Influence Maximization in Dynamic Networks Using Reinforcement Learning

S. Haleh S. Dizaji, Kishor Patil, Konstantin Avrachenkov

SN Computer Science (2024) 5:169
https://doi.org/10.1007/s42979-023-02453-1

ORIGINAL RESEARCH

Influence Maximization in Dynamic Networks Using Reinforcement Learning

S. Haleh S. Dizaji[1,2,3] · **Kishor Patil**[2] · **Konstantin Avrachenkov**[2]

Received: 23 March 2022 / Accepted: 25 October 2023

Abstract
Influence maximization (IM) has been widely studied in recent decades, aiming to maximize the spread of influence over networks. Despite many works for static networks, fewer research studies have been dedicated to the IM problem for dynamic networks, which creates many challenges. An IM method for such an environment, should consider its dynamics and perform well under different network structures. To fulfill this objective, more computations are required. Hence, an IM approach should be efficient enough to be applicable for the ever-changing structure of a network. In this research, an IM method for dynamic networks has been proposed which uses a deep Q-learning (DQL) approach. To learn dynamic features from the network and retain previously learned information, incremental and transfer learning methods have been applied. Experiments substantiate the good performance of the DQL methods and their superiority over compared methods on larger sizes of tested synthetic and real-world networks. These experiments illustrate better performance for incremental and transfer learning methods on real-world networks.

Keywords Influence maximization · Dynamic networks · Reinforcement learning · Deep Q-learning · Incremental learning · Transfer learning

Introduction

Networks play an important role in information dissemination and provide a means for intervening propagation process and controlling it. In many social and human networks, it is very demanding to maximize the spread of information, which is the objective of the influence maximization (IM) problem. Some examples include spreading information through social or human networks (e.g., viral marketing), spreading fake (rumors) or true news during elections, and maximizing road traffic [1]. Furthermore, detecting the most influential nodes in the network can help in immunizing the network by isolating those nodes, such as controlling the spreading of a contagious disease, virus, or rumors. These networks usually are dynamic in nature and evolve continuously over time. For example, in social networks, new friendship links can appear or old ones can disappear. The dynamism presents many challenges in the analysis of these networks.

During recent decades, many efforts have been made for solving IM problem over networks. The objective of an IM method is to maximize information propagation through a network by triggering a series of nodes (e.g., by giving a piece of information to certain people), which could cause the most significant effect on information spread. This problem is known to be an NP-hard problem under most propagation processes [2]. Hence, many approximation methods have been proposed for solving it, which try to approximate the future influence of each node. This problem can be considered as a trade-off between the method's accuracy and its efficiency (space and time complexity). Furthermore, in a network with probabilistic propagation, this problem becomes more challenging.

✉ S. Haleh S. Dizaji
h.sdizaji@tabrizu.ac.ir

Kishor Patil
Kishor88k@gmail.com

Konstantin Avrachenkov
k.avrachenkov@inria.fr

[1] Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran

[2] Inria Sophia Antipolis, Valbonne, France

[3] Klagenfurt University, Klagenfurt, Austria

Published online: 08 January 2024

In addition to the aforementioned problems in static networks, many challenges arise for the IM problem on dynamic networks. Due to the ever-changing structure of these networks, an IM method should be generalizable or time-efficient to handle these dynamics. There have been several efforts for solving this problem. In the paper [3] extensions of classical methods for dynamic networks were proposed. While these methods capture some dynamic properties of a network, but they cannot always guarantee a good performance. In the paper [4], a subgraph-based method for dynamic networks was introduced. This method also requires test network snapshots beforehand, which might not be always available previously and requires updating and running the method for arriving networks. In the method of paper [5], influential nodes are tracked incrementally, using already found seed nodes for previous snapshot and replacing them. This method assumes that seed sets for consecutive snapshots are similar which may not be true for networks with high dynamics. Hence, there is the need for devising a method that captures some general properties of the network during different time steps, where the learned properties can be applicable for unseen structures.

The IM problem can be formulated as a decision-making problem and it can be solved by reinforcement learning (RL) methods, which are powerful tools for Markov decision processes (MDP). These methods can capture the probabilistic structure of the environment (here is network) and find an optimal policy (order of nodes that are triggered or seed nodes) under certain conditions. The Q-learning approach as one of RL methods has been widely applied in several areas. This method uses state–action values (Q values) for finding an optimal policy that Q function can be evaluated using Q tables, approximation Q-learning, and deep Q-learning (DQL). The approach of Q tables suffers from memory limitations for large state and action spaces. The approximation methods with naive functions might be inaccurate in some situations. Hence, DQL methods have become more popular, where neural networks (NNs) are used for approximating Q functions. This provides nonlinear dependencies without any assumption about parametric Q function that results in increased accuracy of these methods.

DQL methods have been applied to solve graph problems, which are scalable and efficient for large graphs or graphs with dynamic properties. In [6–9], DQL methods have been proposed for solving optimization problems over graphs, where methods of [7] and [9] have been used for dynamic networks. In these methods that are based on the method introduced in [6], NNs are used for embedding networks and approximating Q function. The introduced approximation function in [6] employed in our research is an extendable method that can be used for networks with similar structures, making it suitable for dynamic networks. In RL frameworks proposed in [9] and [8], pre-calculated reward functions are applied, which might not be efficient and scalable for larger dynamic graphs. In addition, calculating rewards in execution time is not efficient. Hence, in this research, an RL framework with an immediate reward function that is mentioned in [10] is defined, which makes it efficient for rather large graphs.

The other drawback of the aforementioned DQL models is their weakness in extending the DQL model of [6] for dynamic networks. Recently, several incremental (lifelong) learning methods have been proposed for tackling dynamic and streaming data. The objective of these methods is to avoid forgetting previously learned information while learning from new data (catastrophic forgetting). These methods can be categorized into three groups: (i) regularization, (ii) structural, and (iii) replay memory (rehearsal)-based methods [11]. In regularization-based methods, importance of parameters in old data, is included in the regularization term of the loss function while training on new data. Replay memory methods use rehearsal strategy of previous data while training on new data and structural methods dynamically change the structure of the NN in presence of new data by adding new neural layers (or neurons). Because of improved performance of methods using incremental learning methods in learning from dynamic data, these methods are utilized in our research for capturing some dynamics of a network, and the method of [6] has been extended using two regularization-based methods, elastic weight consolidation (EWC) [12] and synaptic intelligence (SI) [13], and a rehearsal method. As it has been shown in this research, they have resulted in improved performance of our method for smaller networks. Besides, a naive transfer learning method is applied, which also has improved the results for small networks.

In a nutshell, we summarize our contributions as below:

- We define the RL framework with immediate rewards [10] for the IM problem which reduces calculation time of reward function.
- We introduce the DQL optimization method for dynamic networks using incremental and transfer learning algorithms.
- The proposed methods are evaluated using different real-world and synthetic networks and are compared extensively with some heuristic and sketch-based methods. The experiments depict the good performance of our proposed methods especially for larger sizes of the tested networks.

Related Work

During recent decades, a large number of works are dedicated for IM over static networks, and some of thesm are reviewed in [14]. These methods include simulation-based [2, 15–17], heuristic [18–26] sketch-based [27–30], and

other methods [31]. In these methods, network structure and propagation probabilities are assumed to remain constant. In simulation-based methods, Monte Carlo (MC) simulations are performed to estimate influence of nodes [14], which has high time complexity for large networks. In some of these methods, these simulations are reduced, such as CELF [15] and CELF++ [16] methods. In a group of heuristic methods some centrality measures are defined, which approximate influences of nodes according to some measure. These measures include, degree centrality, PageRank [18], distance centrality [19], and collective influence (CI) method [14, 20]. In the CI method, degrees of distant neighbors are considered as a proxy measure instead of direct degree of a node. These methods suffer from the overlapping of nodes' influences. Therefore, some methods, such as degree discount [21], group PageRank [22], and [26], have been introduced for mitigating this effect [14]. The degree discount method considers a discount factor in influences of selected seeds' neighbor nodes. In sketch-based methods, for improving the theoretical efficiency of simulation-based methods and retaining their accuracy, some sketches are constructed which speed up the computations of simulation-based methods [14]. One of known methods in this category is reverse influence sampling (RIS) method [28], where a number of reverse reachable (RR) sets for random nodes are generated using sampled graphs (sketches). .These sets are used for selecting influential nodes (more repeated nodes in these sets). The number of these RR sets is selected in a way to guarantee the approximate ratio.

