上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

BACHELOR'S THESIS

论文题目：社交网络中的信息传播与预测

学生姓名：胥拿云

学生学号：515030910635

专　　业：计算机科学与技术（致远荣誉计划）

指导教师：高晓泓

学院(系)：致远学院

Submitted in total fulfillment of the requirements for the degree of

Bachelor in Computer Science and Technology

Information Diffusion and Prediction in Social Networks

Nayun Xu

Advisor

Prof. Xiaofeng Gao

ACM Honored Class
Zhiyuan College
Shanghai Jiao Tong University
Shanghai, P.R. China

社交网络中的信息传播与预测

摘要

社交网络对现代社会有巨大的影响，因此社交网络分析(SNA)称为了一个重要的研究课题，其中影响力最大化问题(IM)是一个主要研究对象。影响力最大化问题是解决通过在社交网络中选取 k 个用户作为种子集合，最大化种子集合通过信息传播能够影响到的期望用户数量。由于影响力最大化问题具有巨大的研究价值与应用潜力，该问题已经被国内外学者广泛研究。在本论文中，我们提出了一种基于贪心算法框架，使用强化学习与图嵌入的方法来解决影响力最大化问题。

影响力最大化问题的贪心算法的主要瓶颈在于每次尝试增加节点到当前解中时都要为所有可行的选择计算其影响力增量。我们注意到如果可以将这个时间复杂度极高的影响力计算替换成一种更快的评估函数，贪心算法的效率将大大提高。我们发现这个需要的函数和强化学习中的 Q 学习十分相似。因此我们结合贪心算法的框架与 Q 学习算法来解决影响力最大化问题。为了将复杂的网络结构转化为可以被强化学习算法用来表征“状态”要素的表示，我们使用了网络嵌入的方法来获得低维度的网络表示。

我们用了许多不同的实验来检验评估我们的模型。具体来说，实验包括在相同的网络上使用不同的种子集合大小来评估我们的模型，比较使用不同的训练网络大小所获得的模型的优劣，将我们的模型与现有的两个基准模型在随机生成数据以及现实公开数据集比较等。相较于基准模型，我们的模型取得了不错的实验结果，同时通过分析具体的训练过程，我们发现强化学习中奖励函数的设计以及超参数的设定依然存在不足，这意味着我们的模型在现有结果的基础上依然有进一步的提升空间。

关键词：社交网络分析，影响力最大化，贪心算法，网络嵌入，强化学习

上海交通大学
毕业设计（论文）学术诚信声明

本人郑重声明：所呈交的毕业设计（论文），是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

作者签名：

日期：　　年　　月　　日

上海交通大学
毕业设计（论文）版权使用授权书

本毕业设计（论文）作者同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本毕业设计（论文）的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本毕业设计（论文）。

保密□，在___年解密后适用本授权书。

本论文属于

不保密□。

（请在以上方框内打“√”）

作者签名：　　指导教师签名：

日期：　　年　　月　　日　　日期：　　年　　月　　日

Information Diffusion and Prediction in Social Networks

ABSTRACT

Social networks have great influence on modern society. Thus, Social Network Analysis (SNA) has become a important research topic. The Influence Maximization (IM) problem, which selects k users as a seed set from a social network to maximize the expected number of influenced users, is a major problem in SNA area. Due to its great potential in application and difficulty in algorithm design, IM has been vastly studied by global researchers. In this paper, we propose an algorithm based on the greedy algorithm framework using reinforcement learning and network embedding to solve IM problem.

The bottleneck of the greedy algorithm for IM problem is the evaluation of influence when adding every feasible node. We notice that replacing the time-consuming evaluation with a faster function would significantly improve the effiecncy. We discover that the function we want shares similar properties with the Q-function in reinforcement learning. Therefore, we combine the greedy algorithm structure and Q-learning algorithm to solve the IM problem. In order to convert the comlex network structure to a representation which can be used as 'State' in the reinforcement learning, we employ the network embedding to obtain low-dimensional representation of the network.

We evaluate our methods through various experiments and compare our model with existing works. In particular, we evaluate our model performance on the same

network given different seed set sizes, compare models trained with different training network sizes, and compare our model with two baseline models on generated data and real-world data. Our proposed model achieves a decent result compared to exsiting works and the detailed training process indicates that it still has problems with the reward function and reinforcement learning hyper-paramters, which means our model has space to improve.

Keywords: Social Network Analysis, Influence Maximization, Greedy Algorithm, Network Embedding, Reinforcement Learning

Contents

Chapter 1 Introduction

Nowadays, social networks have become an important component of modern society. Wechat moments, Weibo, Facebook, Instagram and other social media construct an immense social network which contains a great amount of information for analysis. Obviously, the analysis of social network is both difficult and of enourmous potential of application. For example, ghost users who are paid to post fake comments or forward specific posts to create a buzz are frequently seen in weibo. Such phenomenon is annoying and would misguide public opinion. We can detect such ghost users through analyzing the content of their comments, relation pattern, time of posting and forwarding, and etc. For peopole who want to advertise through social media, selecting appropriate users to post their products is significant which can also be addressed through analyzing user relationships and preference. Therefore, Social Network Analysis (SNA) which investigates social structures throught the use of networks and graph theory has become a popular research topic in the recent decade.

SNA has been studied vastly and its major research obejctive concentrates on nodes and their relationships. Challenging problems like comminity detection, structural Balance, influence maximization, network difffusion draw a lot of attention from researchers. As is mentioned above, detecting ghost users can be viewed as an application scenario of community detection, and selecting users to post advertisement can be related to influence maximization. In this paper, we focus on the influence maximization (IM) problem.

IM aims to select k users in a social network as a seed set with the maximized influence which means the expected number influenced users through information diffusion from the seed set. IM has great potential in application, which has been mentioned above, meanwhile, the research of IM is faced with technical challenges. First, the influence calculation relies on information diffusion which is hard to model in a social network. Second, optimally sovling IM has been proven to be

NP-hard in most settings [1–3]. Furthermore, given a seed set, the calculation of influence is complex because the information diffusion is a stochastic process.

Such challenges have driven researchers to develop extensive techniques and algorithms for infuluence maximization. We have surveyed a lot of exsiting work, and notice that most of algorithms for IM probelm can be devided into three categories simulation-based methods, proxy-based methods and sketch-based methods, according to [4]. The explanations of the general ideas and frameworks of the three categories are in Chap. 2.

IM can be viewed as selecting a seed set S on a graph and optimizing the influence function $I(S)$. We discover that it shares some similarities with traditional combinatorial optimization problems on graphs. For example, the Minimal Vertex Cover problem (selecting minimum number of nodes to cover all the edges of a graph) can be viewed as selecting a set of nodes that can cover all the edges, and minimizing the size of this set. The difference is that for traditional combinatorial optimization problems, given a solution, we can calculate the objective function to be optimized accurately within polynomial time, but for IM problem, given a seed set, we cannot calculate the influence easily which makes the problem more challenging. However, inspired by the similarity, we find that some of the ideas of combinatorial optimization algorithms on graphs can be adopted to solve IM problems. Hanjun Dai, Elias B. Khalil, Yuyu Zhang et al.(2017) [5] proposed a method using reinforcement learning [6] to solve combinatorial optimization problems on graph. Our idea is to adopt their work to the IM problem and modify the algorithm to tackle the challenges within IM mentioned above.

Our algorithm use a simulation-based method to measure the influence given a seed set. During reinforcement learning process, we improve the measuring method to be incremental to lower the overall complexity of measuring influence during a complete learning process. The model is trained under many small graphs to improve its training efficiency. Furthermore, we feed different graph sizes and seed set sizes with an ergodic grid search manner to ensure good generalization ability.

Chapter 2 Related Work

In this chapter, we give general explanations to three major categories of methods on influence maximization problem, and briefly introduce two important techniques used in our proposed model.

2.1 Simulation-Based IM Method

We explain the influnece maximization algorithms which belong to the simulation-based category. The method of this category estimates influence function using *Monte Carlo* (MC) simulations. It combines the MC simulation with a greedy algorithm framework. There are many different algorithms that belong to this category, and we only introduce the basic framework. Kempe el al. proposed the seminal work on the simulation-based method in [1]. It iteratively selects a node to add into the seed set if the node maximizes the marginal reward. Furthermore, it estimates influence for each user set formed by the current seed set and a new node added. The esitmation is averaged simulation result of several rounds.

In [1] and many following works [7], they do not theoretically analyze the number of rounds to run the simulation and state a emprical large number which will result in a small enough error. Therefore, some works put their attention on reducing the number of simulations. CELF [8] proposed a method to estimate the upper bound of influence so as to dismiss the simulation of sets with minor influences. CELF adopts the power law principle: most nodes do not provide significant influences and can be ignored in subsequent simulations. CELF++ [9] further prunes unnecessary simulations to improve CELF.

There are other works, for example Wang et al. [10] focus on reducing the simulation complexity using the idea of devide-and-conquer.

2.2 Proxy-Based IM Method

We explain the influence maximization algorithms which belong to the proxy-based category. The method of this category estimates influence spread of the seed set through proxy models. Different to the simulation-based method, the proxy-based method does not involve burdensome simulation calculations and therefore makes IM algorithms more efficient to be adopted on larger graphs. However, most proxy-based algorithms are designed for a specific diffusion model and will make use of the corresponding properties to speed up the estimation of the influence. The proxy-based algorithms usually do not have guarantees in theory, but they provide significant performance improvement over the simulation-based algorithms. Nevertheless, the solution quality of proxy-based models is competitive with those given by simulation-based algorithms.

Many of the proxy-based algorithms follow a intuitive idea: influence ranking, which ranks all users in the network according to some metric and selects seed set by the ranking directly. The metrics used in the influence ranking should approximate the true influence of each user. Therefore, the problem becomes finding good enough metrics for the ranking.

Proxy-based algorithms can either use simple ranking proxies like degree, PageRank [11], Distance Centrality [12] or other advanced ranking proxies like influence-aware ranking proxy and self-consistent ranking.

Simple ranking proxies are intuitive but have some drawbacks which may prevent the algorithm from getting fair solutions. First, such ranking proxies are not designed for influence while they may have some latent connections with influence [1, 13]. Second, they do not take influence overlap into consideration, which means they view and calculate the ranking metric of each user separately and consider the overall influence as the linear combination of each single user. If the users in the graph have significant influence overlap, such simple ranking proxies will easily overestimate the influence.

Some influence-aware ranking proxies take the drawbacks of simple ranking

proxies into account. They either use a discount proxy for influence estimation [13]. or modify the simple proxy like PageRank [14,15]. The idea of [13] is an improved version of degree proxy. The major modification is that, after selecting a user into the seed set, the proxy score of the selected user's neighbours will be discounted to make up for the influence overlaps. However, it only consider the direct overlap between neighbours, and ignore the overlaps between users connected by indirect diffusion paths. The idea of [14] is to generalize the PageRank proxy to consider a set of users instead of a single user.

There are many other proxy-based algorithms view the problem from a different angle. They aim to simplify the diffusion process thus to address the hardness of influence evaluation. There are two major ideas in general: (1) reduce the stochastic process into a deterministic process, and (2) limitting the diffusion range of each user to a small subgraph. After simplifying the diffusion process, they employ the simple greedy algorithm framework to derive a solution [16–18].

2.3 Sketch-Based IM Method

We explain the influence maximization algorithms which belong to the sketch-based category. The method of this category aims to address the bottleneck of simulation-based algorithms, i.e. the heavy simulation calculations for influence estimation. Sketch-based algorithms usually have better theoretical efficiency without loss of the approximation bound. The main idea of sketch-based algorithms is to construct some sketches for the target network, and estimate the influence on the constructed sketches. Such constructed sketches are designed for some specific diffusion model, and sketch-based algorithms address the bottleneck of simulation-based algorithms at the cost of losing generalization ability for different diffusion models.

