaches is shown in Algorithm 4.

Algorithm 4 RIS

Input: graph: G, size of seed set: k, number of random RR sets to generate: θ
Output: seed set: S

```
4 begin
5     S ← ∅  R ← θ random RR sets  for i ← 1 to k do
6         n ← node with the most appearances in random RR sets in R  add n to S
          remove all random RR sets in R that contain n
```

The number of random RR sets to be generated is the key to balancing efficiency and performance. The higher the number, the better the performance but the higher the complexity, and vice versa. Thus, determining the amount of generation becomes critical. Borgs et al. [89] proposed a strategy that keeps generating random RR sets until the number of nodes involved in the generation process is greater than a predefined threshold. To more accurately establish the relationship between the number of random RR sets and approximation loss, Tang et al. [90] directly build the relationship in their method TIM/TIM++ which also follows the RIS framework.

RIS-based approaches Reduced time complexity but increased space complexity [1] because all random RR sets have to be stored in the memory and a large number of samples have to be generated. Wang et al. [91] proposed a lazy sampling technique (BKRIS). They first provide a low bound on the optimal number of random RR sets which leads to a lower θ. Then, they utilize the bottom-K minHash technique and it is not necessary to fully materialize all θ sketches in the process of generating the seed set.

Overall, sampling-based approaches have advantages over both of the approaches already described. It can increase efficiency while having a rigorous approximation guarantee. But there are still drawbacks that cannot be ignored. Firstly, the space complexity is extremely high, requiring the storage of a large number of random RR sets. secondly, the parameter setting is difficult, the number of random RR sets is difficult to determine, and even additional computational complexity is required to determine the number of random RR sets.

A.2 Static Non-Progressive IM Problem

kempe et al. [6] proposed that the non-progressive IC and LT model can be reduced to a progressive model by replicating each node for every timestamp in the time horizon under consideration, and connecting each node to its neighbors in the previous timestamp. This way all the algorithms developed for the progressive model can be used for the non-progressive model. However, replicating a large network for each timestamp over a large time horizon will clearly make this approach impractical for large social networks containing millions of nodes. [8] Therefore, this approach is mainly of theoretical interest.

Devi et al. [69] proposed a model named ICIRS that can capture the non-progressive influence diffusion process and proposed a model named TSGC that shows a good performance on the ICIRS model. TSGC first divide the graph into different non-overlapping communities. Each of the communities is taken as a subgraph by ignoring the connecting links between them. The most influential users are identified by using each node's local and global centrality measures in the given sub-graph of a graph. Finally, the ranking of each node is performed by calculating the $Iscore_p$ of each node within a whole network.

The vast majority of IM-related research has focused on the study of progressive IM problems, and these two methods are the ones that, to the best of my ability, have actually proposed solutions for non-progressive IM problems.

A.3 Dynamic Progressive IM Problem

The dynamic progressive IM problem is first proposed by Aggarwal et al. [10]. They formalized the problem and proposed heuristics to solve the problem. They first propose a method, DynInfluenceVal, to evaluate the influence of a given set of nodes. Then, they proposed forward-influence and backward-influence algorithms to solve the dynamic influence maximization problem. The forward-influence algorithm combines the greedy approach with forward temporal analysis in order to determine the most influential points in the network. It first evaluates the dynamic influence spread from each individual node by using DynInfluenceVal and nodes are sorted. Different combinations of top-ranked nodes are tested using DynInfluenceVal to select the most optimal one. In the inverse algorithm, they turn the problem into an optimization problem to obtain the best combination. Zhuang et al. [9] assume that changes in the graph can only be detected by probing a small number of nodes at regular intervals. So, they split the problem into two parts to solving it. The first construct a subgraph by probing a set of nodes in the underlying graph such that the influence diffusion can be best observed. Then, the seed set that can maximize the influence on the underlying graph is found on the subgraph. These two methods are single-step seeding methods that seed set is decided at one time. Besides it, there is also a sequential seeding strategy. In this strategy, the seeds are decided in the process of the simulation of influence spreading without making any assumption about the spreading process. Michalski et al. [92] proposed a method taking this strategy. They use dynamic graphs in the form of snapshots and one node is determined in each snapshot. In each snapshot, the inactivated node having the highest degree centrality along with nodes that are activated in the last snapshot will be set as seed. An influence-diffusion model will be used to simulate influence diffusion in this snapshot. All the nodes that are activated after the influence diffusion process are still activated in the next snapshot. There are some other works that proposed solutions to the dynamic IM problem, such as [26] and [93], which are all based on traditional methods.