According to our knowledge, despite the vast amount of studies for static networks, few have been dedicated to dynamic networks. In [3], three extensions of static IM methods (i.e., degree discount [21], CI [20], and RIS [28]) for dynamic networks are proposed. Dynamic degrees [32] have been employed in dynamic degree discount and CI methods, where varying neighborhoods of nodes are considered. In the dynamic extension of RIS method, all durations of network (snapshots) are considered for calculating RR sets. These methods have less time complexity than simulation-based methods, but they do not always guarantee good performance. In [33], a greedy adaptive IM method for networks with dynamic propagation probabilities has been proposed which guarantees its performance in the dynamic independent cascade model. In [4] a (DIM) method for evolving networks is proposed in which sample subgraphs are generated from different network snapshots and are compressed by two approaches (horizontal and vertical compression) to calculate average reachability of each node to select seed nodes. In paper [34], an IM method for evolving networks is proposed where it is assumed network changes are observed through node probing. In this method, a node probing algorithm is proposed for approximating best seed sets in every network snapshot. In paper [5], an influential node tracking method for evolving networks (both structure and propagation probabilities) is proposed. In this paper, the most influential nodes for each snapshot (seeds) are incrementally approximated based on the seeds found for previous snapshot and using an interchange heuristic (upper bound interchange greedy method), which constantly replaces previous seed nodes using an upper bound approach of [35] for replacing gain, which reduces MC simulations.

Recently, RL methods have been extensively employed in several applications and have shown good performance on different optimization problems over networks, including IM. In [10], a combination of RL and game theory has been used for multi-party IM over static networks. In [6], a DQL method has been proposed to solve three combinatorial optimization problems for static networks. This work that has been employed in our research introduces a NN embedding and a Q function approximation method where nodes of the network are mapped into a low dimensional space using a nonlinear embedding formula. An advantage of this method is the independence of the number of its parameters from the input data. Hence, it has been shown that a trained model for small networks can be used for larger networks with similar structures. In [6], only static networks have been considered, but its approximation formula can also be applied for dynamic networks, as also demonstrated in our research.

The method of [6] has been employed as the base method in several studies [7–9]. In [9], this method has been extended for topic-aware IM, that it considers attributes of users in addition to graph properties in embedding formula and a Double Q-learning algorithm has been employed. As stated in this paper, this method can also handle dynamic propagation probabilities, but dynamics network structure is not considered. In [7], the method of [6] has been used for graph protection problem in dynamic networks, where this problem is converted into a Minimum Vertex Cover problem. In this work, several neural models are trained on several network snapshots and the best model is selected. Hence, it cannot properly capture the dynamic properties of the network. In [8], a combination of a supervised method (for selecting good nodes) and the method of [6] has been used for three combinatorial problems including IM. This research confirms its scalability for large static networks.

In our research, the method of [6] is employed as the base method, and an RL Framework with immediate rewards is applied. Furthermore, this method is extended for dynamic networks using incremental and transfer learning algorithms.

Preliminaries and Influence Maximization

In this research, the proposed IM method is extended for dynamic networks. In order to define the environment precisely, in this section, first, the considered assumptions are

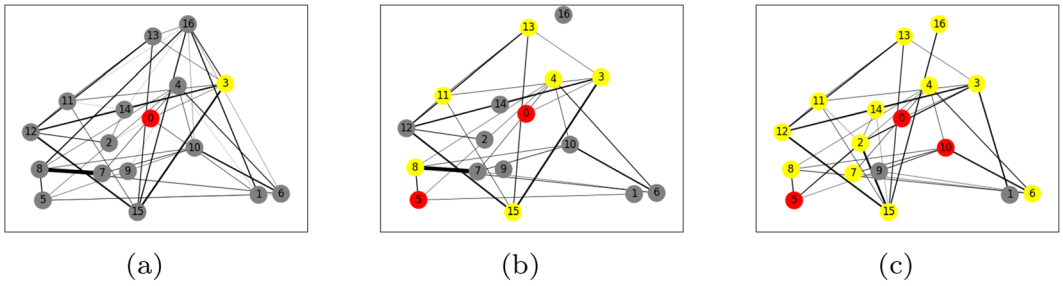

Fig. 1 IM process in three snapshots of a dynamic network (a part of raccoon dataset [39]). Red, yellow and gray nodes represent seed, active and inactive nodes respectively. Edge thicknesses are proportional to propagation probabilities over them

explained and then the IM problem and the proposed methods are presented.

Dynamic Networks and Diffusion Process

In this research, a dynamic network $\{G^t\}_1^T$ is represented as a series of network snapshots at each time step, $G^1, G^2, \ldots, G^T$. Each snapshot G^t is a graph represented as $\{V, E^t\}$ where V is the set of nodes of the graph which are assumed to remain unchanged during the time and E^t is the set of un-directed edges of the graph at time t. In this research, only dynamics over edges are considered. For propagation process over this network, a weighted network with weight matrix W is considered, where W is symmetric zero-diagonal matrix and each entry $w(i, j)$ represents the probability of propagating information through edge between nodes i and j. The propagation process is also assumed to remain constant. In other words, in the presence of links, propagation probabilities over them do not change over time.

Different diffusion processes can be assumed in this model and the propagation model assumed for this research is an epidemic process for weighted networks. This diffusion model is the susceptible-infected process introduced in [36, 37], which has also been defined for weighted networks [38]. In this process, nodes can have one of two states, S (susceptible) and I (infectious), and each node after infection remains in this state (I) forever. In this model, each infected node i can infect its neighbors at each time step, e.g., j with probability equal to their edge weights $w(i, j)$. This process continues until no more nodes can be infected.

IM Problem over Dynamic Networks

The objective of IM is to maximize the spreading of information through the network during a limited or unlimited time. Therefore, it is crucial to select the optimal seed nodes for activation. The activation process can happen in one or several steps, and the latter is assumed in this paper. In other words, by this assumption, after selecting each seed node, the propagation process is performed for one step. The seed selection and the propagation process are performed until no inactive node is in the network. An IM solution in this environment tries to find an order of seed nodes $\{s_1, s_2, \ldots, s_k\}$ which are activated one at each step, such that the overall discounted influence is maximized.

In the case of dynamic networks, because of the evolving structure of a network, at each step of seed selection, the network structure can change. Therefore, in the IM problem over a dynamic network, the objective is to find an order of seed nodes that are activated one at each network snapshot G^t. This process is illustrated in Fig. 1. As it is represented, after selecting a seed node at each network snapshot, the propagation process is performed for one step according to that snapshot. This process continues until all network is activated.

Reinforcement Learning

In this research, the explained IM problem has been formulated as an MDP problem and has been solved by an RL algorithm. In this section, after formulating the RL problem and a brief introduction to Q-Learning, the proposed methods are presented.

Framework

RL that is based on MDP, is a powerful optimization tool for solving decision-making problems for its solution without the complete knowledge of the system. RL framework consists of a tuple (S, A, T, R, γ) where S represents the state space, A is the set of actions, T is a function $S \times A \rightarrow S$ representing transition probabilities, R is the reward function defined as $S \times A \times S \rightarrow \Re$ and γ is a discounting factor. Given this information about the environment, the goal of RL is to find an optimal policy π which is defined as a function $S \rightarrow A$ (in the deterministic policy) and maximizes total return J under policy π which is formulated as:

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r_t(\tau)\right], \quad (1)$$

where τ represents an episode under policy π and r_t is the immediate reward at step t of this episode.

The IM problem to be solved in this research is formulated as an RL framework which is defined as the following:

Environment

The environment consists of a dynamic network as defined in the beginning of "Preliminaries and Influence Maximization" section with its diffusion model which is the susceptible-infected process in this research. Propagation process of the network represents the environmental effect that information is propagated through the network after selecting an action.

State Space

States consist of network status at each time step (i.e., snapshots with propagation probabilities) and activated nodes until the current step. This state space can be very large (it would be in the order of 2^N in a static network with N nodes), but as described in the next section, these states are converted into real vectors and it is not required to retain a Q-table.

Action Space

Action at each state is to select the next node as a seed node and activate it. Hence, there are N actions at each state.

Reward

After selecting an action, the environment returns an immediate reward evaluated by the number of activated nodes after one-step propagation, defined as follows:

$$r_t = [I(t) - I(t-1)], \quad (2)$$

where $I(t)$ is the number of activated nodes until time t. With this definition of the reward function, it is not required to evaluate it previously, which can be time-consuming and in addition, it requires a large space for storing these values. In addition, the reward function is not assumed to be constant for each (state, action) pair, which reflects the uncertainty property of the network due to propagation probabilities. Therefore, reward values may differ in several simulations, but it is converging in expectation in the case of static networks. The expected value of the reward function at each step can be calculated as the following:

$$\mathbb{E}[r_t] = \sum_{v\notin I(t-1)}\left(1 - \prod_{\substack{u\in N_t(v),\\ u\in I(t-1)}}(1-w(u,v))\right), \quad (3)$$

where $N_t(v)$ represents the set of neighbors of node v at time step t. Since calculating this expectation at each state requires considering all edges between activated and inactive nodes, it is time-consuming. Hence, immediate rewards are applied in this research.

Q-Learning

Our RL approach is based on the Q-learning method. In this method, optimization is performed by updating state-action values (Q values) according to the following equation

$$Q_{t+1}(s_t,a_t) = Q_t(s_t,a_t) + \alpha_t \left(r_t + \gamma \max_a Q_t(s_{t+1},a) - Q_t(s_t,a_t)\right), \quad (4)$$

where $Q_t(s,a)$ is the value of selecting action a in state s at time t, r_t is the immediate reward at time t and α_t is learning rate at time t. Since the original Q-learning algorithm needs to retain Q values to update it constantly, it is not suitable for large state and action spaces. Hence, in this research, an approximation-based Q-learning algorithm using deep NNs, i.e., DQL method has been used which is explained in the next section.