Compared to the simulation-based method and the proxy-based method, the sketch-based method is less intuitive and involve more complex algorithms and theoretical techniques. There are different ways to construct sketches for a graph based on a specific diffusion model. Based on the flow of considering the influence,

sketch construction can be divided into two categories, forward-sketch algrotihms and reverse-reachable-sketch algorithms.

Forward-sketch algorithms consider the influence in a traditonal way, i.e. measuring the influence through influence diffusion from the seed set. Considering the influence diffusion process, forward-sketch algorithms construct a sketch using subgraphs derived by the specific diffusion model.

A simple way to construct sketches based on the independent cascade diffusion model is to remove each edge with a probabilty of failure to cascade through this edge, and add the subgraph to a sketch set. After constructing a number of sketches, the influence can be estimtated through averaging the number of users that can be reached from the seed set in each constructed sketches. The approximation rate of the estimation is related to the size of the sketch set.

NewGreIc [13], StaticGreedy [19], PrunedMC [20], SKIM [21] and many other works fall into the category of forward-sketch algorithms. Although such algorithms have good theoretical effiency and at the same time have fair approximation guarantee, when they encounter the worst case, the time complexity is still beyond the computational capacity on large graphs.

Reverse-reachable-sketch algorithms consider the influence in a reverse manner, i.e. measuring the influence by observing the portion of users that can be reached by the seed set. For a randomly selected user, reverse-reachable-sketch algorithms generate a reverse-reachable set which contains the users that can have influence on that selected user. If a user has great influence on other users, it is highly possible that the user will appear in such randomly generated reverse-reachable sets. At the same time, a good seed set should cover as many as possible of the reverse-reachable sets.

TIM [22], IMM [23], BKRIS [24] and some other works fall into the category of reverse-reachable-sketch algorithms. In general, the reverse-reachable-sketch algortihms perform better in efficiency and complexity compared to forward-sketch algorithms.

2.4 Network Embedding

Nowadays, information network has witnessed explosive increment. There could be billions of nodes and edges in an information network, which results in that launching complex inference or algorithms on the entire network would be faced with a lot of difficulties. Therefore, researchers proposed the Network Emebedding method to address this problem. The key idea of the network embedding method is to find a function to map the nodes in the network to lower dimensional representations.

Generally, the network embedding method has several characteristics: adaptability, scalability, community aware, low dimensional and continuous. The networks in our real life are developing rapidly, which requires algorithms to have the abiility to quickly adapt to new data without rerunning overhead computations. Most of the networks that have great potential for research and application are large-scale networks, and graph embedding methods should be efficient to deal with such networks. The distance between the embedded representations re used to measure the similarity between the corresponding nodes, which requires the generalization for homogenous networks. In real-life application, we don't always have enough labelled data, so low-dimensional models which can converge fast are preferred. The continuous representations can help researchers make better use of the embedding result for classification and model partial membership.

In traditonal view, network embedding is considered as a process to reduce the dimension of a network, and there are some mathematical techniques, for example principal component analysis (PCA) [6] and factorization-based methods [25]. In the early stage of network embedding research, most of the algorithms can be modeled to find a matrix representation of size $n \times k$ to represent the original $n \times m$ graph ($k \ll n$).

As the deep learning becomes the most popular technique both in academic research and industrial application, network embedding methods combining deep learning comes into life. Deepwalk [26] is the first network embedding algorithm

using deep learning. Deepwalk views the nodes as 'words' and uses short random walk to generate 'sentences' to fill the gap between network embedding and word embedding. Then techniques in language models like skip-gram can be applied to the generated 'sentences' and get the network embedding results.

The advantages of Deepwalk can be concluded as following:

- The random walk 'sentence' can be generated as demanded.
- The skip-gram model is optimized for each sample generated by random walk. Thus Deepwalk becomes an online algorithm.
- Deepwalk is scalable, because generating random walk 'sentence' and optimizing the skip-gram model can be done in parallel.

Deepwalk belongs to unsupervised network embedding methods. There are many other works using an unsupervised learning framework.

LINE [27] uses breadth first search to generate context nodes: only the nodes within two steps from the selected node can be viewed as its neighbour. Besides, LINE uses negative sampling to optimize the skip-gram model.

Node2Vec [28] is an extended version of Deepwalk, it combines both breadth-first-search style and the depth-first-search style for neighborhood searching.

Walklets [29] discovers the bias of Deepwalk and propose learning multiscale network embedding to address the bias. The complexity of computing a sample is the square of the network size, and Walklet approximate the sample by skipping some nodes during the random walk.

GraRep [30] applies SVD to powers of the adjacent matrice so as to obtain low-dimensional representation of nodes.

GraphAttention [31] proposed an attention-based model which can learn multi-scale representations and predict the connections in the original netowork. GraphAttention does not set hyper-parameter to control the distribution of nodes in the context, it learns the attention automatically.

SDNE [32] learns node level representations using the deep auto encoder model.

It preserve the distance of 2-step neighbors and further maintain the distance through minimizing the Euclid distance between the representations.

DNGR is another network embedding method based on deep neural networks. It employs random surfing strategy to capture structual information from the graph. It further converts the structural information to PPMI matrices to embed network nodes.

The unsupervised network embedding methods mentioned above only take network structural information as input to obatin low-dimensional representations. However, the real-life networks usually contain addtional information on nodes and edges, which is called attribute. In Wechat Moments, the contents of Moments posted by users are useful for analysis, and many algorithms require such information as input. To deal with such demand, we expect network embedding methods can take the attribute into account.

TADW [33] put attention to the situation where text information is attached to network nodes. It employs a matrix factorization based method.

CENE [34] simulates the network structure and text information at the same time. CENE views the text information as special nodes. It uses node-wise connection and attribute connection to do embedding.

Maxmargin Deepwalk [35] is a semi-supervised method. It learns the representation from partial-labelled network.

2.5 Reinforcement Learning

As machine learning becomes a popular research topic because of the deep neural networks [36], reinforcement learning [6], an important branch of machine learning also draws a lot of attention and has achieved great success. The essence of reinforcement learning is decision making. Reinforcement learning aims to train an agent that can automatically and continuously make decisions. The problem reinforcement learning aims to address is universal, which makes reinforcement learning have great application potential in robitics, optimal control, chess game, game strategy, traffic control and many other areas.

Reinforcement learning has four key elements: agent, state, action and reward. The target of reinforcement learning is to achieve as much accumulated reward as possible. Taking kids learning to walk as an example, consider the kid as the agent, he needs to do actions from a action space like step left or step right, every time before he makes an action, he can see the state, i.e. the location and his own pose, and every time after he makes an action, he would fall (0 reward) or not fall (positive reward). His goal is to keep walking with the least times of falling, which equals to maximize the accumulated reward.

Different from supervised learning, reinforcement learning aims to maximize an accumulated reward along a decision process, so that there are no labels for each action. The agent learns the decision making policy through seeing the reward is good or not. Furthermore, in reinforcement learning settings, agent sometimes does not see the reward immediately after it makes an action, which could make the agent update more complicated.

Reinforcement learning can be divided into three major categories according to the key elements:

- Pocliy based: focus on finding the best policy.
- Value based: focus on finding the best total reward.
- Action based: focuse on finding the best action in each step.

There can be more detailed divisions according to different criteria:

- According to modeling environment

  Model-free: do not model the environment, make actions according to the feedback from the environment.

  Model-based: model the environment, use the model to predict all possible feedbacks and pick a best action according to the model.

- According to updating rule

Monte-carlo update: for each episode, update the agent when the episode ends.

Temporal-difference update: for each step, update the agent when it makes an action.

- According to the agent

  On-policy: agent learns through acting and iteracting with the environment itself.

  Off-policy: agent learns through observing other agents' learning history.

There are many existing works on reinforcement learning, and the major frameworks are as followed:

- SARSA: makes use of Markov property, only use the information of the next step, explores the actions and feedbacks and updates state value at each step. S means state, A means action, R means reward, the second S means the next state, the second A means the next action.
- Q-learning: the framework is similar to SARSA. The key difference is that it aims to learn the q-function which estimates the score of each action.
- Policy Gradients: instead of using the current reward to update agent, policy gradients methods use a discounted future reward.
- Actor-Critic: the framework contains two parts, Actor and Critic. Actor updates the policy, Critic updates the value.
- Monte-carlo learning: after generating a complete state-action-reward sequence, update the agent according to states.
- Deep-Q Network: the key idea is experience replay which stores the exploration results and updates the neural network parameters through random sampling.

Chapter 3 Problem Description and Preliminary

In this chapter, we present a detailed explanation of the problem studied in this paper.

3.1 The Influence Maximization Problem

This problem is first raised by Domimgos and Richardson. They modeled the problem as a Markov random field and solve the problem using heuristic algorithm. Kempe et al. [1] designed a greedy algorithm to iteratively add the node with the maximum marginal gain. The greedy algorithm framework has great influence on the following works.

Given a social network, we can view it as a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. In social network setting, users are the nodes, and user relationships (follow, friends) are edges. The influence maximization problem is to find a seed set of size k with the maximum influence.

The influence of a seed set relies on the information diffusion process. For example, forwarding a friend's Wechat moment is a kind of information diffusion. However, how to model the diffusion process is complicate. As in the example, how to give estimation to whether a user would forward a specific Wechat moment is hard to define. Therefore, researchers have proposed some diffusion models to restrict the diffusion process, which may not be realistic but helpful for academic research.

Definition 3.1 (**Diffusion Model & Influence**). *Given a social network $G = (V, E)$, a seed set $S \subseteq V$ and a diffusion model M models the process of information spread from S. The influence function of S defined as $I_{G,M}(S)$ is the expected number of nodes influenced by the seed set S, where $I_{G,M}(S)$ is a non-negative set function defined on any subset of nodes.*

Definition 3.2 (**Influence Maximization**). *Given a social network $G = (V, E)$,*

a diffusion model M and a positive integer k, influence maximization problem aims to select a set S^* of k nodes from V as the seed set to maximize the influence function $I_{G,M}(S^*)$, i.e.,

$$S^* = \underset{S \subseteq V \bigwedge |S| \leq k}{\arg\max} I_{G,M}(S). \tag{3-1}$$

Obviously, the diffusion model M will have a significant impact on the influence function $I(\cdot)$. Therefore defining diffusion models is necessary before digging into the influence maximization problem,

3.2 Diffusion Models

There are many different diffusion models that formulate the diffusion process. We show several diffusion models that are usually used in the influence maximization problem: the independent cascade (IC) model, the linear threshold (LT) model, the triggering (TR) model, time-aware diffusion models and non-progressive diffusion models.

Before introducing these different diffusion models, there is a common framework that works for each of the following diffusion models. The diffusion process is based on a graph $G = (V, E)$. First, we define the state of each node as active or inactive. Denote the state of node v as $s(v)$, v is active if $s(v) = 1$ and v is inactive if $s(v) = 0$. At the beginning, a seed set S is selected and it contains k nodes $\{v_1, v_2, \cdots, v_k\}$. Initially, the nodes in S are active, and other nodes are inactive. Then, the active nodes can 'influence' their neighbours to be active, and the newly activated nodes can further activate their neighbours until no new nodes can be activated. The framework usually considers the 'activate' process as a stochastic process, and the influence function $I(S)$ is the expected number of active nodes when the diffusion process terminate. In most of the influence maximization research, the diffusion models are progressive, which means active nodes cannot be activated again.

3.2.1 The Independent Cascade (IC) Model

In the independent cascade model, the graph is defined as $G = (V, E, P)$ where a influence probability set P is introduced. For each edge $e_{=}(u, v)$, there is a influence probability $p_{u,v}$ which represents the probablity that u can activate v. Node v can be activated by all its active incoming neighbours independently by the influence probablity. Under this setting, the influence diffusion process can be unfolded step by step. Considering the newly activated nodes in step 0 are the nodes in the seed set S, in step i (≥ 1) the newly activated nodes in step $i-1$ will try to activate their outgoing neighbours that are inactive by the influence probability P. The termination condition of the diffusion process is that there are no newly activated nodes. The influence function $I(S)$ is the expected number of active nodes in G with seed set S when the diffusion process terminates.

