A Learning-based Framework to Handle Multi-round Multi-party Influence Maximization on Social Networks

Su-Chen Lin
Dept. of Electrical Engineering
National Taiwan University
sclin@arbor.ee.ntu.edu.tw

Shou-De Lin
Dept. of Computer Science
and Information Engineering
National Taiwan University
sdlin@csie.ntu.edu.tw

Ming-Syan Chen
Dept. of Electrical Engineering
National Taiwan University
mschen@cc.ee.ntu.edu.tw

ABSTRACT

Considering nowadays companies providing similar products or services compete with each other for resources and customers, this work proposes a learning-based framework to tackle the multi-round competitive influence maximization problem on a social network. We propose a data-driven model leveraging the concept of meta-learning to maximize the expected influence in the long run. Our model considers not only the network information but also the opponent's strategy while making a decision. It maximizes the total influence in the end of the process instead of myopically pursuing short term gain. We propose solutions for scenarios when the opponent's strategy is known or unknown and available or unavailable for training. We also show how an effective framework can be trained without manually labeled data, and conduct several experiments to verify the effectiveness of the whole process.

Categories and Subject Descriptors

J.4 [**Computer Applications**]: Social and behavioral sciences—*Economics*

Keywords

Influence maximization; reinforcement learning; social network

1. INTRODUCTION

Influence maximization has been an eye-catching task in social network analysis for more than a decade. The goal is to identify a small subset of seed nodes that have the best chance to influence the most number of nodes through a given influence propagation process. The idea of viral marketing is widely accepted as a proper application scenario for influence maximization. A company intends to select a small set of seed customers in a social network to distribute their trial products, with the hope that these customers as a group spread their praises to a larger population through the word-of-mouth effect. The problem then becomes who to be selected as the initial seeds in order to maximize the final outcome. Domingos and Richardson [1,2] are the first to study influence maximization as an algorithmic problem. Kempe, et al. then formulate the problem as a discrete stochastic optimization problem [3]. They consider the independent cascade model and linear threshold model and prove that the optimization is NP-hard. They further prove that a greedy approximation algorithm guarantees the spread achieves at least (1-1/e) of the optimal spread. However, such a greedy model suffers from high computational complexity. Consequently, several works have been proposed to handle the efficiency problem [4–8].

KDD'15, August 11-14, 2015, Sydney, NSW, Australia.

DOI: http://dx.doi.org/10.1145/2783258.2783392.

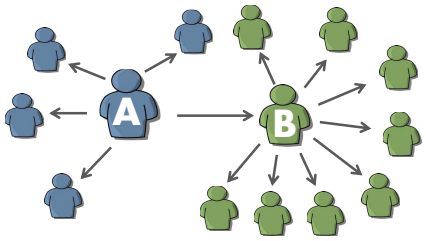
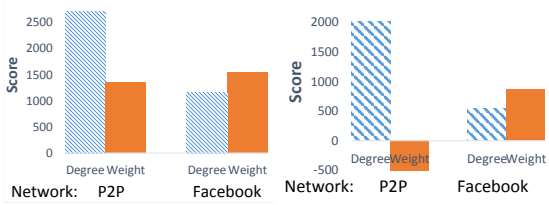


Figure 1: The example of competitive network

Competitive influence maximization (CIM) is a natural extension of influence maximization. Instead of considering only one type of innovations or items being propagated in the social network, competitive influence maximization or multi-party influence maximization assumes multi-items to be propagated in a social network. Furthermore, concurrent diffusions may interfere with each other as a single node can only be activated by one type of item (or by one party). For example, people generally only subscribe to one internet service provider (ISP). Thus for viral marketing purposes, two ISPs are forced to compete with each other to allure potentially influential customers to use their trial package. The same situation applies to companies that sell high-end electronic products as people are less likely to buy the same product if they already own one. In such competitive environments, the optimization strategy for single party may fail due to the existence of a competitor. Below we provide a toy example to elaborate this point.

EXAMPLE 1. *Given a single party in the network shown in Figure 1, the optimal choice is obviously the root node A assuming IC model of equal edge-weight. However, in CIM with two parties, each choosing one node in turn, the optimal choice is no longer node A. For the first player, the best choice is node B since if it chooses A, then the second player*



(a) Single party IM (b) Multi-party IM

Figure 2: The best strategy varies for different networks.

will choose B to block A's influence path. Thus, choosing B maximizes first player's expected influence while second player's optimal choice is node A.