Deep Q-Learning and Double Q-Learning

In DQL, Q values are approximated using deep NNs and the parameters of these networks can be updated using stochastic gradient methods. This parameter update formula is as follows:

$$\boldsymbol{\psi_{t+1}} = \boldsymbol{\psi_t} + \alpha_t(Y_t^Q - Q(s_t,a_t;\boldsymbol{\psi_t}))\nabla_{\boldsymbol{\psi_t}} Q(s_t,a_t;\boldsymbol{\psi_t}), \quad (5)$$

where ψ_t are model parameters and Y_t^Q is the target Q-value α_t. The target Q-value Y_t^Q is defined by

$$Y_t^Q = r_{t+1} + \gamma \max_a Q(s_{t+1},a;\boldsymbol{\psi_t}). \quad (6)$$

As noted in [40], this method has an overestimation problem, and for solving this overestimation problem Double DQL algorithm (DDQL) has been introduced in [41]. This algorithm uses two separate models, policy and target networks for action selection and evaluation. In this method, the target Q-value is formed as follows [41]:

$$Y_t^{\text{DoubleQ}} = r_{t+1} + \gamma Q(s_{t+1}, \operatorname*{argmax}_a Q(s_{t+1},a;\boldsymbol{\psi_t});\boldsymbol{\psi_t'}), \quad (7)$$

where $\boldsymbol{\psi_t}$ and $\boldsymbol{\psi_t'}$ represent the parameters of policy and target networks, respectively. DDQL algorithm increases the

speed of convergence for the Q-learning algorithm; hence it is used in this research.

Proposed Methods

The Q-learning approach proposed in this research is utilized for both static and dynamic networks and is extended for dynamic networks using incremental and transfer learning algorithms. In this section, first, Q-learning solution is explained, then its extension details are presented.

Q-Function Approximation

The approximation function for Q values is adapted from [6]. In this method for transforming raw-state information into a low-dimensional vector space, a network embedding method is proposed which exploits the power of recurrent NNs. In this research, this structure is used with some modifications, which are:

$$\mu_v^{k+1} = \text{Relu}(\psi_1 x_v + \psi_2 \sum_{u \in N_t(v) \bigcup v} w(u,v)\mu_u^k + \psi_3 \sum_{u \in N_t(v)} \text{Relu}(\psi_4 w(u,v))), \tag{8}$$

$$\mu_g = \sum_{v \in Nodes(G)} \mu_v^K, \tag{9}$$

where μ_v^k is a d-dimensional vector for node v at iteration k of embedding process, x_v represents activation status of node v (it is 1 if the node is activated and 0 otherwise), $w(u, v)$ is the propagation probability between nodes u and v and $\psi_1 \in \mathfrak{R}^{1\times d}$, $\psi_2 \in \mathfrak{R}^{d\times d}$, $\psi_3 \in \mathfrak{R}^{d\times d}$, and $\psi_4 \in \mathfrak{R}^{1\times d}$ are the model parameters. The variable μ_g, the summation over vectors of all nodes, represents the network state. The iteration formula in Eq. 8 is performed for K steps with zero initial vectors. This embedding formula captures the network's activation and structure information (including propagation probabilities). In this research, edge weights are included in summation over neighboring nodes (the second expression in Eq. 8) and the vector of each node itself is also considered in this summation as its history. An advantage of this embedding formula is the independence of the number of model parameters from the size of the network so that it can be applied for large networks. In addition, as it is represented in the experiments section ("Experimental Setup and Numerical Results" section), a model trained for a small size network can be employed for testing (or training) on more extensive networks with similar structures.

For calculating Q values, another NN is employed as in [6]. Calculated network and selected node vectors, i.e., μ_g and μ_a are given as the state s and actions a inputs to this Q function which is illustrated in the following formula:

$$Q(s,a) = \psi_5^T \text{Relu}(\text{concat}(\psi_6\mu_g, \psi_7\mu_a)), \tag{10}$$

where μ_a is vector of node a and $\psi_5 \in \mathfrak{R}^{2d\times 1}$, $\psi_6 \in \mathfrak{R}^{d\times d}$ and $\psi_7 \in \mathfrak{R}^{d\times d}$ are model parameters.

In the vanilla–DDQL model, benefiting from the extendable property of the model, it is trained using one train snapshot and is used for testing on several test snapshots. Experiments for training on several snapshots of a small dynamic network illustrate no significant difference with training on one or two snapshots and because of memory requirements it is not efficient for large networks. This algorithm is given in Algorithm 1. In this algorithm, a replay memory is utilized to retain traversed samples and use them for training the model in the subsequent iterations. At first iterations of this algorithm, a policy relaxation method has been used, in which no exploitation step, i.e., selecting the action from the model, is performed, and it only traverses state-space randomly and fills up the replay memory. Once the replay memory is full, epsilon-greedy policy [42] is followed for action selection. In this policy, with probability ϵ exploration (selecting action randomly) is performed. Otherwise, action is selected using the model. This policy helps the model to explore the environment as it learns. But for meeting convergence of the algorithm, this exploration is decayed through time, concentrating more on the model. After action selection, information is propagated as the environmental effect, and the next state and reward are determined and the tuple $(state, action, reward, next\ state)$ is written in replay memory for use in training. Then if memory is filled, a batch from memory is sampled randomly and the mean squared error (MSE) loss is calculated for batch b according to:

$$\text{Loss}_b = \frac{1}{b} \sum_{(s_t,a_t,r_{t+1},s_{t+1})\in b} \Big(r_{t+1} + \gamma Q(s_{t+1}, \underset{a}{\text{argmax}}\, Q(s_{t+1}, a; \psi_{policy}); \psi_{target}) - Q(s_t, a_t; \psi_{policy}) \Big)^2. \tag{11}$$

Parameters of the policy network are updated by Adam optimizer and target net parameters are updated using Polyak averaging method [43] as following:

$$\boldsymbol{\psi}_{target} = \tau * \boldsymbol{\psi}_{policy} + (1-\tau) * \boldsymbol{\psi}_{target}. \tag{12}$$

An episode terminates when all nodes of the network are activated.

Algorithm 1 DDQL Algorithm

Input: Network snapshot G and propagation probabilities W
Output: Model parameters ψ_{policy} and ψ_{target}

Initialize ψ_{policy} and ψ_{target}
procedure TRAIN-DDQL(G, W)
  for $episode = 1, 2, ..., E$ **do**
    Current state $s \leftarrow s_0$
    while *Network is not exhausted* **do**
      if *Replay Memory is not full* **then**
        Choose action a randomly ▷ *Policy relaxation*
      else
        Sample z $\sim$ Bernoulli(ϵ) ▷ *Epsilon greedy algorithm*
        if $z = 0$ **then**
          Choose action a randomly
        else
          action $a \leftarrow \text{argmax}_a\, Q(s_t, a;\, \psi_{policy})$
      Propagate information for one step, get next state s'
      reward $r \leftarrow I(t) - I(t-1)$
      Add tuple (s, a, r, s') to Replay Memory
      if *Replay Memory is full* **then**
        Sample batch b from Replay Memory ▷ *Training*
        Calculate Loss according to Eq. 11
        Update ψ_{policy} by Adam optimizer
        Update ψ_{target} by Eq. 12

Incremental and Transfer Learning

For tackling the problem of the ever-changing structure of a dynamic network, incremental and transfer learning methods are also applied, which benefit from several network snapshots available for training (two snapshots used in this research). In this section, these methods and overall extended DDQL algorithms are presented.

Incremental learning is the method of training a NN model in the presence of dynamic or streaming data. A major problem that arises is catastrophic forgetting of already learned information while learning new information. In other words, previous weights of the NN change so dramatically that it does not suit previous data anymore. Hence, incremental learning methods try to alleviate this catastrophic forgetting, that the learned model is still useful for previous data and new data.