The independent cascade model is a proposed model which does not reflect the real-world diffusion process, and in real-world data, there is no given influence probability. Therefore, if the algorithm needs to directly get the influence probability, we need to assign the influence probabilty to each edge. A commonly-used method is to to assign $p_{u,v}$ on edge $e = (u, v)$ as $\frac{1}{d_v^{in}}$, where d_v^{in} is the in-degree of v.

Some recent studies view the influence probability in another way. They try to learn the influence probablity from data like the propagation actions in social networks [37–39].

3.2.2 The Linear Threshold (LT) Model

In the linear threshold model, the graph is defined as $G = (V, E, W, T)$, where W is the edge weight set, and T is the threshold set. Each edge $e = (u, v) \in E$ is associated with a weight $w_{u,v}$. Denoting $N_{in}(v)$ as the set of incoming neighbours of node v, W satisfies that

$$\sum_{u \in Nin(v)} w_{u,v} \leq 1 \tag{3-2}$$

Each node v is associated with a threshold t_v and t_v is sampled uniformly at random from $[0, 1]$. Under this setting, the influence diffusion process can be unfolded step by step. Considering the newly activated nodes in step 0 are the nodes in the seed set S, in step i $(i \geq 1)$, for each inactive node v, denoting the set of newly active incoming neighours of v in step $i-1$ as $A_{in}(v) \subseteq N_{in}(v)$, v is activated when

$$\sum_{u \in A_{in}(v)} w_{u,v} \geq t_v \tag{3-3}$$

The termination condition of the diffusion process is that there are no newly activated nodes. The influence function $I(S)$ is the expected number of active nodes in G with seed set S when the diffusion process terminates.

Most influence maximization algorithms use heuristics to assign the weight $w_{u,v}$ for each edge $e = (u, v) \in E$, for example, uniformly assigning a probability from the set $\{0.1, 0.01, 0.001\}$ at random or using a similar method to what is introduced in Sec. 3.2.1.

So far, there is no existing work on data-driven approach to assign threshold for the linear threshold model.

3.2.3 The Triggering (TR) Model

The triggering model is proposed by Kempe et al. [1]. The purpose of proposing this model is to generalize the independent cascade model and the linear threshold model.

In the triggering model, for node v, there is a distribution from which we can draw a subset T_v of v's neighbours. The distribution means the likelihood that a subset of v's neighbours can influence v. Under this setting, the influence diffusion process ca be unfolded step by step. Considering the newly acitvated nodes in step 0 are the nodes in the seed set S, in step i, for each inactive node v, a triggering set T_v is randomly drawn from the distribution, v is activated if T_v contains at least one node that is newly activated in step $i-1$. The diffusion process terminates when there is no newly activated node. Similar to the aforementioned two diffusion

models, the influence function $I(S)$ is the expected number of active nodes in G with seed set S when the diffusion process terminates. It has been proved that both the independent cascade model and the linear threshold model are special cases of the triggering model [1].

There are other models that extend the independent cascade model and the linear threshold model in a more generalized manner than the triggering model. For example, [40] extends the independent cascade model to a decreasing cascade (DC) model. The decreasing cascade model defines the influence probability from node w to node v given subset S of acitve neighbours of v as $p_v(w, S)$ where the decreasing cascade model enforces

$$p_v(w, S) \geq p_v(w, T), S \subseteq T \tag{3-4}$$

Kempe el al. [1] propose a general threshold (GT) model that extends the independent cascade model and the linear threshold model. The general threshold model defines the threshold function of node v to be $f_v(S)$ where S is the set of active neighbours of node v . When $f_v(S)$ is larger than the threshold value t_v, v is activated. Although the general threshold model does not approximate the influence maximization problem in the most general case, [41] gives a proof that the influence spread of the general threshold model shares the similar funtional properties as those of the independent cascade model, the linear thershold model, and the triggering model.

3.2.4 Time-Aware Diffusion Models

The independent cascade model, the linear threshold model, and the triggering model do not take 'time' into consideration. Their influence diffusion termination condition is that there is no newly activated node. However, to view these models, in the aforementioned diffusion process explanations, they can be unfolded into steps. It reminds us that in some cases, the influence diffusion process can be time-critical and may have constraints and limitations on time. To address the

problem in such situations, the time-aware models are proposed. There are mainly two categories of algorithms of time-aware models:

- The discrete-time models in which the influence diffusion process only happens in discrete time steps.
- The continuous-time models in which the influence diffusion process happens in continuous time.

The discrete-time models model the influence diffusion process between neighbours as a random variable which is discrete over the time steps. They can been seen as extensive models of the independent cascade model. Furthermore, as mentioned in Sec. 3.2.1 and Sec. 3.2.2, the discrete-time models share the same property that influence diffusion can be unfolded to discrete steps as the independent cascade model and the linear threshold model.

In realistic situations, for example, forwarding Wechat moments, the action of forwarding happens continuously in time. Although we can devide the time to several slots and make it similar to discrete steps, the inherent property would be corrupted by this kind of modifications. In order to address the problems in the continuous-time situation, the continuous-time models are proposed in [42, 43]. The continuous-time independent cascade models suppose the likelihood of influence propagation between neighbours can be seen as a continuous distribution over continuous time. To be specific, if node u is activated at time t_u, and u is trying to activate its inactive neighbour v, the probability that u can successfully activate v at time t_v ($t_v > t_u$) is defined as a conditional probability:

$$p(t_v|t_u; a_{u,v})$$

where $a_{u,v}$ is the parameter that defines the continuous influence distrubution over time. $a_{u,v}$ can be considered as the influence strength propagting from node u to node v. In the setting of continuous-time models, the termination condition takes the time into account, where a termination time $T > 0$ is pre-defined. The

continuous-time diffusion process terminates when there is no newly activated node before the termination time T. There are different ways to model the distribution. A simple and commonly-use distribution model is the exponential model, i.e.,

$$p(t_v|t_u; a_{u,v}) = a_{u,v} \cdots e^{-a_{u,v}(t_v - t_u)}, t_v > t_u \tag{3-5}$$

$$p(t_v|t_u; a_{u,v}) = 0, t_v \leq t_u \tag{3-6}$$

There is another commonly-used distribution model proposed in [43] called DynaDiffuse which supposes the probablity the influence diffusion would happen decreases exponentially over time. If node u is activated at time t_u and consider the edge $e = (u, v)$ with the influence strength $a_{u,v}$, the influence diffusion probability from u to v at time t_v $(t_v > t_u)$ is

$$p(t_v|t_u; a_{u,v}) = 1 - e^{a_{u,v} \cdot (t_v - t_u)} \tag{3-7}$$

Similarly, the DynaDiffuse model also has the termination time constraint $T > 0$. Obviously, we can see the similarity between the DynaDiffuse model and the aforementioned continuouse-time diffusion model.

Chapter 4 Proposed Model

In this chapter, we describe our proposed model through separating the algorithm into several parts and explain each part in detail.

4.1 Model Overview

Our proposed model is based on a greedy algorithm framework for the influence maximization problem. Inspired by [5], we notice that the setting of reinforcement learning on solving combinatorial optimization problems over graphs shares a similar decision process as the greedy framework on the influence maximization problem, where we view the already selected nodes and the graph structure as the state, selecting the next seed node as the action, and the increment of the influence function after adding the newly selected node into the seed set as the reward. Therefore, our model is to apply the reinforcement learning algorithm to the traditional greedy algorithm framework for the influence maximization problem.

To apply the reinforcement learning algorithm, we need to first find appropriate representations for the graph. The original graph representation of adjacent matrix is not suitable as the input for the reinforcement learning agent. Therefore, we use the network embedding to convert the original graph structure to low-dimensional node level representations and feed the embedded representations to the reinforcement learning agent. In the reinforcement learning part, we apply the Q-learning and Deep-Q network as the training framework.

This whole algorithm framework is similar to the proposed model in [5], and our main contribution is to generalize the framework to the complex setting where the algorithm termination condition is dynamic and reward calculation is hard and time-consuming.

To be specific, this framework has already been applied to some combinatorial optimization problems like the minimal vertex cover (MVC) problem and the

traveling salesman problem (TSP). In such settings the algorithms have clear termination conditions, which can be that the selected nodes can cover all the edges (in the minimal vertex cover problem) or that the selected node sequence contains each node exactly once (in the travelling salesman problem), and the reward (or the score we can give to the solution with which we can evaluate how good the solution is) can be -1 everytime a new node is selected (in the minimal vertex cover problem) or minus the increment of the total distance of the node sequence (in the travelling salesman problem), which are both easy for calculation.

However, under the setting of the influence maximization problem, the termination condition is related to the given size k of seed set S which can be different for the same network. For example, given a same graph $G = (V, E)$ where $|V| = 100$, then $k = 5$ or $k = 20$ will obviously make a big difference for the selecting strategy. Furthermore, when we have a partial solution S' where $|S'| < k$, and we are going to select node v to add to the solution, then the reward should be

$$I(S' \cup u) - I(S')$$

which is #P-hard. To address the dynamic termination problem, we propose a training strategy which employs a grid search manner to improve the generalization ability of the agent for different seed set size k. To address the calculation of influence function $I(\cdot)$, we use a simulation-based method to estimate the function. The aforementioned difficulties and our solutions will be further explained in Sec. 4.6.

4.2 The Greedy Algorithm Framework

We present the greedy algorithm framework for the influence maximization problem in this section.

Before introducing the greedy algorithm framework, the approximability of the algorithms are guaranteed by the properties of the influence function $I(\cdot)$, monotonicity and submodularity which are defined below.

Definition 4.1 (Monotonicity)**.** *Monotonity means that adding an element v to S will never decrease the function value. Formally, a monotune function $f(\cdot)$ should satisfy the constraint:*

$$f(S \cup \{v\}) \geq f(S) \tag{4-1}$$

Definition 4.2 (Submodularity)**.** *Submodular functions are used to describe the phenomena of diminishing marginal return which broadly exists in the real world. The diminishing marginal return means that the marginal gain of adding an element v to S will be greater or equal to the marginal gain of adding v to S' where S' is a superset of S. Formally, a submodular function $f(\cdot)$ should satisfy the constraint:*

$$f(S \cup \{u\}) - f(S) \geq f(S' \cup \{u\}) - f(S') \tag{4-2}$$

where

$$S \subseteq S'$$

There are many diffusion models which are introduced in Sec. 3.2, and the influence functions are based on the specific diffusion model. Based on the commonly-used diffusion models, i.e., the independent cascade model, the linear threshold model, the triggering model and continuous-time models, the influence functions are submodular and monotone.

Theorem 4.3. *The influence functions under the independent cascade model, the linear threshold model, the triggering model and continuous-time models are monotone and submodular.*

The detailed proof of Thm. 4.3 is given in [1].

The general idea of the greedy algorithm is to select k nodes as the seed set through k round iterations, in each iteration, the algorithm selects the node with the highest marginal gain. To describe the algorithm clearly, we define some symbols:

- $G = (V, E)$ is the network the algorithm is running on.

- S_i is the partial solution of the seed set in round i.
- $I(\cdot)$ is the influecne function defined on any subset of G.
- $I(v|S_i) = I(S_i \cup \{v\}) - I(S_i)$ represents that the marginal gain by adding v to S_i.

The algorithm has k rounds of iterations. At the begining, initial $S_0 = \emptyset$. During iteration $i \geq 1$, compute the marginal gain $I(v|S_{i-1})$ for each $v \in V$, select the node with the maximal marginal gain $v = \arg\max_{v \in V \setminus S_{i-1}} I(v|S_{i-1})$ and add v to the partial solution $S_i = S_{i-1} \cup \{v\}$. After k round iterations, S_k is the solution. The pseudo code of the algorithm is shown below in Alg. 4.1.