Competitive diffusion optimization has been studied by several researchers [9–15]. Most of them consider less complex settings such as known opponent strategy and single-round of choice. Here we propose a general solution for a more complicated but realistic setup. That is, the opponent's strategy can be either known or unknown, while a company needs to make sequential decisions in multiple rounds to choose seeds with the objective to maximize its own influence in the long run. We assume the influence is exclusive, implying that once a node is influenced by party A, it cannot be influenced again by party B. For simplicity, in this paper we assume for each round all parties choose one node and then influence propagates before the next round of selection starts.

Our work aims at filling the following gaps in competitive influence maximization:

1. Most of the strategies proposed for competitive influence maximization assume the opponent's strategy is given, or consider the selections of the opponents being visible [12–15]. Such an assumption is problematic as in real world scenario it is very difficult to decipher the opponent's strategy, and in some situations one needs to make decision without complete information about the opponent's selection. To make things even more complicated, the opponents can dynamically change their strategy. Our solution, on the other hand, works for scenarios where opponents' strategies are known or unknown. If it is known, or unknown but available to compete with for multiple times, we propose to train a model to gradually learn a strategy to defeat the opponent's strategy. If the opponent's strategy is unknown and unavailable for training, then we propose a game-theoretical solution to seek the Nash equilibrium as the best strategy which a rational agent should act.

2. In many of the previous works, only single-round propagation is considered. That is, after the player and the opponent pick the seeds, the propagation starts and the final outcomes are generated. Here we consider a more general and probably more realistic case where the players can keep selecting seeds to activate after the previous round ends, until reaching a predefined threshold of rounds or until all the nodes are activated. Such a multi-round scenario makes the task even more interesting though difficult to solve, as one does not have to stick to one single strategy through all the rounds. Conceivably a more flexible and intelligent strategy that adjusts itself during the process has a better chance of beating a model that remains the same.

3. In most of the previous works, the influence maximization problem is solved by model-driven methods which, given a known influence propagation model (e.g. LT model), apply certain heuristic to choose the seed nodes in the network [12–15]. One concern for such strategies is that they are not adaptive to the network topology. For example, Figure 2(a) displays a single-player influence maximization experiment we conducted on two real-world networks: P2P and Facebook networks. The results show that Degree strategy (i.e., choosing high degree nodes as seeds) works better in P2P network while Weight strategy (i.e., choosing nodes whose overall weights of adjacent edges are maximal) works better in the FB network. Similar situations also happen on the multi-party influence maximization problem. In Figure 2(b), it shows that against the same opponent, in the P2P network, Degree is more effective but the opposite is observed in the Facebook network. Thus, we believe to excel in the competitive influence maximization task, the model has to be general and flexible enough to adapt not only the opponent's strategy but also the environment setting such as the underlying network topology and the diffusion model.

4. To design such a general framework, we resort to a learning-based approach. One critical issue of exploiting learning-based approach lies in the acquisition of data. Previously, data-driven strategies have been proposed to predict the influence paths, under the assumption that there is only one party trying to propagate the media in a social network [16–18]. In these studies, the real propagation records are exploited to construct an appropriate diffusion model. However, it is extremely hard to obtain the propagation record in a multi-party case, not to mention the underlying strategies these parties deploy. Chen et. al. once proposed a multi-party influence optimization game whose goal is to record the behavior of different parties through playing a chess-like game to optimize the activation of influenced nodes [19]. However, to use such data to train a learning model, one needs to collect playing record for many networks, which is not very cost-effective in particular when the network size becomes large. To improve the feasibility of building a learning-based model, here we propose a strategy to exploit the influence-simulation outcomes for learning an optimal policy that alleviates the requirement for human-playing data in training a model. To be more precise, our framework performs data generation and model learning at the same time. The data, which is the experience generated through simulation by applying the current model, will become the feedbacks to refine the model for better performance. In this sense the data sparsity issue can be overcome.