The incremental learning methods employed in this research are: EWC [12], SI [13], and replay memory methods. In addition, a naive transfer learning method is used for training the model using more dynamic information. In the incremental methods, the model trained on the first snapshot, is updated according to the used incremental method while training on the second snapshot. These methods are explained in the following:

- **EWC method** EWC is a regularization-based incremental learning method introduced in [12]. In this method, the importance of parameters in the previous task is evaluated using the Fisher information matrix and their training in the new task is slowed down according to these values. To meet this goal, after training the model for task A, this model is then trained on task B using the modified loss function according to:

$$\mathcal{L}(\psi) = \mathcal{L}_B(\psi) + \sum_i \frac{\lambda}{2} F_i(\psi_i - \psi_{A,i}*)^2, \tag{13}$$

  where $\mathcal{L}_B$ is the loss for task B only, λ is the importance of the old task comparing with the new task, $\psi_{A,i}*$ is the final value of parameter ψ_i in previous task A and F is Fisher information matrix which is defined as:

$$F_i = \frac{1}{L} \sum_{i=1}^{L} (\nabla_{\psi_i} \log \mathcal{L}(\psi))^2, \tag{14}$$

  where L is the number of samples chosen from the previous task A for calculating the Fisher values. These values represent importance of parameters in previous task and control their training in the next task.
- **SI method** SI method proposed in [13] is another regularization-based method for incremental learning, which evaluates the importance of parameters in previous task

A by measuring amount of contribution of parameters in the change of loss. The loss function in the new task B is presented in the following:

$$\mathcal{L}(\psi) = \mathcal{L}_B(\psi) + c \sum_i \Omega_i^B (\psi_{A,i} * -\psi_i)^2, \tag{15}$$

where c is the strength parameter and Ω_i^B denotes per-parameter regularization strength which is defined by:

$$\Omega_i^B = \frac{\omega_i^A}{(\Delta_i^A)^2 + \zeta}, \tag{16}$$

where ω_i^A is the importance measure of parameter ψ_i in task A, Δ_i^A defined as $\Delta_i^A = \psi_{A,i} * -\psi_{i_{\text{init}}}$ is the amount of change in parameter ψ_i during training for task A and ζ is damping parameter which avoids division by zero in the case of $\Delta_i^A \to 0$. The importance measure for each parameter is given by:

$$\omega_i^A = \sum_{k=1}^{T} \nabla_{\psi_i} \mathcal{L}_A(\psi) \delta\psi_i, \tag{17}$$

which measures the contribution of parameter ψ_i in the change of loss during the previous task A. T is the total number of parameter update steps during task A.

- **Replay memory** The replay memory (rehearsal) method is common in RL, which alleviates catastrophic forgetting by replaying data from previously learned tasks. In this method, after training the model on task A, some samples, i.e., (*state*, *action*, *reward*, *next state*) tuples are stored in the memory and used for training for the next task B. At this step, stored data from the previous task are selected for training the model with a certain probability. This method for training the model on task B is initialized with the learned parameters for task A.
- **Transfer learning** While incremental learning algorithms try to find a model suitable for both new and old data, in transfer learning methods, the goal is to obtain a model for new tasks using already learned model for previous data. In this research, a simple transfer learning algorithm is employed, where the model for training on new data is initialized with parameters learned from the previous data. This speeds up training the model for the new task and carries some information of the prior task.

The combination of these algorithms with the vanilla–DDQL algorithm is given in Algorithms 2, 3, and 4. Algorithm 2 illustrates the use of regularization-based incremental learning methods i.e., EWC and SI methods. In this algorithm, the same as DDQL Algorithm 1, the model is trained using the first training snapshot, then learned parameters are used as the initialization for training on the second train snapshot. In this step, according to the used incremental method, the corresponding loss regularization expression is added to the loss function, and the model is trained using this new loss. In Algorithm 3, replay memory method for the DDQL algorithm is illustrated. In this method, also after training the model for the first snapshot, samples of this step are selected and stored for replaying in the next training step. When the model is being trained using the second snapshot with a certain probability p, the stored samples from the previous training step are selected and used for training. Finally, Algorithm 4 represents the naive transfer learning algorithm, where it only keeps the previously trained model and updates it using the second train snapshot.

Algorithm 2 Incremental Regularization DDQL Algorithm

Input: Network snapshots G^1, G^2 and propagation probabilities W, Loss Regularization Func

Output: Model parameters ψ_{policy} and ψ_{target}

1: Initialize ψ_{policy} and ψ_{target}
2: $\boldsymbol{\psi_{policy}} \leftarrow$ Train-DDQL(G^1,W)
3: Train-DDQL(G^2,W) with Loss Regularization Func

Algorithm 3 Rehearsal DDQL Algorithm

Input: Network snapshots G^1, G^2 and propagation probabilities W, Replay Memory Probability p

Output: Model parameters ψ_{policy} and ψ_{target}

1: Initialize ψ_{policy} and ψ_{target}
2: $\boldsymbol{\psi_{policy}} \leftarrow$ Train-DDQL(G^1,W)
3: $Prev\ Replay\ Memory \leftarrow Sample\ from\ Replay\ Memory$
4: Train-DDQL-Replay(G^2,W, p)

Algorithm 4 Transfer DDQL Algorithm

Input: Network snapshots G^1, G^2 and propagation probabilities W
Output: Model parameters ψ_{policy} and ψ_{target}

1: Initialize ψ_{policy} and ψ_{target}
2: $\boldsymbol{\psi_{policy}} \leftarrow$ Train-DDQL(G^1,W)
3: Train-DDQL(G^2,W)

Experimental Setup and Numerical Results

In this section, implementation details and obtained results are presented. First, a few IM methods that have been compared with the proposed methods, are explained and then experimental details for both synthetic and real-world datasets are presented.

Baseline IM Methods

Our proposed IM methods are compared with some heuristics and a sketch-based IM method. Some extensions of these methods for dynamic networks have been compared with our methods. The following briefly explains these methods:

Heuristic Methods

The heuristic algorithms compared in this research include heuristics for static and dynamic networks. The heuristics for static networks include max degree and degree discount. For dynamic networks, in addition to static heuristic algorithms, some dynamic heuristics, such as dynamic degree and dynamic degree discount algorithms, were also tested. These algorithms are explained in the following:

- **Degree** In the max degree algorithm, seed nodes are selected according to their degrees (or weighted degrees) in descending order if they are not activated until the current time step. In the static case, nodes degrees are calculated at the start of execution, and in the dynamic case, they are calculated at every step according to the current snapshot.
- **Degree discount** The degree discount algorithm introduced in [21] assumes a discount factor for neighboring nodes of seed nodes in calculating their degrees. The discounted degrees of nodes are obtained using the following equation:

$$dd_u = k_u - 2t_u - (k_u - t_u)t_u\lambda, \tag{18}$$

where k_u denotes the degree of node u without discount (in this research, both weighted and naive degrees are assumed), t_u is the number of times node u has been in the neighborhood of a seed node, and λ represents the infection probability of node u by the currently selected seed. Note that the λ in this research is propagation probabilities.

- **Dynamic degree** In max dynamic degree algorithm, nodes are ordered by their dynamic degrees defined by [32]:

$$D_T(v) = \sum_{1<t\leq T} \frac{|N_{t-1}(v) \setminus N_t(v)|}{|N_{t-1}(v) \bigcup N_t(v)|} |N_t(v)|, \tag{19}$$

where $D_T(v)$ represents the dynamic degree of node v calculated during T network snapshots, and $|\cdot|$ represents cardinality operator. This method evaluates changes in neighbors of the nodes as their dynamic degrees. In this research, both weighted and unweighted dynamic degrees have been calculated.

- **Dynamic degree discount** Dynamic degree discount heuristic introduced in [3] considers dynamic degrees, defined in Eq. 19 as node degrees in Eq. 18. For calculating discounted degrees, all nodes adjacent to node u during all time steps (T) are considered as the neighborhood of that node. Weighted dynamic degrees are also considered for calculating weighted dynamic discounted degrees.

RIS Methods

In addition to heuristic methods, our methods have also been compared with RIS method. In this research, both static and dynamic extension of RIS method have been tested, which are explained in the following:

- **Static RIS** In the static RIS method, RIS method is applied on the aggregated train snapshots (which are different from test snapshots). In this method, aggregated network of train snapshots is obtained and used for gener-

ating instance graphs (transition probabilities were used for selecting edges). RR sets were constructed using breadth first search algorithm with a maximum distance of N (number of nodes in the network) from the randomly selected node. The number of generated RR sets depending on the size of the network was chosen between 500 and 10,000 sets (because of increasing time complexity, less RR sets were generated for larger networks). Since the seed set obtained by this method might not cover all nodes (because of probabilistic selection) and the purpose of the assumed IM problem is to infect all nodes, after obtaining the seed set, if the list was smaller than node numbers, the remaining nodes are added to the end of this list in random order.

- **RIS for each Snapshot** In this variation of RIS method for dynamic networks, RIS method is applied for each test snapshot separately and ordered seeds are obtained. From this set, the first inactive seed is selected as the next seed (for the current snapshot). In this method, also in the case of not covering all nodes by each seed set, remaining seeds are selected randomly. Because of higher time complexity of this method during test, less RR sets were created which were in the range of 200–1000 sets according to the size of the network.

In the methods which require obtaining seeds sets beforehand (i.e., degree, degree discount, dynamic degree, dynamic degree discount and static RIS), training snapshots were used for calculations, and these sets were tested on test snapshots (by discarding active seeds). In the methods, which obtain seeds for each snapshot (i.e., degree algorithms and RIS for each snapshot), test snapshots were used for calculations.

Although, in this research, the explained methods have been compared with the proposed methods, but for more clarity of the results, only a few of them are illustrated in this paper. These results are discussed in the next section.