Algorithm 4.1: Greedy algorithm for the influence maximization problem

```
INPUT: G = (V, E), k
Initialize S_0 = ∅
for i = 1, ··· , k do
    for v ∈ V \ S_{i−1} do
        Compute I(v|S_{i−1})
    v* = arg max_{v∈V\S_{i−1}} I(v|S_{i−1})
    S_i = S_{i−1} ∪ {v*}
OUTPUT: S_k
```

This greedy algorithm can achieve an approximation ratio of

$$1 - 1/e - \epsilon$$

The ϵ is caused by the #P-hardness of the $I(\cdot)$ calculation. $I(\cdot)$ usually is estimated by a sampling algorithm, and ϵ stands for the sampling error. Although the algorithm have the best approximation ratio so far, we need to compute the marginal gain of each node which brings heavy computation, which makes it impossible to be applied on large networks.

We employ this framework, and aim to improve the efficiency by taking advantage of Q-learning in the reinforcement learning to learn a function that is similar to the marginal gain in this algorithm.

4.3 Network Embedding

In order to convert a comlex network to a low-dimensional latent representation to enable the reinforcement learning agent to receive as input, we need to employ the network embedding techniques. There are many methods to embed a network including traditional mathematical methods like matrix factorization and the modern deep learning based methods. According to our survey, deep learning based methods outperform the traditional embedding methods in most cases. Therefore, we choose to use a deep learning based network embedding method. We surveyed and implemented several network embedding models: DeepWalk [26], Node2Vec [28], Struc2Vec [44], and Structure2Vec [45].

4.3.1 DeepWalk

The goal of DeepWalk is to learn a low-dimensional representation $X_E \in \mathbb{R}^{|V|\times d}$of a partially labeled graph $G = (V, E, X, Y)$, where $X \in \mathbb{R}^{|V|\times S}$ is the attribute set, and $Y \in \mathbb{R}^{|V|\times|y|}$ is the set of labels. The motivation is that in social networks, the links between the attribute vectors violate the i.i.d. assumption which makes the problem not suitable for the traditional classification methods which try to learn a mapping from X to Y. Therefore, the latent representation which contain the structural information of the network learnt by DeepWalk is helpful in such situation.

The major contribution of DeepWalk is that

- The groundbreaking work to apply deep learning to graph analysis and construct a robust representation which can be used in statistical models.
- It performs well in the situation where labeled data is scarce which is frequently seen in real world data.
- The framework is extendable to large data through streaming and is able to run in parallel.

The main idea of DeepWalk is to use random walk on the network to generate 'sentences' and apply the neural language model SkipGram on the generated 'sentences' to obtain the low-dimensional latent representation of the graph. The reason why neural language models can be applied to the network representation is because both words in natural languages and vertices appearing in the random walk on social networks follow the power-law distribution.

The algorithm consists of two main components: a random walk generator and an update process. The random walk generator samples a node v uniformly from the graph $G = (V, E)$, and generate a random walk starting from v. In each step of the random walk, a node u is uniformly sampled in random from the neighbours of the current node. There is a constraint t on the random walk length which means the a random walk instance contains at most t steps. The random walk from node v is denoted as $W_v = (W_v^1, W_v^2, W_v^2, \cdots, W_v^t)$. In practice, the algorithm generates γ random walk instances from each node v. Then W_v is to update the representation, the model that converts W_v to a low-dimensional latent representation is SkipGram. The core of the algorithm is described in Alg. 4.2.

Algorithm 4.2: DeepWalk

INPUT: $G = (V, E), k$
    SkipGram window size w
    embedding dimension d
    number of random walk instances per node γ
    length of random walk instances$t$
Initialize Φ
Build a binary tree T from V
foreach $i = 1, \cdots, \gamma$ **do**
    $O = \text{Shuffle}(V)$
    foreach $v_i \in O$ **do**
        $W_{v_i} = RandomWalk(G, v_i, t)$
        SkipGram(Φ, W_{v_i}, w)
OUTPUT: matrix of the node representations $\Phi \in \mathbb{R}^{|V| \times d}$

In the SkipGram part, the language model learns the low-dimensional latent representation using the random walk instances starting from each node. Given a sequence of random walk $(v_1, v_2, v_3, \cdots, v_{i-1})$, the likelihood of oberserving v_i is

estimated as

$$Pr(v_i|(v_1, v_2, v_3, \cdots, v_{i-1}))$$

The goal of the language model is to learn a latent representation for each node, i.e. a mapping function Φ mapping $v \in V$ to $\mathbb{R}^{|V|\times d}$. Therefore, the problem can be converted to estimate the likelihood:

$$Pr(v_i|(\Phi(v_1), \Phi(v_2), \cdots, \Phi(v_{i-1})))$$

If the walk length is large, the computation of the likelihood would be not affordable. To make the computation feasible, we need to view the problem from a different angle

- Predict the context given a word (node) instead of predicting a word (node) given a context.
- The context is composed of the words (nodes) to the both sides of the given word (node).
- Ignore the contraint of the order.

Then we can get optimize the mapping function Φ through optimzing the log-likelihood

$$\underset{\Phi}{\text{minimize}} -\log Pr(\{v_{i-w}, \cdots, v_{i+w}\} \setminus v_i|\Phi(v_i)) \tag{4-3}$$

Therefore, the SkipGram part of the DeepWalk algorithm can be described as Alg. 4.3.

Algorithm 4.3: SkipGram

```
INPUT: Φ, W_{v_i}, w
foreach v_j ∈ W_{v_i} do
    foreach u_k ∈ W_{v_i}[j − w : j + w] do
        J(Φ) = − log Pr(u_k|Φ(v_j))
        Φ = Φ − α · ∂J/∂Φ
```

In practice, the DeepWalk algorithm uses hierachical softamx to compute the likelihood. The $Pr(u_k|\Phi(v_j))$ in Alg. 4.3 is not feasible for calculation because the graph can be extremely large and the cause the computation too expensive. Therefore, the algorithm uses a binary tree to optimize the computation complexity. The algorithm uses the numebr of occurences of each node as the weight, and build a Huffman tree. Then the prediction problem becomes maximizing the likelihood of a specific path on the Huffman tree, i.e.,

$$Pr(u_k|\Phi(v_j)) = \prod_{l=1}^{\lceil |V| \rceil} Pr(b_l|\Phi(v_j)) \tag{4-4}$$

where b is the path from root v_j to leaf u_k.

Since many of the social network datasets are very big, and in real-world application, companies are usually faced with dynamic networks which can develop over time. An interesting and useful variant of the DeepWalk model is the streaming approach which can allow the algorithm take a part of the network as input and update the representation directly.

4.3.2 Node2Vec

Node2Vec aims to propose a flexible representation learning algorithm. It can convert the nodes which belong to the same community to similar embeddings. Furthermore, the nodes with similar functions (have similar structure links in the corresponding community) will obtain similar embeddings.

Node2Vec follows the idea of Word2Vec in the natural language processing area. It learns the embedding representation through maximizing the likelihood of preserving network neighborhoods of nodes in a d-dimensional feature space which shares a similar to the DeepWalk algorithm. The difference is that Node2Vec considers the communiity of each node instead of a path starting from each node. Node2Vec employs second-order random walk to generate the community of each node.

Obviously, how to define the community is crucial to this algorithm. Node2Vec

defines series of biased random walk to explore different communities of each node. Therefore, the algorithm is highly flexible and its paramters are not fixed. The parameters can be learned through semi-supervised learning.

Node2Vec employs both depth-first search and breadth-first search to explore the neighborhood of each node. It shares many similarities with the DeepWalk algorithm, and it can be seen as a extended DeepWalk which combines the depth-first search and breadth-first search based random walk.

Given graph $G = (V, E)$, denote $f(v)$ is the mapping function which converts u to a low-dimensional vector. For each $v \in V$, denote $N_S(v)$ as the neighborhood set of v sampled under sampling strategy S.

The optimization objective of Node2Vec is to maximize the log probability of the neighborhood given a specific node, i.e.,

$$\max_f \sum_{u \in V} \log Pr(N_s(u)|f(u)) \tag{4-5}$$

To make the optimization objective feasible, Node2Vec propose two assumptions:

- **Conditional independence**. Supposing the source node is given as u, the probability any node occurs in the neighborhood set $N_S(u)$ is independent from the rest nodes in $N_S(u)$, i.e.,

$$Pr(N_s(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u)) \tag{4-6}$$

- **Symmetry in feature space**. When node v is a source node, $f(v)$ is the same as when v is a neighborhood node. Then the conditional likelihood can be written as a softmax version

$$Pr(n_i|f(v)) = \frac{exp(f(n_i) \cdot f(v)}{\sum_{u \in v} exp(f(u) \cdot f(v))} \tag{4-7}$$

Under these two assumptions, the final version of the objective function be-

comes

$$\max_{f} \sum_{v \in V} \left[-\log Z_v + \sum_{n_i \in N_S(v)} f(n_i) \cdot f(v) \right] \tag{4-8}$$

Where $Z_v = \sum_{u \in V} exp(f(v) \cdot f(u))$ is the normalizer and is too expensive to compute on large networks. Therefore, the algorithm approximate it with negative sampling.

The sampling strategy of Node2Vec is still random walk, which is similar to the DeepWalk algorithm. However, the random walk of Node2Vec introduces bias.

Given the source node v, the sampling algorithm aims to sample a random walk of a fixed length l. Denote $c = (c_1, c_2, \cdots, c_l)$ as the sequence of the random walk ($c_1 = v$). The sampling process works under the distribution:

$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

where π_{vx} is the unormalized transition probability between v and x, and Z is the normalizer.

Node2Vec introduces two hyper-parameters p and q to control the strategy of the random walk. Supposing the current random walk has just stepped from t to v through edge (t, v) , and now it needs to decide the next step. Denote $\pi_{vx} = \alpha_{pq}(t, x) \cdot w_{vx}$, where w_{vx} is the edge weight of (v, x), let

$$\alpha_{pq}(t, x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$

d_{tx} is the distance of the shortest path from t to x.

Under the setting of α_{pq}, the hyper-parameters p and q will influence the random walk strategy in the following way

- The influence of p: p serves as the return parameter which controls the

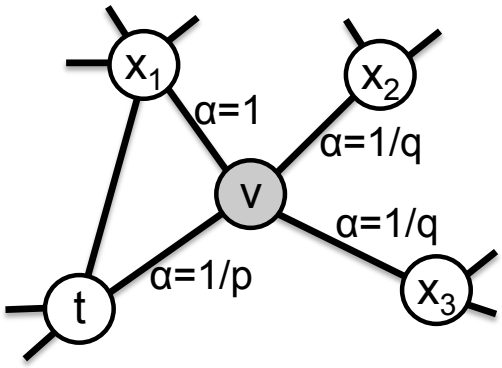

Figure 4.1 Illustration of a single step of the random walk instance [28]

probability that the random walk will visit the node which has just been visited. Under the aforementioned t, v. x setting, it controls the probablity $t = x$.

- The influence of q: q serves as the In-out parameter which controls whether the random walk steps outside or inside. If $q > 1$ the random walk tends to visit nodes closer to t (closer to breadth-first search). If $q < 1$, the random walk tends to visit nodes distant from t (closer to depth-first search).

Fig. 4.1 illustrates a single step of a random walk instance.

After the random walk, the rest part is similar to the DeepWalk algorithm. The entire algorithm structure is described in Alg. 4.4 and Alg. 4.5.