We propose the STrategy-Oriented Reinforcement-Learning based influence Maximization (STORM) framework that is capable of learning a good multi-party influence-maximization strategy. The framework utilizes arbitrary existing single-player influence maximization strategies as its actions, and finds the best policy to select them given the observed conditions. The rewards are designed as the gain of the influenced nodes compared to its opponents. The optimal strategy is learned through interacting with the network environment and the opponents. The actions are taken to optimize the expected accumulated rewards in the long term avoiding being myopic on short-term gain. We specifically consider the following three scenarios. The first scenario assumes the opponent uses a known strategy for every round, in which our model can gradually learn an optimal strategy to win. The second scenario assumes the component's strategy is unknown but available for training, in which our model can still learn a strategy through competing with it.

In scenario 3, the opponent's strategy is unknown and unavailable for training, we propose to model it as a rational and smart agent to maximize our own influence, which also guarantees to reach equilibrium. We design the STORM-Q, STORM-QQ, STORM-MM learning models to handle above mentioned scenarios. Our experiment results on real-world datasets demonstrate the usefulness of the model.

2. PROBLEM STATEMENT

In the competitive influence diffusion environment, the influences from different parties are assumed to propagate at the same time. We assume the media being propagated are exclusive, meaning if a node is activated by a party, it cannot be activated again by another party. This paper focuses on the competitive LT model (CLT) for experiments, but the proposed model shall be applicable to competitive IC model as well.

DEFINITION 1. COMPETITIVE LINEAR THRESHOLD (CLT) *CLT model is a multi-party diffusion model. Similar to the original LT model, in which a node v is activated by party i, denoted as P_i, at round t when $\sum_{u \in O_t^i} w_{u,v} > \theta_v$, where $w_{u,v}$ is the weight of node u to node v and O_t^i is the set of nodes that have been activated by P_i before t^{th} round. The θ_v is the activation threshold of v. Once v is activated by one party, it cannot be activated again by any party. When more than one party is eligible to activate the same node in the same round, node v will be activated by P_i which has the highest overall influence on node v, that is, $\forall_j \sum_{u \in O_t^i} w_{u,v} > \sum_{u \in O_t^j} w_{u,v}$. This is regarded as the majority rule of conflict.*

Note that the conflict rule is slightly different from those proposed by He et al. and Chen et al, in which the former always assigns tie-break to the negative party in the influence blocking maximization task and the latter breaks the tie based on a random experiment whose successful rate is proportional to the overall influence value to the target node [15, 19].

Then, we define the multi-round competitive influence maximization problem with CLT model as follows.

DEFINITION 2. MULTI-ROUND COMPETITIVE INFLUENCE MAXIMIZATION (MRCIM) *Given the information of network G and CLT diffusion model, each party P_i, $1 \leq i \leq n$, chooses a set of seed nodes N_t^i among G in turn and then performs influence propagation with CLT model at round t. Note that for each round, the influence only propagates to k-layer of neighbors (in our experiment, k=1) before the next round starts. The objective of each party is to maximize its overall relative influence F^i after T rounds, where F^i is the difference among activated nodes of different parties. That is, $F^i = |O_T^i| - \sum_{\forall j \neq i} |O_T^j|$, where T is the terminal round number.*

Note that since we are considering a multi-round scenario, the opponents' previous decisions can be regarded as observed, but their future decisions are unknown to our model.

Figure 3 summarizes the process of MRCIM. In the beginning of round t, each party P_i chooses N_t^i seeds in turn, and then the system performs CLT diffusion. Based on the outcomes of previous rounds, each party selects another N_{t+1}^i seeds to activate for the next round, and the process continues. After the predefined round-limit has reached, the overall influence of each party can then be evaluated.

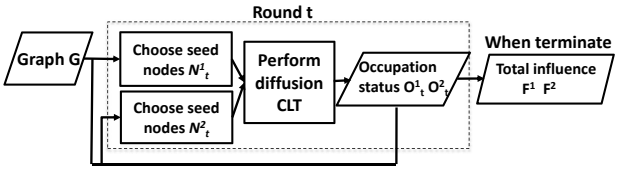


Figure 3: Multi-round multi-party system model

3. RELATED WORK

We have introduced model-driven and data-driven solutions for the single-party influence maximization problem in Section 1. Here we would like to focus on solutions for competitive influence maximization.