Evaluation metrics

We use mainly two metrics to evaluate the proposed methods. The average discounted reward and the average seed length. The first metric is the main evaluation metric as our proposed methods are reward-based optimization. The other metric, seed length, is only used to provide the completeness of the results and to ensure how models are performing even though we are not directly optimizing for it. These are explained in detail as below:

- **Average discounted reward** This metric evaluates total discounted rewards as defined in Eq. 20:

$$R = \sum_{t=0}^{M-1} \gamma^t r_t, \tag{20}$$

where r_t represents the immediate reward at time step t, i.e., the number of activated nodes at time t and M is the total number of steps required for activating all nodes in the network. Since the number of activated nodes at each time step is discounted by the γ factor, this metric evaluates the how efficiently model selects the nodes so that information spread is fast. The average value of this measure for several test iterations is calculated and reported in the experiments. The higher value here denotes that model is able to identify the efficient nodes quickly.

- **Average seed length** This metric represents the average number of selected seed nodes for activating the whole network. In other words, it represents the number of time steps required for activating all nodes of the network through the seed selection and propagation process. According to this definition, the lower value of this metric represents faster activation of the network.

Experimental Settings

This section explains the experimental setup and provides detailed experiments on both static and dynamic networks. The general setup is explained below. For managing network data, the NetworkX library has been used. Besides, for speeding up the computations, CUDA technology has been used. These models are implemented on a system with an Intel Core i5 processor with NVIDIA GeForce GTX 1650 GPUs and 4 GB GPU RAM.

The parameters of the policy and target networks in the DDQL algorithm are initialized using the Xavier method [44] which performed better than other tested initialization methods. In the epsilon greedy policy for training DDQL model, an exponentially decreasing epsilon has been used as below:

$$\epsilon = \epsilon_{\text{end}} + (\epsilon_{\text{end}} - \epsilon_{\text{start}}) * \exp^{-\text{steps\_done}/\text{decay\_factor}}, \tag{21}$$

where ϵ_{start} and ϵ_{end} represent initial and final values of ϵ which decays with decay_factor parameter. steps_done represents number of steps performed (actions selected) during the execution of the epsilon greedy policy. For training the model, Adam optimization method with constant and decreasing learning rates has been used. For decreasing learning rate, an exponential decay function as the following equation has been utilized:

$$\text{lr} = \text{lr}_0 \exp^{-\text{lr\_decay}*\text{lr\_steps}}, \tag{22}$$

Table 1 Hyperparameters for DDQL algorithm

Parameter	Value
RL discount factor γ	0.9
Embedding iterations K	4
Vectors dimension d	64
Batch size	128
Initial epsilon ϵ_{start}	1
Final epsilon ϵ_{end}	0.3
Epsilon decay factor	300
Target update parameter τ	0.01, 0.005
Replay memory size	5000, 10,000, 20,000, 30,000
Initial learning rate lr_0	0.001
Learning rate decay	0.0001
EWC importance parameter λ	1000
EWC sample numbers N	500
SI strength parameter c	0.1
SI damping parameter ζ	0.001
Replay memory probability p	0.5
Replay memory previous memory size	5000, 10,000, 15,000

Table 2 Datasets for static networks

Dataset	Number of nodes	Number of edges
Highschool	120	348
Memetracker	960	4888

where lr_0 represents the initial value of the learning rate, and lr_steps is the number of steps performed during learning (parameter updates). Hyperparameter values are represented in Table 1.

Experiments on Static Networks

DDQL algorithm has been tested on two static real-world networks, Highschool[1] and Memetracker[2] datasets. These datasets are explained in the following, and a summary of their characteristics is illustrated in Table 2:

Highschool 2013 Dataset

This dataset contains contact information between high school students in Marseilles, France, during five days in December 2013, collected by the work in [45]. This dataset contains a total number of 120 nodes and is a directed network. In our research, this network has been converted to an undirected network by assuming a link between each pair of nodes if there is a directed link between them. This network contains edge weights that represent contact duration between nodes. These weights are reported as discrete values which are 1, 2, 3, or 4. In our research, these weights have been mapped to decimal values 0.25, 0.5, 0.75, and 1 considered as propagation probabilities. For computing these probabilities in the case of two directed links between nodes, the average edge weights have been considered as the weights of the undirected links.

[1] http://www.sociopatterns.org/datasets/high-school-contact-and-friendship-networks/.

[2] http://snap.stanford.edu/netinf/#data.

Table 3 Results for real-world static networks

Dataset	Method	Avg. discounted reward	Avg. seed length
High school	Deg.	92.29	8.08
	W. Deg. Disk.	93.14	**7.52**
	DDQL	**94.08**	9.68
Memetracker	W. Deg.	379.27	54.66
	W. Deg. Disk.	378.38	51.23
	DDQL	**380.10**	**50.79**

The numbers in bold font correspond to the performance of the best method regarding each metric

MemeTracker Dataset

This dataset contains news media sites and blogs as nodes and who-copies-whom relationships as edges and is obtained by the work of Gomez-Rodriguez et al. in [46]. This network contains 960 nodes and 5000 edges from 5000 phrase clusters which contain cascade information. This network is a directed network that an edge between two nodes represents that the destination node frequently copies from the source node. In our research, the network has been converted into an undirected network by considering a link between every two nodes if there is a directed link between them. This dataset contains the average propagation time between every two nodes, Δt_{uv} which is the average delay that the destination node v copies from the source node u. In our research, propagation probabilities are calculated by the formula $\alpha/\Delta t_{uv}$ [47]. The parameter α has been set to 0.1 as in [6, 47].

The results of the experiments on these datasets are presented in Table 3. This table represents average total (discounted) rewards and average seed lengths for heuristic and DDQL algorithms. As it is obvious from this table, the improvement is not significant using the DDQL method, and

Table 4 Synthetic dynamic networks characteristics

Node numbers	Community numbers	Snapshot numbers
50	3	70
100	5	120
200	7	220
300	10	350
500	10	600

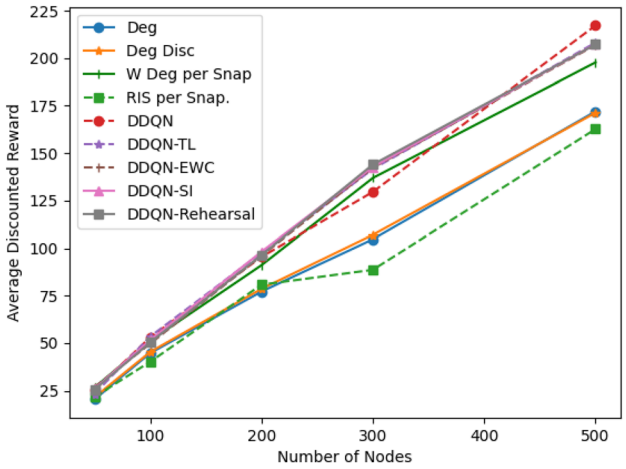

Fig. 2 Comparing average discounted rewards for different sizes of synthetic dynamic Markov SBM networks [48]

Table 5 Comparing average discounted reward for synthetic dynamic networks with different sizes (node numbers)

Method	50 N	100 N	200 N	300 N	500 N
Deg.	20.44	44.67	77.16	104.69	172.03
Deg. Disk.	21.42	45.33	78.60	107.01	171.28
W. Deg. per Snap.	26.50	51.21	91.07	136.93	197.74
RIS	21.56	42.36	79.95	99.83	160.36
RIS per Snap.	21.57	40.13	80.79	88.62	162.71
DDQL	25.10	53.03	95.43	129.59	**217.19**
DDQL-TL	23.23	**53.42**	97.60	141.66	208.16
DDQL-EWC	**26.76**	49.69	96.04	142.33	206.87
DDQL-SI	25.88	51.54	**98.20**	142.75	207.33
DDQL-Rehearsal	25.52	50.34	96.37	**144.02**	207.19

The numbers in bold font correspond to the performance of the best method for each network

a heuristic algorithm can achieve comparable results. Hence, according to computational complexity, the proposed algorithm is not efficient for static networks with known features.

Experiments on Dynamic Networks

Our proposed methods are DDQL method, DDQL with Transfer Learning (DDQL-TL), DDQL with EWC method (DDQL-EWC), DDQL with SI method (DDQL-SI), and DDQL with Rehearsal method (DDQL-Rehearsal), which have been tested using both synthetic and real-world dynamic networks. Details of these experiments and their results are explained in this section.

Synthetic Networks

For generating synthetic networks, different dynamic network models can be used which satisfy dynamic network properties explained in "Preliminaries and Influence Maximization" section. In this section, the employed dynamic network model and the obtained results are presented.