Algorithm 4.4: Node2Vec

INPUT: Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q
π = CalculateWeights(G, p, q)
$G' = (V, E, \pi)$
Initialize *walks* to Empty
for *iter* $= 1$ ***to*** r **do**
  forall the *nodes* $u \in V$ **do**
    walk = node2vecWalk(G', u, l)
    Append *walk* to *walks*
f = StochasticGradientDescent(k, d, *walks*)
OUTPUT: f

Algorithm 4.5: node2vecWalk

INPUT: Graph $G' = (V, E, \pi)$, Start node u, Length l
for $i = 1$ ***to*** l **do**
  $curr = walk[-1]$
  V_{curr} =GetNeighbors$(curr, G')$
  $s = Sample(V_{curr}, \pi)$
  Append s to $walk$
OUTPUT: $walk$

4.3.3 Struct2Vec

Both DeepWalk and Node2Vec use random walk to explore the neighborhood and optimize the representation through likelihood maximization. The neighborhood exploration strategy introduces their common assumption: nodes with short distance have higher similarity. However, in many cases two distant nodes can have high similarity which cannot be captured by the aforementioned embedding methods.

For example, in the network shown in Fig. 4.2 node u and node v have high similarity in structure, although they are distant.

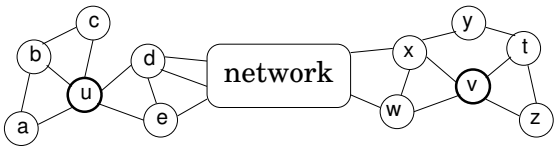

Figure 4.2 Illustration of a network with two distant but similar nodes [44]

Struc2Vec proposes a new criteria to measure the structural similarity between two nodes. Given a network $G = (V, E)$, and $v \in V$, denote $R_k(v)$ as the set of nodes with distance exactly k from node v, and $R_1(v)$ is the direct neighbour set of v. Let $s(S)$ denote the ordered degree sequence of a node set $S \subseteq V$. Let $f_k(u, v)$ denote the structural distance between node u and node v when considering their neighborhood within distance k. Then

$$f_k(u, v) = f_{k-1}(u, v) + g(s(R_k(u)), s(R_k(v))) \tag{4-9}$$

where $k \geq 0$ and $|R_k(u)|, |R_k(v)| > 0$. $g(D_1, D_2) \geq 0$ is a funtion which measures the distance between two ordered sequence D_1 and D_2. Because $s(R_k(u))$ and $s(R_k(v))$ may have different lengths and duplicated elements, Struc2Vec employs Dynamic Time Warping (DTW) to measure the distance. The dynamic time warping method computes the distance between sequence Q and sequence C following the dynamic programming function:

$$\gamma(i,j) = d(q_i, c_j) + \min\{\gamma(i-1, j-1), \gamma(i-j, j), \gamma(i, j-1)\} \tag{4-10}$$

The distance between Q and C is $\gamma(|Q|, |C|)$ Based on this function, the distance function between two element is defined as

$$d(a,b) = \frac{\max(a,b)}{\min(a,b)} - 1 \tag{4-11}$$

$d(a,b)$ makes 1 and 2 are more distant than 101 and 102.

Based on the definition of structural distance $f(\cdot,\cdot)$, Struc2Vec constructs a multi-layer weighted graph M for the context generating (random walk). In layer k, the edge weight of (u,v) is

$$w_k(u,v) = e^{-f_k(u,v)}$$

Between the adjacent layers, there are edges connecting them:

$$w(u_k, u_{k+1}) = \log(\Gamma_k(u) + e) \tag{4-12}$$

$$w(u_k, u_{k-1}) = 1 \tag{4-13}$$

where

$$\Gamma_k(u) = \sum_{v \in V} \mathbb{1}(w_k(u,v) > \bar{w_k}) \tag{4-14}$$

As in the aforementioned network embedding algorihtms, Struc2Vec uses random walks to generate sequences as the context of a given node for representation

learning. Consider a random walk on the constructed multilayer network M, it will decide to stay in the current layer or step to the adjacent layers. The probability of staying the current layer is q.

If the random walk decides to stay in the current layer, the probability of stepping from u to v in layer k is:

$$p_k(u,v) = \frac{e^{-f_k(u,v)}}{Z_k(u)} \tag{4-15}$$

where $Z_k(u)$ denotes the normalization factor for node u in layer k which is

$$Z_k(u) = \sum_{u \in V \bigwedge v \neq u} e^{-f_k(u,v)} \tag{4-16}$$

It is easy to find from the eqautions that the random walk would step to nodes with shorter structural distance to the current node. Thus, the context generated by the random walk will contain more structurally similar nodes, independent from the locations on the original graph G.

The random walk would also choose to enter adjacent layers with a probability of $1-q$, the choice of layer is decided by

$$p_k(u_k, u_{k+1}) = \frac{w(u_k, u_{k+1})}{w(u_k, u_{k+1}) + w(u_k, u_{k-1})} \tag{4-17}$$

$$p_k(u_k, u_{k-1})) = 1 - p_k(u_k, u_{k+1}) \tag{4-18}$$

For each node u, the random walk starts at layer 0 u_0, and will terminate after a relatively small number of steps, and will repeat several times.

Struc2Vec also uses the SkipGram and hierarchical softmax which was adopted in both of the aforementioned netowrk embedding methods to learn the languange model.

4.3.4 Structure2Vec

Different from the aforementioned unsupervised network embedding methods, the Structure2Vec method is a supervised method. It takes networks as input and optimize the learned latent represention through optimizing a mapping from networks to their labels . That is, the input data is defined as

$$D = (X_n, Y_n)_{n=1}^{N}$$

where X_n is a network and Y_n is the corresponding label which can be a real number for regression tasks or discrete label for classification tasks. The latent representation learning method used by Structure2Vec is pairwise Markov random field on the hidden variables H_n and the networks $X_n(V_n, E_n)$.

The embedding of μ_v will be updated over iterations t for all $v \in V$ as

$$\mu_v^{(t+1)} = F(\left\{\mu_u^{(t)}\right\}_{u \in N(v)}, \{\omega(v,u)\}_{u \in N(v)}) \tag{4-19}$$

where $N(v)$ is the set of neighbours of the node v, and F is a non-linear mapping such as a neural network or a kernel function.

4.4 Reinforcement Learning

Reinforcement learning (RL) is the area of machine learning that deals with sequential decision-making. In this section, we describe how the RL problem can be formalized as an agent that has to make decisions in an environment to optimize a given notion of cumulative rewards. It will become clear that this formalization applies to a wide variety of tasks and captures many essential features of artificial intelligence such as a sense of cause and effect as well as a sense of uncertainty and nondeterminism.

A key aspect of reinforcement learning is that an agent learns a good behavior. This means that it modifies or acquires new behaviors and skills incrementally.

Another important aspect of reinforcement learning is that it uses trial-and-error experience (as opposed to e.g., dynamic programming that assumes full knowledge of the environment a priori). Thus, the reinforcement learning agent does not require complete knowledge or control of the environment; it only needs to be able to interact with the environment and collect information. In an offline setting, the experience is acquired a priori, then it is used as a batch for learning (hence the offline setting is also called batch RL). This is in contrast to the online setting where data becomes available in a sequential order and is used to progressively update the behavior of the agent. In both cases, the core learning algorithms are essentially the same but the main difference is that in an online setting, the agent can influence how it gathers experience so that it is the most useful for learning. This is an additional challenge mainly because the agent has to deal with the exploration/exploitation dilemma while learning.

But learning in the online setting can also be an advantage since the agent is able to gather information specifically on the most interesting part of the environment.

For that reason, even when the environment is fully known, RL approaches may provide the most computationally efficient approach in practice as compared to some dynamic programming methods that would be inefficient due to this lack of specificity.

4.4.1 General Framework

The general RL problem is formalized as a discrete time stochastic control process where an agent interacts with its environment in the following way: the agent starts, in a given state within its environment $s_0 \in \mathcal{S}$, by gathering an initial observation $\omega_0 \in \Omega$. At each time step t, the agent has to take an action $a_t \in \mathcal{A}$. As illustrated in Fig. 4.3, it follows three consequences: (1) the agent obtains a reward $r_t \in \mathcal{R}$, (2) the state transitions to $s_{t+1} \in \mathcal{S}$, and (3) the agent obtains an observation $\omega_{t+1} \in \Omega$. Comprehensive treatment of reinforcement learning fundamentals are provided by [6].

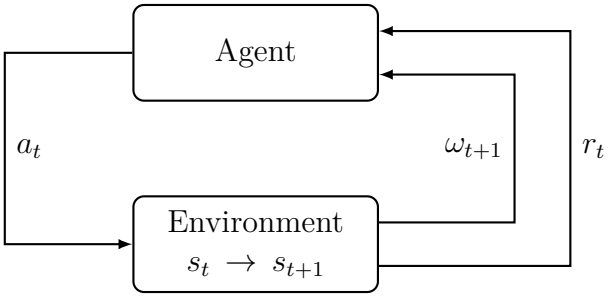

Figure 4.3 Agent-environment interaction in RL [46]

Definition 4.4. *A discrete time stochastic control process is Markovian (i.e., it has the Markov property) if*

- $\mathbb{P}(\omega_{t+1} \mid \omega_t, a_t) = \mathbb{P}(\omega_{t+1} \mid \omega_t, a_t, \ldots, , \omega_0, a_0)$, *and*
- $\mathbb{P}(r_t \mid \omega_t, a_t) = \mathbb{P}(r_t \mid \omega_t, a_t, \ldots, , \omega_0, a_0)$.

The Markov property means that the future of the process only depends on the current observation, and the agent has no interest in looking at the full history.

A Markov Decision Process (MDP) is a discrete time stochastic control process defined as follows:

Definition 4.5. *An MDP is a 5-tuple* $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ *where:*

- $\mathcal{S}$ *is the state space,*
- $\mathcal{A}$ *is the action space,*
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ *is the transition function (set of conditional transition probabilities between states),*
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ *is the reward function, where* $\mathcal{R}$ *is a continuous set of possible rewards in a range* $R_{max} \in \mathbb{R}^+$ *(e.g.,* $[0, R_{max}]$*),*
- $\gamma \in [0, 1)$ *is the discount factor.*

The system is fully observable in an MDP, which means that the observation is the same as the state of the environment: $\omega_t = s_t$. At each time step t, the probability of moving to s_{t+1} is given by the state transition function $T(s_t, a_t, s_{t+1})$

and the reward is given by a bounded reward function $R(s_t, a_t, s_{t+1}) \in \mathcal{R}$. This is illustrated in Fig. 4.4.

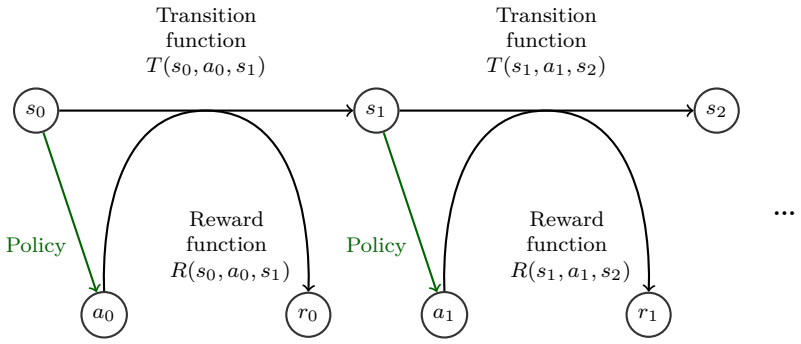

Figure 4.4 Illustration of a MDP [46]

A policy defines how an agent selects actions. Policies can be categorized under the criterion of being either stationary or non-stationary. A non-stationary policy depends on the time-step and is useful for the finite-horizon context where the cumulative rewards that the agent seeks to optimize are limited to a finite number of future time steps.

Policies can also be categorized under a second criterion of being either deterministic or stochastic:

- In the deterministic case, the policy is described by $\pi(s) : \mathcal{S} \to \mathcal{A}$.
- In the stochastic case, the policy is described by $\pi(s, a) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ where $\pi(s, a)$ denotes the probability that action a may be chosen in state s.