3.1 CIM with Opponent Strategy Known

There are several studies addressing the competitive influence maximization problem given known opponent strategy. One of the popular strategies is called the Influence Blocking Maximization (IBM), whose objective is to minimize the opponent's influence given the seed nodes already selected by the opponents [12, 13, 15]. Since IBM is proved to be NP-hard, these studies have provided greedy-based methods to solve it with either IC model or LT model. Borodin et al. and He et al. proposed the greedy methods in LT model [12, 15]. Furthermore, Budak et al. proposed the greedy methods in IC model [13]. They also studied IBM with uncertain data where each node's occupation status is stochastic. They use predictive hill climbing to predict the occupation status and then create a new instance of IBM. For these three studies, since the opponent's choice is assumed to be given and fixed, we can treat the opponents' choices as part of the predictable, observed information and search for the best response directly. This assumption might not be realistic in real world scenario where opponents are strategic, that is, their choices are neither known nor fixed.

3.2 CIM with Opponent Strategy Unknown

There are also solutions that assume the other party's strategy is not known and the opponent's choices cannot be observed before its own decisions are made. They have to 'guess' the opponents' choices and act accordingly. Bharathi et al. resort to game theory to analyze this problem [9]. They show that at least (1-1/e) of the optimal results can be achieved when the opponent's choices can be predicted accurately. However, predicting the opponent's choices is a very challenging task, which has not yet been addressed. Kostka et al. prove that the seed selection tasks of the first and the second parties are both NP-hard [10]. Although their analysis provides nice theoretical bounds on this problem, they do not propose a practical solution to tackle it. Also they only focus on single-round choices, without considering making sequential decisions in multiple rounds.

The most closely related work might be the one proposed by Tsai et al. [11]. They consider the situation where two parties have to make their choices without the opponents' choices in competitive diffusion networks. They propose an framework called Double Oracle. They use EXACT, APPROX, LSMI, and PageRank as the oracle strategies to estimate and block opponent's influence. One of the parties keeps iteratively updating its oracle strategy in turns to choose its seed nodes according to the opponent's temporal choice until their choices converge.

Different from our framework, this solution does not consider the multi-round scenario. In other words, the Double Oracle solution might not reach global optimal in multi-round cases as it myopically searches the best response in each round. Another major concern, similar to that in most of the model-driven approaches, is that there has not yet been a mechanism which is able to adapt to different types of networks and different opponent strategies. Table 1 summarizes the properties of the existing competitive influence maximization algorithms.

Table 1: Summary of CIM related work

feature	multi-party	unknown opponent	multi-round	data/model driven
[12, 13, 15]	V			Model
[9, 10]	V	V		Model
[11]	V	V		Model
Our paper	V	V	V	Data

4. METHODOLOGY

If the strategy of the opponent is fixed and known, the multi-round competitive influence maximization (MRCIM) can be proved as NP-hard by reducing it to a single party influence maximization, which has already been proven as an NP-hard problem [3]. Due to the NP-hardness of MRCIM, finding an optimal solution is impractical for real world problems. Therefore, we will look for an approximated solution. In addition, MRCIM is a repeated process through which all parties can obtain new occupation status next round. Thus the seed node selection of each round cannot be regarded as independent. Myopically maximizing the influence for each round may not maximize the overall influence in the end. We have to consider the impact of the selection for the current round in the long run.

4.1 Preliminary: Reinforcement Learning

We propose to exploit reinforcement learning (RL), a kind of machine learning technique based on Markov Decision Process (MDP), to tackle this task. In RL, the agent keeps interacting with the environment to find the optimal policy π that maximizes his expected accumulated rewards [20]. RL formulates the iterative process as an MPD and solves the optimization problem in a dynamic programming manner. Here we would like to first introduce the RL framework and then discuss how it can be exploited to tackle MRCIM.