Table 6 Comparing average seed lengths for synthetic dynamic networks with different sizes (node numbers)

Method	50 N	100 N	200 N	300 N	500 N
Deg.	20.38	18.89	23.01	28.57	27.83
Deg. Disk.	20.55	18.01	22.43	26.95	27.61
W. Deg. per Snap.	19.44	17.38	20.58	24.12	26.18
RIS	19.66	20.96	20.29	25.38	26.38
RIS per Snap.	19.89	22	21.81	25.62	26.0
DDQL	17.32	15.1	17.96	21.62	23.22
DDQL-TL	18.56	**15.04**	17.85	19.53	**23.15**
DDQL-EWC	**16.44**	15.76	17.84	**19.35**	23.27
DDQL-SI	16.86	16.1	17.65	20.21	23.86
DDQL-Rehearsal	17.75	16.4	**17.18**	20.17	23.51

The numbers in bold font correspond to the performance of the best method for each network

The dynamic network model used in this research is the model introduced in [48] which combines stochastic block model [49] and Markovian property for edge dynamics and defines edge-state transition probability matrices P_{in} and P_{out} by the following formulas:

$$\begin{cases} P_{\text{in}}(1,1) = q_{\text{in}}, \\ P_{\text{in}}(0,1) = \mu_{\text{in}}(1-q_{\text{in}})/(1-\mu_{\text{in}}), \end{cases} \tag{23}$$

where $P_{\text{in}}(a,b)$ represents the intra-community probabilities of edge-state transition from a to b (edge state 1 means link establishment and 0 means no link), q_{in} is the within community link persistence probability, μ_{in} is within community link creation probability in the first snapshot. The inter-community parameters, $P_{\text{out}}(a,b)$ are also defined with corresponding parameters as well. To set more real parameters for synthesized networks, the dynamic network model parameters obtained from the Highschool dataset [45] are used for generating networks. The obtained parameters are: $\mu_{\text{in}} = 0.029$, $\mu_{\text{out}} = 0.000042$, $q_{\text{in}} = 0.19$, $q_{\text{out}} = 0.096$. Different networks have been generated using these parameters, and their details are represented in Table 4.

In order to train and test different methods (heuristic and DDQL methods), separate train and test sets (network snapshots) have been considered. In dynamic heuristic algorithms, 5–50 train snapshots were used to calculate dynamic degrees. For training the vanilla–DDQL algorithm, only one last snapshot in the training set was used and trained like a static network, but for incremental and transfer learning algorithms, two of the last training snapshots were considered, where the trained model on the first snapshot was used for training on the second snapshot. The rest of the snapshots

Table 7 Results of testing pre-trained replay memory DDQL models on larger networks (represented in columns)

Model	50 N	100 N	200 N	300 N	500 N
DDQL-Rehearsal-50 N	**27.26**	53.33	93.60	125.99	211.38
DDQL-Rehearsal-100 N	–	**53.94**	**97.66**	**143.44**	**222.27**
DDQL-Rehearsal-200 N	–	–	97.65	142.44	216.92
W. Deg. per Snap.	25.67	49.46	87.99	136.92	197.74

The numbers in bold font correspond to the performance of the best method for each network

First three rows represent the results of trained DDQL-Rehearsal models using networks with 50, 100 and 200 nodes respectively

Table 8 Datasets for dynamic networks

Dataset	# of nodes	Duration	# of snap	Snap. intervals (s)
Hospital	75	4.02 days	89	2500
Hypertext	113	2.46 days	136	900
Primaryschool	242	1.33 days	310	200
Highschool	327	4.21 days	363	400
Infectious	403	1.32 days	437	60

was considered the test set, where different snapshots were considered at each step of seed selection and propagation. Note that dynamic heuristics (except max degree per snapshot methods), in general by design, need more information (more snapshots) to give better results. However, our methods do not need to have more information in testing. The NN during training can capture some intrinsic properties of the network which is further used while testing.

The results of the experiments on synthetic networks are presented in Fig. 2 and Tables 5 and 6. These results represent the average of the obtained results over 100 test iterations. According to these results, the weighted degree heuristic at each network snapshot has the best performance among heuristic and RIS methods. The results represented in Table 5 confirm good performance of DDQL methods, which reveals their better performance for larger networks. For a network with 500 nodes, vanilla–DDQL method has the best average discounted reward around 9.8% improvement over the best heuristic method, as indicated in Table 5. The degradation of incremental and transfer learning methods for larger networks can result from the complexity of retraining the model for incremental (or transfer) learning in larger datasets. In addition, as illustrated in Table 6, average seed lengths of DDQL methods are also lower than in other methods, which represents another advantage of using DDQL methods.

To measure the extendability power of the proposed method, in another experiment, different models have been trained on smaller data using the replay memory incremental DDQL method. These models have been trained on synthetic networks with 50, 100 and 200 nodes and they have been tested directly on larger networks up to 500 nodes with the same dynamic network model parameters (μ and q values). The results of this experiment are illustrated in Table 7, which represents average discounted rewards of compared methods. These results have been compared with max weighted degree at each network snapshot heuristic algorithm which is the best heuristic algorithm among compared heuristics. As Table 7 presents, the pre-trained DDQL models outperform the heuristic method in all these networks. This demonstrates the reusability power of the DDQL method which decreases training overhead for large datasets.

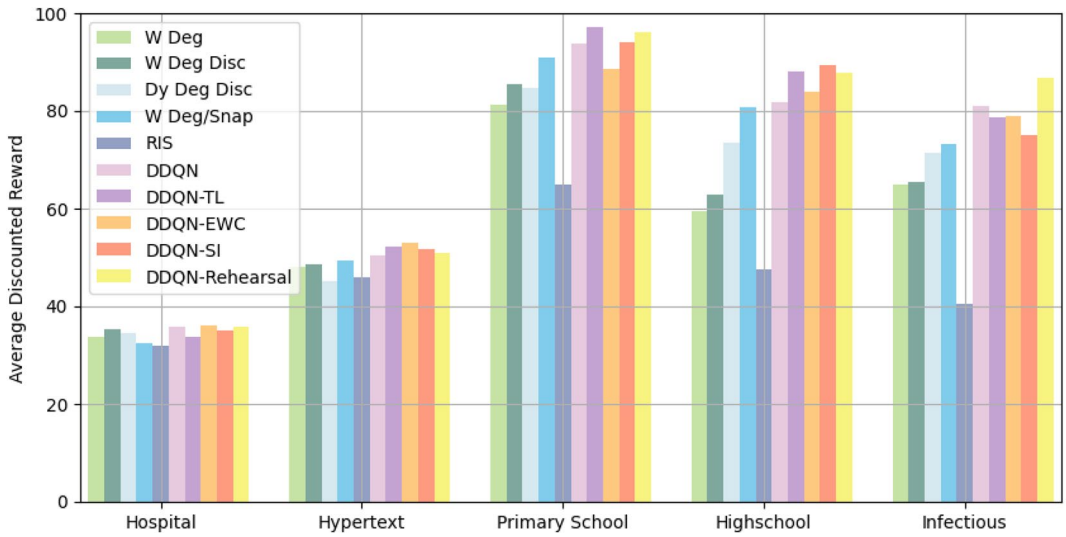

Fig. 3 Results for different real-world dynamic networks

Table 9 Comparing average discounted rewards for real dynamic networks

Method	Hospital	Hypertext	Primary school	Highschool	Infectious
W. Deg.	33.73	48.03	81.33	59.59	64.83
W. Deg. Disk.	35.39	48.70	85.55	62.82	65.48
Dy. Deg. Disk.	34.62	45.31	84.57	73.57	71.47
W. Deg. per Snap.	32.46	49.38	91.00	80.66	73.19
RIS	31.91	46.05	64.84	47.43	40.59
RIS per Snap.	31.06	45.64	50.86	52.47	39.88
DDQL	35.95	50.37	93.84	81.84	81.06
DDQL-TL	33.84	52.30	**97.11**	88.03	78.78
DDQL-EWC	**36.17**	**52.88**	88.63	83.78	79.03
DDQL-SI	35.16	51.57	93.99	**89.32**	75.07
DDQL-Rehearsal	35.84	50.98	96.03	87.68	**86.63**

The numbers in bold font correspond to the performance of the best method for each network

Table 10 Comparing average seed lengths for real dynamic networks

Method	Hospital	Hypertext	Primary school	Highschool	Infectious
W. Deg.	39.39	42.2	38.55	64.91	188.17
W. Deg. Disk.	30.92	34.94	34.73	61.93	164.56
Dy. Deg. Disk.	31.5	**33.65**	**32.82**	56.41	**157.91**
W. Deg. per Snap.	40.42	45.23	39.3	67.81	210.18
RIS	40.64	41.26	34.59	66.2	176.2
RIS per Snap.	39.01	42.25	47.65	70.36	194.21
DDQL	30.68	37.38	41.99	56.18	165.33
DDQL-TL	47.0	35.33	33.59	56.9	195.59
DDQL-EWC	**30.55**	34.42	64.0	57.57	197.76
DDQL-SI	31.8	35.25	43.01	56.92	201.71
DDQL-Rehearsal	32.33	34.39	41.61	**54.27**	193.55

The numbers in bold font correspond to the performance of the best method for each network

Real-World Networks

The proposed methods have been tested on different real-world dynamic networks, which are summarized in Table 8 and are explained in more detail in the following:

Hospital Dataset[3]

This dataset contains interaction information between 75 patients and healthcare workers during about 5 days (6–10 December 2010) in a hospital ward in Lyon, France.

Hypertext Dataset[4]

This dataset represents contact information between attendees in ACM Hypertext 2009 conference and consists of 113 nodes. This data has been collected during about 2.5 days.