We consider the case of an RL agent whose goal is to find a policy $\pi(s, a) \in \Pi$, so as to optimize an expected return $V^\pi(s) : \mathcal{S} \to \mathbb{R}$ (also called V-value function) such that

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, \pi\right], \tag{4-20}$$

where:

- $r_t = \underset{a \sim \pi(s_t, \cdot)}{\mathbb{E}} R\big(s_t, a, s_{t+1}\big)$,

- $\mathbb{P}\big(s_{t+1}|s_t, a_t\big) = T(s_t, a_t, s_{t+1})$ with $a_t \sim \pi(s_t, \cdot)$,

From the definition of the expected return, the optimal expected return can be defined as:

$$V^*(s) = \max_{\pi \in \Pi} V^\pi(s). \tag{4-21}$$

In addition to the V-value function, a few other functions of interest can be introduced. The Q-value function $Q^\pi(s, a) : \mathcal{S} \times A \to \mathbb{R}$ is defined as follows:

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a, \pi\right]. \tag{4-22}$$

This equation can be rewritten recursively in the case of an MDP using Bellman's equation:

$$Q^\pi(s, a) = \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma Q^\pi(s', a = \pi(s'))\right). \tag{4-23}$$

Similarly to the V-value function, the optimal Q-value function $Q^*(s, a)$ can also be defined as

$$Q^*(s, a) = \max_{\pi \in \Pi} Q^\pi(s, a). \tag{4-24}$$

The particularity of the Q-value function as compared to the V-value function is that the optimal policy can be obtained directly from $Q^*(s, a)$:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s, a). \tag{4-25}$$

The optimal V-value function $V^*(s)$ is the expected discounted reward when in a given state s while following the policy π^* thereafter. The optimal Q-value $Q^*(s, a)$ is the expected discounted return when in a given state s and for a given action a while following the policy π^* thereafter.

4.4.2 Q-Learning

The basic version of Q-learning keeps a lookup table of values $Q(s, a)$ (Eq. 4-22) with one entry for every state-action pair. In order to learn the optimal Q-value function, the Q-learning algorithm makes use of the Bellman equation for the Q-value function whose unique solution is $Q^*(s, a)$:

$$Q^*(s, a) = (\mathcal{B}Q^*)(s, a), \tag{4-26}$$

where $\mathcal{B}$ is the Bellman operator mapping any function $K : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ into another function $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and is defined as follows:

$$(\mathcal{B}K)(s, a) = \sum_{s' \in S} T(s, a, s') \left(R(s, a, s') + \gamma \max_{a' \in \mathcal{A}} K(s', a') \right). \tag{4-27}$$

By Banach's theorem, the fixed point of the Bellman operator $\mathcal{B}$ exists since it is a contraction mapping.

In practice, one general proof of convergence to the optimal value function is available under the conditions that:

- the state-action pairs are represented discretely, and
- all actions are repeatedly sampled in all states (which ensures sufficient exploration, hence not requiring access to the transition model).

This simple setting is often inapplicable due to the high-dimensional (possibly continuous) state-action space. In that context, a parameterized value function $Q(s, a; \theta)$ is needed, where θ refers to some parameters that define the Q-values.

4.4.3 Fitted Q-Learning

Experiences are gathered in a given dataset D in the form of tuples $< s, a, r, s' >$ where the state at the next time-step s' is drawn from $T(s, a, \cdot)$ and the reward r is given by $R(s, a, s')$. In fitted Q-learning [47], the algorithm starts with some

random initialization of the Q-values $Q(s, a; \theta_0)$ where θ_0 refers to the initial parameters (usually such that the initial Q-values should be relatively close to 0 so as to avoid slow learning). Then, an approximation of the Q-values at the k^{th} iteration $Q(s, a; \theta_k)$ is updated towards the target value

$$Y_k^Q = r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta_k), \tag{4-28}$$

where θ_k refers to some parameters that define the Q-values at the k^{th} iteration.

In neural fitted Q-learning (NFQ) [48], the state can be provided as an input to the Q-network and a different output is given for each of the possible actions. This provides an efficient structure that has the advantage of obtaining the computation of $\max_{a' \in A} Q(s', a'; \theta_k)$ in a single forward pass in the neural network for a given s'. The Q-values are parameterized with a neural network $Q(s, a; \theta_k)$ where the parameters θ_k are updated by stochastic gradient descent (or a variant) by minimizing the square loss:

$$\mathrm{L}_{NFQ} = \left(Q(s, a; \theta_k) - Y_k^Q \right)^2. \tag{4-29}$$

Thus, the Q-learning update amounts in updating the parameters:

$$\theta_{k+1} = \theta_k + \alpha \left(Y_k^Q - Q(s, a; \theta_k) \right) \nabla_{\theta_k} Q(s, a; \theta_k), \tag{4-30}$$

where α is a scalar step size called the learning rate. Note that using the square loss is not arbitrary. Indeed, it ensures that $Q(s, a; \theta_k)$ should tend without bias to the expected value of the random variable Y_k^Q

Hence, it ensures that $Q(s, a; \theta_k)$ should tend to $Q^*(s, a)$ after many iterations in the hypothesis that the neural network is well-suited for the task and that the experience gathered in the dataset D is sufficient.

When updating the weights, one also changes the target. Due to the generalization and extrapolation abilities of neural networks, this approach can build large errors at different places in the state-action space. Therefore, the contraction

mapping property of the Bellman operator in Eq. 4-27 is not enough to guarantee convergence. It is verified experimentally that these errors may propagate with this update rule and, as a consequence, convergence may be slow or even unstable.

Another related damaging side-effect of using function approximators is the fact that Q-values tend to be overestimated due to the max operator. Because of the instabilities and the risk of overestimation, specific care has be taken to ensure proper learning.

4.4.4 Deep-Q Networks

Leveraging ideas from NFQ, the deep Q-network (DQN) algorithm introduced by [49] is able to obtain strong performance in an online setting for a variety of ATARI games, directly by learning from the pixels. It uses two heuristics to limit the instabilities:

- The target Q-network in Eq. 4-28 is replaced by $Q(s', a'; \theta_k^-)$ where its parameters θ_k^- are updated only every $C \in \mathbb{N}$ iterations with the following assignment: $\theta_k^- = \theta_k$. This prevents the instabilities to propagate quickly and it reduces the risk of divergence as the target values Y_k^Q are kept fixed for C iterations. The idea of target networks can be seen as an instantiation of fitted Q-learning, where each period between target network updates corresponds to a single fitted Q-iteration.

- In an online setting, the replay memory keeps all information for the last $N_{\text{replay}} \in \mathbb{N}$ time steps, where the experience is collected by following an ϵ-greedy policy. The updates are then made on a set of tuples $< s, a, r, s' >$ (called mini-batch) selected randomly within the replay memory. This technique allows for updates that cover a wide range of the state-action space. In addition, one mini-batch update has less variance compared to a single tuple update. Consequently, it provides the possibility to make a larger update of the parameters, while having an efficient parallelization of the algorithm.

A sketch of the algorithm is given in Fig. 4.5.

In addition to the target Q-network and the replay memory, DQN uses other important heuristics. To keep the target values in a reasonable scale and to ensure proper learning in practice, rewards are clipped between -1 and +1. Clipping the rewards limits the scale of the error derivatives and makes it easier to use the same learning rate across multiple games (however, it introduces a bias). In games where the player has multiple lives, one trick is also to associate a terminal state to the loss of a life such that the agent avoids these terminal states (in a terminal state the discount factor is set to 0).

In DQN, many deep learning specific techniques are also used. In particular, a preprocessing step of the inputs is used to reduce the input dimensionality, to normalize inputs (it scales pixels value into [-1,1]) and to deal with some specificities of the task. In addition, convolutional layers are used for the first layers of the neural network function approximator and the optimization is performed using a variant of stochastic gradient descent called RMSprop [36].

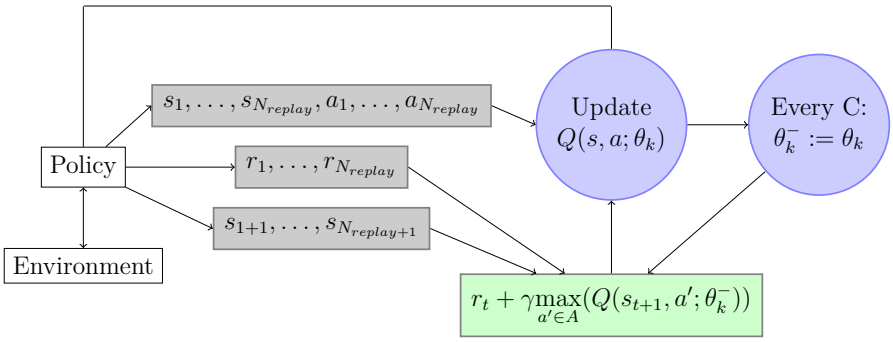

Figure 4.5 Sketch of the DQN algorithm [46]

4.5 Algorithm Pipeline

After introducing the greedy framework and two major components of our proposed model, we show the pipeline of our method in this section.

The core of our proposed model is a Deep-Q network based reinforcement learning agent which takes a greedy strategy to add nodes into the seed set.

Denoting the set we have selected is S, as mentioned in the Sec. 4.2, the motivation of our model is to replace the heavy computation of marginal gain

regarding node v and current solution S to an evaluation function $Q(S, v, G)$ which can be learned with reinforcement learning methods. Therefore the greedy algorithm can be modified as Alg. 4.6.

Algorithm 4.6: Modified greedy algorithm with Q function

INPUT: $G = (V, E), k, Q(S, v, G)$
Initialize $S_0 = \emptyset$
for $i = 1, \cdots, k$ **do**
  $v* = \arg\max_{v \in V \setminus S_{i-1}} Q(S_{i-1}, v, G)$
  $S_i = S_{i-1} \cup \{v*\}$
OUTPUT: S_k

Therefore, our goal becomes learning the Q function. Obviously, the Q function naturally fits the reinforcement learning formulation in Sec. 4.4. Since in the reinforcement learning setting, we need to formulate *state* and the $Q(S, v, G)$ function, which are all related to the network structure, we need to convert the network into low-dimensional representations. The network embedding method introduced in Sec. 4.3 can be used on this task. There are several embeddings we can use, and different embedding methods will make the formulation of the reinforcement learning different because of the supervised or unsupervised and online updated or offline updated nature. However, the different choice of the network embedding method will not affect the general framework of the reinforcement learning.

Denoting the embedding of node v is μ_v, the formulation of the reinforcement learning is:

- **State**: A state should contain the information of the network $G = (V, E)$ and the information of the current selected set S. The choice of the network embedding method makes a difference in how to represent the S and the structure of G at the same time. If the embeddings are obtained by DeepWalk, Node2Vec or Struc2Vec, the state can be repsented as

$$s = \sum_{v \in V \setminus S} \mu_v \tag{4-31}$$

because these methods give fixed embeddings which will not change as S changes. If the embedding are obtained by Structure2Vec, the state can be represented as

$$s = \sum_{v \in V} \mu_v \tag{4-32}$$

Structure2Vec can update the embeddings iteratively as the S changes by adding a signal vector denoting whether each node is in S to the input of the $F()$ function mentioned in Sec. 4.3.4. Through the iterative update, the embedding μ contains the information of the current seed set S.

- **Transition**: The transition is also different according to the embedding method. Given the node to be selected is v, if the embedding method is DeepWalk, Node2Vec or Struc2Vec, the transition is to minus μ_v from the current s, and if the embedding method is Structure2Vec, the transition is to update the μ with new S and recompute the state s.

- **Action**: The action is simple in this setting - the node v to be selected.

- **Reward**: The reward function r is defined on the current state (seed set) S the action v, the new state $s' = (s, v)$

$$r(s, v) = I(s', G) - I(s, G) \tag{4-33}$$

where I is the influence function. Therefore the cumulative reward of the termination state will be equal to the influence of the final solution.