The goal of an RL is to learn a policy $\pi(s)$ to determine which action to take given a certain environment represented by state s. RL formulates the expected accumulated rewards of a state (also called the V function) and the expected accumulated rewards given a state-action pair (also called the Q function) to estimate how good the policy π is in maximizing the accumulated reward R_t. The V function $V^\pi(s)$ given a policy π is defined as

$$V^\pi(s) = E_\pi\{R_t|s_t = s\} = E_\pi\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\}, \quad (1)$$

where γ is the discount factor. The Q function $Q(s, a)$ given the policy π is defined as

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\{R_t|s_t = s, a_t = a\} \\ &= E_\pi\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\}. \quad (2) \end{aligned}$$

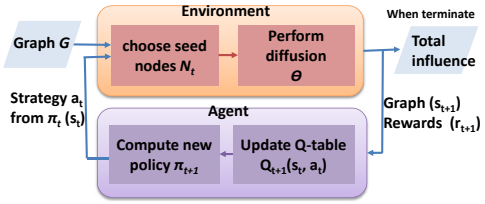


Figure 4: STORM framework in training phase

The two functions (1) and (2) can be learned through samples of agents interacting with the environment. The optimal policy $\pi(s)$ can be obtained given the Q function, that is, $\pi = \arg\min_{a\in\mathcal{A}} Q(s, a)$.

4.2 Strategy-Oriented Reinforcement-Learning

We propose the STrategy-Oriented Reinforcement Learning based influence Maximization (STORM) framework to learn a strategy for seed node selection. To simplify the analysis, here we assume there are only two parties that compete with each other. There are two parts of STORM we would like to discuss, the training phase (see Figure 4) and the competition phase (after a model is learned). To exploit the STORM framework, we need to first define the environment, the reward, the action and the state.

ENVIRONMENT We treat the influence propagation process as the environment effect, which propagates the influence of currently activated nodes to its neighbors and activate new ones.

REWARD The environment returns the reward r designed as the delayed reward to represent the difference of activated nodes between parties at the last round. That is, $r_t^i = 0$ when $t \neq T$, and $r_t^i = |O_T^i| - \sum_{\forall j\neq i}|O_T^j|$ when $t = T$. The rewards are then propagated to the previous states through Q-function updating. Although using the delayed rewards generally requires more iterations to converge, it usually converges to a more accurate Q-value compared to the immediate rewards $r_t^i = (|O_t^i| - |O_{t-1}^i|) - \sum_{\forall j\neq i}(|O_t^j| - |O_{t-1}^j|)$ [21].

ACTION The key of our model lies in the design of the action set A. Straightforwardly we can design actions as choosing a certain node to activate. However, it will lead to a very large action space to learn from. Also the key to the success of RL lies in its capability to learn under given states which action should be taken. With an overwhelming number of actions, it would then be hard to correlate the states with the actions, raising the issue of overfitting. Here we introduce the concept of meta-learning into RL framework by designing actions using a set of single-party information maximization strategies specifically sub-greedy, degree-first, blocking, max-weight in our experiments. Thus the size of action set can be reduced to the size of different strategies to be chosen (e.g., 4 in our experiments). That is, our agent chooses one strategy to perform as an action, while the actual seed to select is determined by this strategy. Note that to fully utilize the strength of our model, it is preferred to have diverse strategies that aim optimizing different aspects of the influence. The meta-learning framework eventually learns which strategy to choose given the current state. We call these basis strategies the *candidate strategies*. In this sense we can take advantage of any powerful IM strategy by using them as one of the candidate strategies to yield superior outcome.

STATE We need to model the *state* to represent the network and environment status. A straightforward way is to record the occupation status of all nodes as states. By doing so, the number of possible states can reach $3^{|V|}$ given there are two players (i.e., a node can be either activated by player 1 or player 2 or not yet activated). When $|V|$ is large, the problem becomes intractable since it is infeasible to either visit each state or store the Q-table. Also an overwhelming number of states can also lead to the problem of overfitting. Here we need to resort to the design of features to represent the current occupation status as well as the condition of the network. Note that when designing the features, we need to make sure they are explicitly or implicitly correlated with the rewards and the choice of actions. Below are the features we have designed:

1. Number of free (i.e., non-occupied) nodes
2. Summation of degrees of all free nodes
3. Summation of weight of the edges for which both vertices are free
4. Maximum degree among all free nodes
5. Maximum sum of free out-edge weight of a node among all nodes
6. Maximum sum of free out-edge weight of a node among nodes which are the first player's neighbors
7. Maximum sum of free out-edge weight of a node among nodes which are the second player's neighbors
8. Maximum activated nodes of a node for the first player after two rounds of influence propagation
9. Maximum activated nodes of a node for the second player after two rounds of influence propagation

We quantize each feature into three levels (i.e., low, medium, and high). Thus, we have at most 3^9 states. In practice, since some of the features are dependent, not all the combinations of states can happen. Note that these features are somewhat correlated with the candidate strategies used as our actions. For instance, it is not hard to imagine that if the max free degree is low, then it makes less sense to choose degree-first strategy.