Table 11 Extracted features of dynamic networks

Dataset	Avg. density	P(0,1)	P(1,0)
Hospital	0.0093	0.013	0.772
Hypertext	0.0072	0.006	0.91
Primary school	0.0069	0.005	0.828
Highschool	0.0022	0.002	0.825
Infectious	0.0008	0.0006	0.747

Primary School Dataset[5]

This dataset involves interaction information between students and teachers which is used in the study published in BMC Infectious Diseases 2014. This network contains 242 total nodes and has been collected during about 1.5 days.

[3] http://www.sociopatterns.org/datasets/hospital-ward-dynamic-contact-network/.

[4] http://www.sociopatterns.org/datasets/hypertext-2009-dynamic-contact-network/.

[5] http://www.sociopatterns.org/datasets/primary-school-temporal-network-data/.

Highschool 2013 Dataset[6]

This dataset is the dynamic version of the static High-school 2013 dataset containing 327 nodes which contains additional time interval information (every 20 s) in which every two nodes have been in contact.

Infectious Dataset[7]

This dataset contains face-to-face interactions between 405 participants of the 2009 SFHH conference in Nice, France during two days (June 4–5, 2009). This data was released by the publication [50].

To generate network snapshots from these datasets, aggregated networks in specified time intervals are constructed. The aggregation procedure is as follows: if there is an interaction between any two nodes, an edge is considered between them at each time interval. By this procedure, all network snapshots during the total interaction times were generated. For avoiding sparse graphs, the snapshots with a high number of zero-degree nodes ($\geq 90\%$ of nodes) were removed. The considered time intervals for each dataset are presented in Table 8. Random probabilities in the interval [0, 1] were generated for all datasets for propagation probabilities.

The results of testing different methods on real dynamic networks have been illustrated in Fig. 3 and Tables 9 and 10. As Fig. 3 and Table 9 demonstrate, except for the Hospital dataset, max weighted degree at each snapshot has the best discounted rewards for real datasets among heuristic and RIS methods. For these datasets, DDQL methods have better performance (in terms of discounted reward) than other methods. In addition, replay memory DDQL has better results for larger networks, by having about 18% improvement of average reward in Infectious data, over the best heuristic method (weighted degree at each snapshot).

Practical Consideration

In this section, we now analyze the efficiency of different methods on different types of data. In addition to the general features of datasets, we extract some additional features. The considered features include the size, average density and dynamic features of the networks, which are explained and measured in this section.

Regarding the network sizes, the experimental results of "Experiments on Dynamic Networks" section represent that the replay memory method performs better for larger real networks. This might be the result of more rigorous settings required for the other incremental methods in larger datasets and need more exploration for finding the best setting for them.

The other measured feature of the networks is the average density of the network during several snapshots. The density of a static network is defined by

$$\text{Density} = \frac{2|E|}{N(N-1)}, \tag{24}$$

where $|E|$ represents the number of edges in the network. For dynamic networks, this metric is evaluated on consecutive snapshots and averaged over them. The value of this metric for five real-world networks is measured and is given in Table 11. According to the values of this feature and the results of the average discounted rewards given in Table 9, the DDQL-Rehearsal method has better efficiency on sparse networks (Infectious dataset), whereas other methods (transfer learning, SI, and EWC) have better performance on more dense networks.

The other considered feature of networks is the amount of dynamicity of networks. As Tables 3 and 9 represent, our methods do not have significant improvement in static networks compared with heuristic methods, but they are more efficient in dynamic networks in terms of average reward values. Furthermore, the probabilities of edge appearance and disappearance in the aforementioned Markov model in "Experiments on Dynamic Networks" section are evaluated for different real networks using 50 snapshots of each dataset assuming no clustering and are reported in Table 11. It should be noted that because of considering different time intervals for constructing snapshots of different datasets, these metrics cannot be used for comparing the dynamicity of networks, but some features of different methods can be deduced according to snapshot dynamicity. Tables 11 and 9 represent that for the networks with higher transition probabilities (Hospital and Hypertext data), the EWC method achieves higher reward values (Table 9). In addition, the transfer learning method also represents good results in smaller and more dynamic network snapshots.

Complexity Analysis

The main motivation here is that the time complexity of a method devised for dynamic networks should not be very high and these methods should be able to perform in a dynamic environment according to demand. We now calculate the running time complexity of our methods, which is the time required for taking action at each step (i.e., for each snapshot of the network). This time constitutes the time required for embedding the network which is $O(N^2d + d^2N)$ for each iteration of the embedding process (Eq. 8) and is $O(K(N^2d + d^2N))$ for the entire embedding process. The

[6] http://www.sociopatterns.org/datasets/high-school-dynamic-contact-networks/.

[7] http://www.sociopatterns.org/datasets/sfhh-conference-data-set/.

complexity of calculating Q values (Eq. 10) is $O(d^2N)$ and finally selecting the best action (with maximum Q value) has $O(N)$ time complexity. Therefore, selecting an action has a total of $O(KN^2d)$ time complexity. Since K and d are set to 4 and 64 in the experiments respectively, and only N and probably d grow with the size of the network, by choosing an appropriate dimension, we can control the time complexity when implementing these algorithms for larger networks.

Conclusion

In this research, DQL and incremental learning-based methods for solving IM problem over dynamic networks are proposed. In the first proposed method, a DDQL method, based on the method of [6], is trained on only one network snapshot and is used on several snapshots. In this method, an immediate reward function is applied, which increases the method's efficiency (in terms of memory and time). In addition, this method has been extended using incremental learning methods, i.e., EWC, SI, and rehearsal, and a naive transfer learning algorithm for tackling dynamic data. Empirical results demonstrate the superiority of these methods over heuristic and RIS methods (in terms of total influence) for the growing size of the network. In addition, the use of incremental and transfer learning algorithms yields better results for real-world networks.

There are several avenues for future research, which are: (i) using more powerful and extendable incremental learning methods to learn from more network snapshots and capture more dynamic information, (ii) defining a more accurate reward function, which considers dynamics of the network, (iii) inferring dynamic network model in addition to optimization task, (iv) inferring network properties (structure and propagation probabilities) for unknown networks during IM task, (v) devising an extendable method for more extensive networks.

Acknowledgements This work received partial support from the Graph-Massivizer project funded by the European Union under grant agreement number: Project 101093202. This work has also been supported by the French government, through the RISE Academy of UCAJEDI Investments in the Future project managed by the National Research Agency (ANR) with the reference number ANR-15-IDEX-0001 and by the project of Inria - Nokia Bell Labs "Distributed Learning and Control for Network Analysis". Authors would also like to thank Dr. Saeid Pashazadeh and Dr. Javad Musevi Niya for their help and valuable comments on the paper.

Funding Open access funding provided by University of Klagenfurt.

Data Availability The MemeTracker dataset [46] is publicly available on the SNAP Stanford website (http://snap.stanford.edu/netinf/#data). The static network of Highschool 2013 dataset [45] and all the real-world dynamic networks used in this research i.e. Hospital, Hypertext, Primary School, Highschool 2013, and Infectious datasets) are publicly available on the website of SocioPatterns (https://www.sociopatterns.org/datasets). The synthetic networks can be provided by the corresponding author upon request (seyedehhaleh.seyeddizaji@aau.at).

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Leskovec J, Adamic LA, Huberman BA. The dynamics of viral marketing. ACM Trans Web. 2007;1(1):5-es. https://doi.org/10.1145/1232722.1232727.
2. Kempe D, Kleinberg J, Tardos E. Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining. KDD '03. New York: Association for Computing Machinery; 2003. p. 137–46. https://doi.org/10.1145/956750.956769.
3. Murata T, Koga H. Extended methods for influence maximization in dynamic networks. Comput Soc Netw. 2018;10:5. https://doi.org/10.1186/s40649-018-0056-8.
4. Huang S, Bao Z, Culpepper JS, Zhang B. Finding temporal influential users over evolving social networks. In: 2019 IEEE 35th international conference on data engineering (ICDE). 2019. p. 398–409.
5. Song G, Li Y, Chen X, He X, Tang J. Influential node tracking on dynamic social network: an interchange greedy approach. IEEE Trans Knowl Data Eng. 2017;29(2):359–72. https://doi.org/10.1109/TKDE.2016.2620141.
6. Dai H, Khalil EB, Zhang Y, Dilkina B, Song L. Learning combinatorial optimization algorithms over graphs. In: Proceedings of the 31st international conference on neural information processing systems. NIPS'17. Red Hook: Curran Associates Inc.; 2017. p. 6351–61.
7. Wijayanto AW, Murata T. Effective and scalable methods for graph protection strategies against epidemics on dynamic networks. Appl Netw Sci. 2019;04:4. https://doi.org/10.1007/s41109-019-0122-7.
8. Manchanda S, Mittal A, Dhawan A, Medya S, Ranu S, Singh A. Learning heuristics over large graphs via deep reinforcement learning. 2019. arXiv:1903.03332. [cs.LG].
9. Tian S, Mo S, Wang L, Peng Z. Deep reinforcement learning-based approach to tackle topic-aware influence maximization. Data Sci Eng. 2020;03:5. https://doi.org/10.1007/s41019-020-00117-1.
10. Lin SC, Lin SD, Chen MS. A learning-based framework to handle multi-round multi-party influence maximization on social networks. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. KDD