- **Policy**: The policy is a deterministic greedy policy based on $Q(S, v, G)$ function

$$\pi(v|S) = \arg\max_{v' \in V \setminus S} Q(S, v', G) \tag{4-34}$$

Based on the formulation, we can derive the reinforcement learning algorithm with the knowledge of Q-learning in Sec. 4.4.2. The algorithm is shown in Alg. 4.7.

Algorithm 4.7: Q-learning for the Greedy Algorithm

Initialize experience replay memory M to capacity N
for ***episode*** $e = 1$ ***to*** L **do**
  Draw graph G from distribution $\mathbb{D}$ Initialize the state to empty $S_1 = ()$
  for ***step*** $t = 1$ ***to*** T **do**
    $v_t = \begin{cases} \text{random node } v \in V \setminus S_t, & \text{w.p. } \epsilon \\ \arg\max_{v \in V \setminus S_t} Q(S_t, v, G), & \text{otherwise} \end{cases}$
    Add v_t to partial solution: $S_{t+1} = (S_t, v_t)$
    if $t \geq n$ **then**
      Add tuple $(S_{t-n}, v_{t-n}, R_{t-n,t}, S_t)$ to M
      Sample random batch from $B \overset{iid.}{\sim} M$
      Update Q by SGD over B

The entire pipeline of our proposed model can be described as Fig. 4.6.

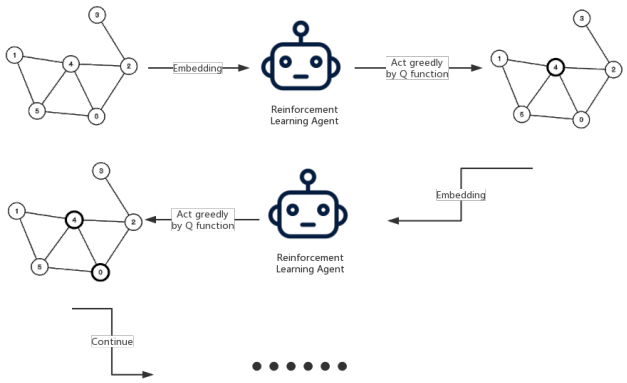

Figure 4.6 Pipeline of the proposed model

4.6 Obstacles and Implementation Details

During the algorithm design and experiments we have come across some obstacles

- The choice of the network embedding method. It non-trivial to decide which network embedding method would perform the best in our proposed model.
- The computation of the reward function which is defined in Eq. 4-33 is time

consuming.

- The computation of the reward function which is defined in Eq. 4-33 requires multiple rounds of simulations, the different simulation results will affect the state of the reinforcement learning.

- The size of seed set S is flexibile and has significant impact on the strategy of the agent. Fixed size during training may results in the loss of generalization ability, while dynamic size may results in the instability of the training process.

To choose a suitable network embedding method, we applied each on them to launch experiments on small datasets for several iterations, and pick the method which performs the best. Finally, we choose the Structure2Vec as the network embedding method. It makes much sense that it outperforms the other methods because it embeds the structural similarity which cannot be captured by DeepWalk and Node2Vec, and it can be updated during the training of the reinforcement learning and adapt to the current solution, which cannot be done by the Struc2Vec method.

The algorithm we used in the proposed model to estimate the influence of a seed set is a simulation-based method described in Alg. 4.8. Everytime the agent adds a node into the current solution, it needs to run this algorithm to compute the reward, which is time-consuming. We improve the efficiency of the computation by breaking the second loop into each action, i.e., the agent does not go through all the nodes inside the active set, instead it only consider the newly added node and the nodes acitvated because of the current action.

Note that if we use the modified algorithm, we need to keep track of T along each episode of the reinforcement learning, which makes the outter loop of M simulations not suitable for this case. We solve this problem by extending T to M sets, and we can keep track of M sets of active nodes, which can be considered as we put the outter loop outside each episode, or we runes M agent at the same time and combine their single-round estimation results together to get the reward.

Algorithm 4.8: The influence estimation algorithm

INPUT: $G = (V, E), S, p, M$
Initialize the active set $T = S$
Initialize the total influence $I = 0$
for *iter = 1* ***to*** M **do**
  foreach v *in* T **do**
    foreach u *in* $G(v) \setminus T$ **do**
      if *RandomSample(p) = True* **then**
        Insert u into T
  $I = I + |T|$
OUTPUT: I/M

To address the problem caused by dynamic seed set size, we design a training strategy to iteratively feed different seed set size to the agent each time the agent finishes the training under the previous seed set size. This strategy is similar to the learning rate scheduling and fine tuning method in deep learning research.

Chapter 5 Performance Evaluation

In this chapter, we show the expriemnts and the results our model achieves and compare our model to some baseline models for evaluation. We choose the independent cascade model to launch our experiments. Our proposed method can be applied to any of the diffusion models because our reward computation is reward based and the q-learning method does not rely on any property of the diffusion model. However, due to the time cost of the training time, we only evaluate the performance of our proposed model under the independent cascade diffusion model.

5.1 Training Setup

We train the reinforcement learning agent on generated small graphs. It is shown in [5] that this training method can train an agent with the generalization ablity on large graphs. We choose the **Barabasi Albert** algorithm proposed by [50] to generate radom graphs. The Barabasi Albert algorithm generates the graph through incrementally adding new nodes into the current graph, the probability that a new node would be connected with node i is

$$p_i = \frac{k_i}{\sum_j k_j} \tag{5-1}$$

where k_i is the degree of node i. An example of a graph generated by Barabasi Albert algorithm is shown in Fig. 5.1.

We train the agent with different training graph sizes respectively, and evaluate them with the same data.

During the training, we set the seed set size k as $\alpha \cdot |G|$, and go over different α iteratively to train the agent, because the generated graphs have different size,

[1]Fig. 5.1 is from GeeksforGeeks https://www.geeksforgeeks.org/barabasi-albert-graph-scale-free-models/

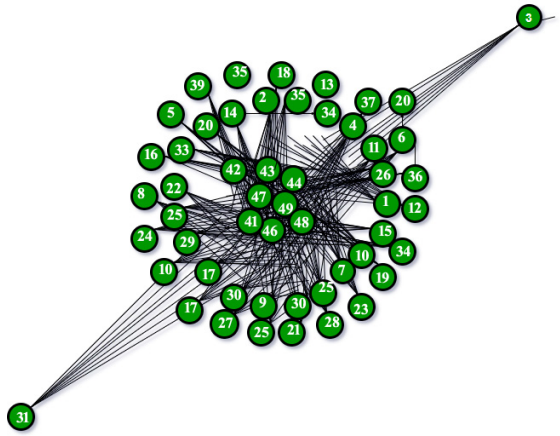

Figure 5.1 Illustration of a graph generated by Barabasi Albert algorithm [1]

which makes it not suitable to set some fixed value as k.

5.2 Datasets

We evaluate our model on two public datasets and generated graphs. The two public datasets are the citation networks of arxiv High Energy Physics – Theory section and arxiv Physics section. We evaluate the performance of our proposed model on the same graph with different seed set sizes.

The citation network of arxiv High Energy Physics – Theory section has 15233 nodes and 31398 edges. The citation network of arxiv Physics section has 37149 nodes and 174161 edges.

5.3 Baseline Models

We choose two baseline models - Degree Discount algorithm [51] and a martingale approach proposed in [52].

5.4 Results

5.4.1 The Training Process

We train the agent using generated training graphs with different sizes. In this section, we show the iteration-reward plot which describes the training process. In

practice, we train three models with training graph size of [40, 50], [80, 100], [1000, 1000] (It means that the graph size is randomly sampled within the interval). The result is shown in Fig. 5.2, 5.3, 5.4.

Figure 5.2 Iteration-reward plot of training size [40, 50]

Figure 5.3 Iteration-reward plot of training size [80, 100]

In general, the agent converges to a fair cumulative reward quickly and struggles around the reward and improves slowly. Specially, in the process of the agent using the largest training data size, it goes through a sudden drop during the training.

We modified the parameter and the training strategy several times, and this is the best result we have got.

One possible reason that would account for this problem is the third obstacle we have mentioned in Sec. 4.6. The proposed reward computation method in Sec. 4.6 does not solve this problem thoroughly.

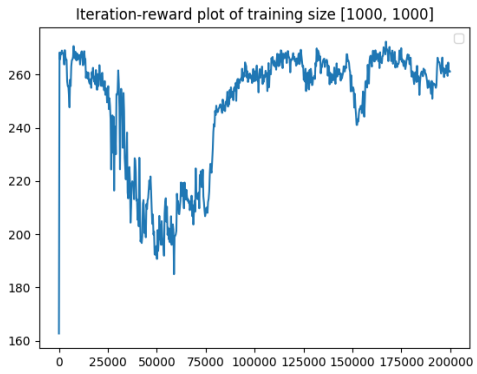

Figure 5.4 Iteration-reward plot of training size [1000, 1000]

5.4.2 The Comparisons over Training Graph Sizes

We compare the three trained models by evaluating their performance on the same dataset. The dataset we use in this comparison is the generated Barabasi Albert graphs with 1000 nodes and 3984 edges. We compare the three trained models denoted as [40, 50] model , [80, 100] model and [1000, 1000] model on the dataset over different seed set size k. (the termination condition). The result is shown in Tab. 5.1 and Fig. 5.5.

	$k = 5$	$k = 10$	$k = 20$	$k = 40$	$k = 80$	$k = 160$
[40, 50] model	182	215	226	272	314	394
[80, 100] model	217	223	236	269	319	396
[1000, 1000] model	198	227	246	271	315	392

Table 5.1 The influence evaluation of different models over different seed set size.

We can see from the result that different training data sizes do not influence the agent performance on large networks, which indicates that the agent has strong generalization ability. Therefore, we can always train the agent with small graphs to save the training time.

5.4.3 Comparisons with Basline Models

In this section, we show the comparisons between the solutions given by our proposed model and those given by baseline models.

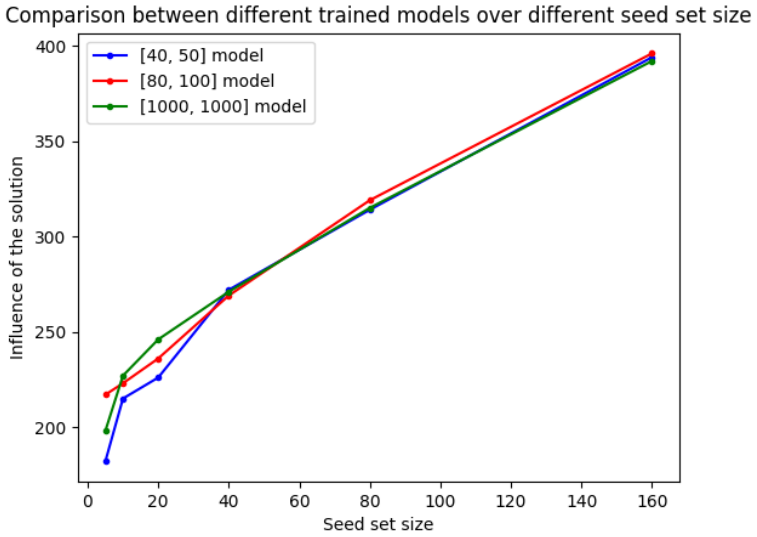

Figure 5.5 Comparison between different trained models over different seed set size

Beacause the model trained with different training data sizes have similar performance on large graphs, we choose the [80-100] model as our proposed model in this comparison. The result is shown in Tab. 5.2. We define G_g as the generated graph with 1000 nodes and 3984 edges, G_{hept} as the citation network of arxiv High Energy Physics - Theory section which contains 15233 nodes and 31398 edges, and G_{phy} as the citation network of arxiv Physics section which contains 37149 nodes and 174161 edges, k as the seed set size.