DATA FOR TRAINING Since we are proposing a data-driven approach, normally it is required to acquire multi-party seed selection data to train our model. Theoretically we need some experience data (i.e., which action given which state can lead to what amount of rewards) to learn the V and Q functions. Unfortunately, as described previously, obtaining human-labeled seed selection data is costly, in particular for multi-party cases. Fortunately, since in our problem the propagation model is known (e.g., we choose LT in our experiments), and the candidate strategies served as actions are predefined, we can simply resort to agent simulation to obtain multi-round seed selection outcomes. That is, in the training phase we train the agent against a certain strategy (or fused strategies) and see how it performs on the given network. Such simulation data can then be used to learn the value functions. Training against opponents can be divided into three different scenarios:

S1. If the opponent's strategy is known, then we can simply simulate the opponent's strategy during training so that the agent can gradually learn how to play with it.

S2. If the opponent's strategy is unknown but available during training, then our agent can still play against it to learn how to optimize its own influence against it.

S3. The opponent's strategy is unknown and unavailable for training. In Section 4.4, we introduce how to train a more general model that has a better chance to defeat such opponents.

In summary, one important advantage for our design is that the data acquisition cost is very low as no human selection data is needed while training a model, which significantly boosts the usability of this solution.

The flow of our framework is shown in Figure 4. Given the social network and the design of the actions, states, and rewards, our agent keeps interacting with the opponent and environment to learn the association between actions, states, and rewards during training. The agent can gradually update its Q function (Q-table) from the simulation experiences throughout the training rounds, and in the meantime update its policy π for choosing seed nodes. Since the framework feedbacks the ultimate rewards back to each previous step to update its V function, we can expect the model to learn the true long-term value of each state-action pair, instead of simply optimizing the immediate rewards myopically. During the competition phase, the agent would not update its Q-table. Instead, it generates the policy π according to the existing Q-table. Next, we will describe in detail how to learn a model given the above three scenarios.

4.3 Strategy-Oriented Reinforcement-Learning with Opponent Strategy Known

We first consider an easier scenario, where the opponent's strategy is given or predictable. Note that predicting the strategy of an opponent is out of the scope of this paper. However, if the opponent's strategy is known, then it can be simulated exactly allowing our model to choose the maximum reward based on past experience. What we need to do is to let our model compete against this known opponent several times to learn the optimal policy against it. Here we propose an algorithm called STORM-Q which updates its Q-function following the concept of Q-Learning [21].

STORM-Q enters next state once the occupation status of the network is changed. We utilize the nine features in Sec. 4.2 to represent the current occupation status. Instead of choosing nodes directly, STORM-Q determines a node selection strategy from candidate strategies for each state. In the training phase, STORM-Q determines the strategy using ϵ-greedy on the current policy derived from Q-table. The ϵ-greedy is used to explore the new directions to avoid sticking at local optimum.

We can adapt Q-learning to the MRCIM problem given opponent's strategy. The idea of Q-learning allows us to approximate the V and Q functions without knowing in advance the state transition probabilities. It updates the functions one step at a time using current experience in the training phase. Since MRCIM is a finite round game, $Q_{t+1}(s_t, a_t)$ is guaranteed to converge to the optimal policy through Q-learning [22]. The $Q_{t+1}(s_t, a_t)$ is updated in the following.

$$Q_{t+1}(s_t, a_t) = (1-\alpha)Q_t(s_t, a_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1})\right], \quad (3)$$

where r_{t+1} is the relative number of activated nodes to the opponent's using strategy a_t in state s_t and $V(s_{t+1})$ is the expected accumulated influence gain from s_{t+1} to the end. The term of $r_{t+1} + \gamma V(s_{t+1})$ in (3) is a fresh expected accumulated reward obtained from this experience and $Q(s_{t+1}, a_{t+1})$ is the previous expected accumulated reward. The learning rate α is a weight to sum the fresh and