'15. New York: Association for Computing Machinery; 2015. p. 695–704. https://doi.org/10.1145/2783258.2783392.
11. Parisi GI, Kemker R, Part JL, Kanan C, Wermter S. Continual lifelong learning with neural networks: a review. Neural Netw. 2019;113:54–71. https://doi.org/10.1016/j.neunet.2019.01.012.
12. Kirkpatrick J, Pascanu R, Rabinowitz N, Veness J, Desjardins G, Rusu AA, et al. Overcoming catastrophic forgetting in neural networks. Proc Natl Acad Sci. 2017;114(13):3521–6. https://doi.org/10.1073/pnas.1611835114 (www.pnas.org/content/114/13/3521.full.pdf).
13. Zenke F, Poole B, Ganguli S. Continual learning through synaptic intelligence. In: Precup D, Teh YW, editors. Proceedings of the 34th international conference on machine learning. Proceedings of machine learning research, vol. 70. International Convention Centre, Sydney, Australia: PMLR; 2017. p. 3987–95. http://proceedings.mlr.press/v70/zenke17a.html.
14. Li Y, Fan J, Wang Y, Tan KL. Influence maximization on social graphs: a survey. IEEE Trans Knowl Data Eng. 2018;30(10):1852–72. https://doi.org/10.1109/TKDE.2018.2807843.
15. Leskovec J, Krause A, Guestrin C, Faloutsos C, VanBriesen J, Glance N. Cost-effective outbreak detection in networks. In: Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '07. New York: Association for Computing Machinery; 2007. p. 420–9. https://doi.org/10.1145/1281192.1281239.
16. Goyal A, Lu W, Lakshmanan LVS. CELF++: optimizing the greedy algorithm for influence maximization in social networks. In: Proceedings of the 20th international conference companion on world wide web. WWW '11. New York: Association for Computing Machinery; 2011. p. 47–8. https://doi.org/10.1145/1963192.1963217.
17. Wang Y, Cong G, Song G, Xie K. Community-based greedy algorithm for mining top-K influential nodes in mobile social networks. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '10. New York: Association for Computing Machinery; 2010. p. 1039–48. https://doi.org/10.1145/1835804.1835935.
18. Brin S. The PageRank citation ranking: bringing order to the web. In: Proceedings of ASIS, 1998; 1998. p. 161–172.
19. Freeman LC. Centrality in social networks conceptual clarification. Soc Netw. 1978;1(3):215–39. https://doi.org/10.1016/0378-8733(78)90021-7.
20. Morone F, Makse H. Influence maximization in complex networks through optimal percolation. Nature. 2015;06:524. https://doi.org/10.1038/nature14604.
21. Chen W, Wang Y, Yang S. Efficient influence maximization in social networks. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '09. New York: Association for Computing Machinery; 2009. p. 199–208. https://doi.org/10.1145/1557019.1557047.
22. Liu Q, Xiang B, Chen E, Xiong H, Tang F, Yu JX. Influence maximization over large-scale social networks: a bounded linear approach. In: Proceedings of the 23rd ACM international conference on conference on information and knowledge management. CIKM '14. New York: Association for Computing Machinery; 2014. p. 171–80. https://doi.org/10.1145/2661829.2662009.
23. Kimura M, Saito K. Tractable models for information diffusion in social networks. In: Fürnkranz J, Scheffer T, Spiliopoulou M, editors. Knowledge discovery in databases: PKDD 2006. Berlin: Springer; 2006. p. 259–71.
24. Chen W, Wang C, Wang Y. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '10. New York: Association for Computing Machinery; 2010. p. 1029–38. https://doi.org/10.1145/1835804.1835934.
25. Goyal A, Lu W, Lakshmanan LVS. SIMPATH: an efficient algorithm for influence maximization under the linear threshold model. In: 2011 IEEE 11th international conference on data mining; 2011. p. 211–20.
26. Rui X, Yang X, Fan J, Wang Z. A neighbour scale fixed approach for influence maximization in social networks. Computing. 2020;102(2):427–49. https://doi.org/10.1007/s00607-019-00778-5.
27. Cheng S, Shen H, Huang J, Zhang G, Cheng X. StaticGreedy: solving the scalability-accuracy dilemma in influence maximization. In: Proceedings of the 22nd ACM international conference on information and knowledge management. CIKM '13. New York: Association for Computing Machinery; 2013. p. 509–18. https://doi.org/10.1145/2505515.2505541.
28. Borgs C, Brautbar M, Chayes J, Lucier B. Maximizing social influence in nearly optimal time. In: Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms; 2014. p. 946–957.
29. Tang Y, Xiao X, Shi Y. Influence maximization: Near-optimal time complexity meets practical efficiency. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data; 2014. p. 75–86.
30. Nguyen H, Thai M, Dinh T. Stop-and-stare: optimal sampling algorithms for viral marketing in billion-scale networks. 2016. p. 695–710.
31. Lei S, Maniu S, Mo L, Cheng R, Senellart P. Online influence maximization. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '15. New York: Association for Computing Machinery; 2015. p. 645–54. https://doi.org/10.1145/2783258.2783271.
32. Habiba, Yu Y, Berger-Wolf TY, Saia J. Finding spread blockers in dynamic networks. In: Giles L, Smith M, Yen J, Zhang H, editors. Advances in social network mining and analysis. Berlin: Springer; 2010. p. 55–76.
33. Tong G, Wu W, Tang S, Du D. Adaptive influence maximization in dynamic social networks. IEEE/ACM Trans Netw. 2017;25:112–25.
34. Zhuang H, Sun Y, Tang J, Zhang J, Sun X. Influence maximization in dynamic social networks. In: 2013 IEEE 13th international conference on data mining; 2013. p. 1313–8.
35. Zhou C, Zhang P, Guo J, Zhu X, Guo L. UBLF: an upper bound based approach to discover influential nodes in social networks. In: 2013 IEEE 13th international conference on data mining. 2013. p. 907–16.
36. Kermack WO, McKendrick AG. A contribution to the mathematical theory of epidemics. Proc R Soc Lond Ser A. 1927;115(772):700–21.
37. Kermack WO, McKendrick AG. Contributions to the mathematical theory of epidemics—II. The problem of endemicity. Bull Math Biol. 1991;53(1):57–87. https://doi.org/10.1016/S0092-8240(05)80041-2.
38. Pastor-Satorras R, Castellano C, Van Mieghem P, Vespignani A. Epidemic processes in complex networks. Rev Mod Phys. 2015;87:925–79. https://doi.org/10.1103/RevModPhys.87.925.
39. Rossi RA, Ahmed NK. The network data repository with interactive graph analytics and visualization. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence. AAAI'15. Washington, DC: AAAI Press; 2015. p. 4292–3.
40. Hasselt H. Double Q-learning. In: Lafferty J, Williams C, Shawe-Taylor J, Zemel R, Culotta A, editors. Advances in neural information processing systems, vol. 23. Red Hook: Curran Associates, Inc.; 2010. https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

41. Hasselt Hv, Guez A, Silver D. Deep reinforcement learning with double Q-learning. In: Proceedings of the thirtieth AAAI conference on artificial intelligence. AAAI'16. Washington, DC: AAAI Press; 2016. p. 2094–100.
42. Sutton RS, Barto AG. Reinforcement learning: an introduction. Cambridge: A Bradford Book; 2018.
43. Polyak BT, Juditsky AB. Acceleration of stochastic approximation by averaging. SIAM J Control Optim. 1992;30(4):838–55. https://doi.org/10.1137/0330046.
44. Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks. In: Teh YW, Titterington M, editors. Proceedings of the thirteenth international conference on artificial intelligence and statistics. Proceedings of machine learning research, vol. 9. JMLR Workshop and Conference Proceedings, Chia Laguna Resort, Sardinia, Italy. 2010. p. 249–56. http://proceedings.mlr.press/v9/glorot10a.html.
45. Mastrandrea R, Fournet J, Barrat A. Contact patterns in a high school: a comparison between data collected using wearable sensors, contact diaries and friendship surveys. PLoS ONE. 2015. https://doi.org/10.1371/journal.pone.0136497.
46. Gomez-Rodriguez M, Leskovec J, Krause A. Inferring networks of diffusion and influence. ACM Trans Knowl Discov Data. 2012. https://doi.org/10.1145/2086737.2086741.
47. Khalil EB, Dilkina B, Song L. Scalable diffusion-aware optimization of network topology. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. KDD '14. New York: Association for Computing Machinery; 2014. p. 1226–35. https://doi.org/10.1145/2623330.2623704.
48. Avrachenkov K, Dreveton M, Leskelä L. Community recovery in non-binary and temporal stochastic block models. ArXiv. 2020; abs/2008.04790.
49. Holland PW, Laskey KB, Leinhardt S. Stochastic blockmodels: first steps. Soc Netw. 1983;5(2):109–37. https://doi.org/10.1016/0378-8733(83)90021-7.
50. G'enois M, Barrat A. Can co-location be used as a proxy for face-to-face contacts? EPJ Data Sci. 2018;7(1):11. https://doi.org/10.1140/epjds/s13688-018-0140-1.