	$G_g, k = 10$	$G_g, k = 50$	$G_{hept}, k = 50$	$G_{phy}, k = 50$
Degree discount algorithm	214	295	873	7152
Martingale approach	132	223	295	2030
Proposed model	217	281	830	6850

Table 5.2 The influence evaluation of solutions of baselines and proposed models

We can see from the result that our proposed model achieves similar solution quality as the degree discount algorithm and outperforms the martingale approach. This result shows that our model has achieved a decent performance and still have much potential to improve. Because of the problems we mentioned in Sec. 5.4.1, we suppose that through parameter tuning and modification on reward computation,

our model can perform better.

The result we achieved is exicting because it shows that the greedy based reinforcement learning framework can perform well in the influence maximization setting. Furthermore, it reveals the potential of the research direction of employing reinforcement learning to the social network analysis problems.

Chapter 6 Conclusion

This paper focuses on the influence maximization problem on social networks, which has been extensively studied during the past decade because of its vast application potential. We propose a model based on the classic greedy algorithm framework for the influence maximization problem and use reinforcement learning combined with network embedding methods to improve the model performance. The motivation is inspired by an existing work on using reinforcement learning to solve greedy-based combinatorial optimization problems. Our main contribution is to transfer the existing framework to the different setting - the influence maximization problem on social networks. We propose new methods to modify the original framework to make it suitable for the influence maximization problem setting. Our modification on the reinforcement learning framework achieves performance improvement as expected.

Experiment results show that our proposed method works successfully on the influence maximization problem. The reinforcement learning agent learned the strategy to select the most influential nodes. Our modification on the reward function computation and the proposed training strategy do improve the efficiency and performance of the model. Our proposed model shows that the greedy algorithm based reinforcement learning framework can be successfully applied to the influence maximization setting, which reveals the potential of future research following this direction.

The model achieves a decent performance but does not outperform the state-of-art works, which indicates that our proposed model has much space of improvement. Due to the limited time and limited computation resources, we do not tune the parameters of reinforcement learning carefully. According to our knowledge on reinforcement learning, the paramter tuning has non-ignorable impact on the model performance.

In the future, we would like to improve the reward function computation

method and tune the parameter carefully. Furthermore, we want to complete the experiments based on other diffusion models such as the triggering model and the linear threshold model. We hope our work can shed more lights on future study of the influence maximization problem on social networks.

References

[1] D. Kempe, J. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC, USA*, 2003, pp. 137–146.

[2] S. Chen, J. Fan, G. Li, J. Feng, K.-l. Tan, and J. Tang, "Online topic-aware influence maximization," *Very Large Database Endowment Endowment (VLDB)*, vol. 8, no. 6, pp. 666–677, 2015.

[3] B. Liu, G. Cong, D. Xu, and Y. Zeng, "Time constrained influence maximization in social networks," in *IEEE International Conference on Data Mining (ICDM), Brussels, Belgium*, 2012, pp. 439–448.

[4] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 30, no. 10, pp. 1852–1872, 2018.

[5] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems (NIPS), Long Beach, CA, USA*, 2017, pp. 6348–6358.

[6] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT press Cambridge, 1998, vol. 135.

[7] W. Chen, L. V. Lakshmanan, and C. Castillo, "Information and influence propagation in social networks," *Synthesis Lectures on Data Management*, vol. 5, no. 4, pp. 1–177, 2013.

[8] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *ACM International Conference on Knowledge discovery and Data Mining (SIGKDD), San Jose, California, USA*, 2007, pp. 420–429.

[9] X. He and D. Kempe, "Stability of influence maximization," *arXiv preprint arXiv:1501.04579*, 2015.

[10] Y. Wang, G. Cong, G. Song, and K. Xie, "Community-based greedy algorithm for mining top-k influential nodes in mobile social networks," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Washington, DC, USA*, 2010, pp. 1039–1048.

[11] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[12] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1978.

[13] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, 2009, pp. 199–208.

[14] Q. Liu, B. Xiang, E. Chen, H. Xiong, F. Tang, and J. X. Yu, "Influence maximization over large-scal social networks: A bounded linear approach," in *ACM International Conference on Conference on Information and Knowledge Management (CIKM), Shanghai, China*, 2014, pp. 171–180.

[15] K. Jung, W. Heo, and W. Chen, "Irie: Scalable and robust influence maximization in social networks," in *IEEE International Conference on Data Mining (ICDM), Brussels, Belgium*, 2012, pp. 918–923.

[16] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks, washington, dc, usa," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2010, pp. 1029–1038.

[17] J. Kim, S.-K. Kim, and H. Yu, "Scalable and parallelizable processing of influence maximization for large-scale social networks," in *IEEE International Conference on Data Engineering (ICDE), Brisbane, Australia*, 2013, pp. 266–277.

[18] W. Chen, Y. Yuan, and L. Zhang, "Scalable influence maximization in social networks under the linear threshold model," in *IEEE International Conference on Data Mining (ICDM), Sydney, Australia*, 2010, pp. 88–97.

[19] S. Cheng, H. Shen, J. Huang, G. Zhang, and X. Cheng, "Staticgreedy: Solving the scalability-accuracy dilemma in influence maximization," in *ACM Inter-*

national Conference on Information Knowledge Management (CIKM), San Francisco, CA, USA, 2013, pp. 509–518.

[20] N. Ohsaka, T. Akiba, Y. Yoshida, and K.-i. Kawarabayashi, "Fast and accurate influence maximization on large networks with pruned monte-carlo simulations," in *AAAI Conference on Artificial Intelligence (AAAI), Québec City, Québec, Canada*, 2014, pp. 138–144.

[21] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *ACM-SIAM Symposium on Discrete Algorithms (SODA), Portland, Oregon, USA*, 2014, pp. 946–957.

[22] Y. Tang, X. Xiao, and Y. Shi, "Influence maximization: Near-optimal time complexity meets practical efficiency," in *ACM International Conference on Management of Data (SIGMOD), Snowbird, UT, USA*, 2014, pp. 75–86.

[23] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *ACM International Conference on Management of Data (SIGMOD), Melbourne, Victoria, Australia*, 2015, pp. 1539–1554.

[24] X. Wang, Y. Zhang, W. Zhang, X. Lin, and C. Chen, "Bring order into the samples: A novel scalable method for influence maximization," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 29, no. 2, pp. 243–256, 2016.

[25] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems (NIPS), Denver, CO, USA*, 2001, pp. 556–562.

[26] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), New York, NY, USA*, 2014, pp. 701–710.

[27] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *International Conference on World Wide Web (WWW), Florence, Italy*, 2015, pp. 1067–1077.

[28] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Francisco, CA, USA*, 2016, pp. 855–864.

[29] B. Perozzi, V. Kulkarni, and S. Skiena, "Walklets: Multiscale graph embeddings for interpretable network classification," *arXiv preprint arXiv:1605.02115*, 2016.

[30] S. Cao, W. Lu, and Q. Xu, "Grarep: Learning graph representations with global structural information," in *ACM International on Conference on Information and Knowledge Management (CIKM), Melbourne, VIC, Australia*, 2015, pp. 891–900.

[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[32] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), San Francisco, CA, USA*, 2016, pp. 1225–1234.

[33] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Chang, "Network representation learning with rich text information," in *International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina*, 2015, pp. 2111–2117.

[34] X. Sun, J. Guo, X. Ding, and T. Liu, "A general framework for content-enhanced network representation learning," *arXiv preprint arXiv:1610.02906*, 2016.

[35] C. Tu, W. Zhang, Z. Liu, and M. Sun, "Max-margin deepwalk: Discriminative learning of network representation." in *International Joint Conferences on Artificial Intelligence (IJCAI), New York, NY, USA*, 2016, pp. 3889–3895.

[36] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[37] A. Goyal, F. Bonchi, and L. V. Lakshmanan, "Learning influence probabilities in social networks," in *ACM International Conference on Web Search and Data mining (WSDM), New York, NY, USA*, 2010, pp. 241–250.

[38] M. Mathioudakis, F. Bonchi, C. Castillo, A. Gionis, and A. Ukkonen, "Sparsification of influence networks," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2011, pp. 529–537.

[39] K. Kutzkov, A. Bifet, F. Bonchi, and A. Gionis, "Strip: Stream learning of influence probabilities," in *ACM International Conference on Knowledge*

Discovery and Data Mining (SIGKDD), San Diego, CA, USA, 2013, pp. 275–283.

[40] D. Kempe, J. Kleinberg, and É. Tardos, "Influential nodes in a diffusion model for social networks," in *Springer International Colloquium on Automata, Languages, and Programming (ICALP), Lisbon, Portugal*, 2005, pp. 1127–1138.

[41] E. Mossel and S. Roch, "On the submodularity of influence in social networks," in *ACM Symposium on Theory of Computing (STOC), San Diego, CA, USA*, 2007, pp. 128–134.

[42] M. G. Rodriguez, D. Balduzzi, and B. Schölkopf, "Uncovering the temporal dynamics of diffusion networks," *arXiv preprint arXiv:1105.0697*, 2011.

[43] M. Xie, Q. Yang, Q. Wang, G. Cong, and G. De Melo, "Dynadiffuse: A dynamic diffusion model for continuous time constrained influence maximization," in *AAAI Conference on Artificial Intelligence (AAAI), Austin, Texas, USA*, 2015, pp. 346–352.

[44] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Halifax, NS, Canada*, 2017, pp. 385–394.

[45] H. Dai, B. Dai, and L. Song, "Discriminative embeddings of latent variable models for structured data," in *International Conference on Machine Learning (ICML), New York City, NY, USA*, 2016, pp. 2702–2711.

[46] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[47] G. J. Gordon, "Stable fitted reinforcement learning," in *Advances in Neural Information Processing Systems (NIPS), Denver, CO, USA*, 1996, pp. 1052–1058.

[48] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *Springer European Conference on Machine Learning and Data Mining (ECML PKDD), Porto, Portugal*, 2005, pp. 317–328.

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[50] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[51] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD), Paris, France*, 2009, pp. 199–208.

[52] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *ACM International Conference on Management of Data (SIGMOD), Melbourne, Victoria, Australia*, 2015, pp. 1539–1554.

Acknowledgements

I would like to express great gratitude to my advisor Professor Xiaofeng Gao. She helped me a lot on the thesis topic selection and gave me many professional and useful instructions throughout the project. Before I selected this topic, I had no research experience on social network analysis (SNA) and barely knows the related concepts and existing works. Professor Gao helped me get familiar with the topic and related areas with kindness and patience.

I must also thank my senior Chao Wang who inspired me of the core idea of this project and helped me with getting access to the servers.

I thank my seniors and peers in the ACM Class. I learned so much from those excellent people and received a lot of useful advice during this project. I want to specially thank my classmate Jiacheng Yang who patiently helped me several times deal with the server environment problems. Without him, I would have faced a lot more difficulties during this project.

I thank the Social Network Analysis Group, Advanced Network Laboratory, and APEX Laboratory for providing me with the servers.

I would express my sincere thanks to the ACM Class, Zhiyuan College, and Shanghai Jiao Tong University, for providing me with academic opportunities, abundant resources, and necessary facilities.

I am also sincerely grateful to my instructors in Shanghai Jiao Tong University, especially the faculties teaching ACM Class for their well-designed courses and careful instrutions during my undergraduate study. I would never finish this thesis without the knowledge and skills they helped me to learn.

I owe my hearties thanks to Professor Yong Yu for his instructions both on academic study and how to behave myself.

I would like to show my utmost gratitude to my parents and family. They always motivate me by giving their consistent love and understanding throughout my life.

Papers Submitted During Study for Bachelor's Degree

Papers submitted during study for Bachelor's Degree:

[1] Ran Li, **Nayun Xu**, Xutong Lu, Yucheng Xing, Haohua Zhao, Liqing Zhang, "Multi-person 3D Pose Estimation From Monocular Image Sequences (2019)," submitted to *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[2] Bowen Tan, **Nayun Xu**, Bingyu Kong, Cewu Lu,"Autonomous Driving in Reality with Reinforcement Learning and Image Translation (2018)," submitted to *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018.