Algorithm 1 STORM-Q algorithm

```
Q(s, a) ← initial value
while training is not terminal do
    s_t ← s_0
    while s_t is not a terminal state do
        Determine strategy a_t using ε-greedy on the current
        policy π(s_t) derived from Q_t(s_t, a_t)
        Take strategy a_t to choose a seed node
        Simulate the opponent's seed choice
        Propagate influence to obtain reward r_{t+1}
        Compute network features as next state s_{t+1}
        Update Q(s_t, a_t) by (3) and s_t ← s_{t+1}
    end while
end while
```

previous expected rewards to derive a new expected accumulated reward $Q_{t+1}(s_t, a_t)$. Then the maximum reward from the learnt model is chosen to obtain the next state's expected accumulated reward $V(s_{t+1})$. That is, $V(s_{t+1}) = \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$. The training process then moves on to update state s_{t+1}. Note that STORM-Q adopts a *pure strategy*, meaning that the most likely strategy is chosen given a state. It is different from the *mixed strategy* that samples an action from the distribution of actions in each state. More details are given in Algorithm 1.

On the other hand, in the competition phase, Q-STORM greedily determines a strategy in each state without exploration on the current policy derived from Q-table.

4.4 Strategy-Oriented Reinforcement-Learning with Opponent Strategy Unknown

In the second scenario we consider the opponent's strategy unknown but available for training. Given this situation, we can still treat the opponent's selection as part of the environment feedback and apply STORM-Q to handle it. The only difference is that the experience cannot be obtained through simulation any more; instead, we have to train against the unknown opponent's strategy during competition. It is feasible because STORM-Q only needs to know the seed-selection outcomes of the opponent to update the Q-table, not the exact strategy it takes.

The most practical and challenging scenario might be the third one that assumes the opponent's strategy is unknown and unavailable for training. In the real world, we normally cannot obtain the competitors' promotion strategy and there is little chance to repeat a promotion campaign multiple times for training purposes. Here our goal is to create a general enough model that is competitive against a variety of rational strategies. Also we make a reasonable assumption that the opponent is rational. That is, it wants to optimize its own influence and knows that its opponents also want to optimize theirs when considering the opposite's strategy.

Under this assumption, the opponent chooses the node selection strategy with maximum rewards according to the experience, just like our agent. Then to conquer it, we propose STORM-QQ model. In STORM-QQ training, two identical STORM-Q agents are constructed to compete against each other and update their Q-tables at the same time. STORM-QQ interacts with each other according to the current Q-table during training phase. The other components are identical to STORM-Q. Note that STORM-QQ can be applied to more than two parties IM. If there exists only one equilibrium in MRCIM, STORM-QQ can reach the equilibrium by updating Q-table until convergence.

However, STORM-QQ consists of STORM-Q, which utilizes a pure strategy. It is not guaranteed that the equilibrium in MRCIM exists given a pure strategy is used which is adopted by STORM-QQ. For this reason, to find the equilibrium in MRCIM, it is preferable to use *mixed strategy* that samples an action from the distribution of actions in each state. Next, we will introduce a solution toward this goal.

We found that MRCIM is a zero-sum game for two players since $(F^1 - F^2) + (F^2 - F^1) = 0$, where $F^1 - F^2$ and $F^2 - F^1$ are the goals of P_1 and P_2, respectively, according to Definition 2. In a two-player zero-sum game, the Nash equilibrium is guaranteed to exist with mixed strategies and we can use MINMAX theorem to find the equilibrium [23]. To model it, we search for our best strategy considering the opponent also using the best strategy for itself. We use a payoff table similar to the Q-table while considering both parties' actions together. That is, $Q(s, a, o)$ is the reward obtained when the first party uses strategy a against the opponent's strategy o in state s. We propose the STORM-MinMax (abbreviated as STORM-MM) algorithm to learn an optimal strategy and apply the MINMAX-Q learning [24] to update the two-dimensional Q-table. The Q-table, $Q_{t+1}(s_t, a_t, o_t)$, in STORM-MM can be updated in the following manner.

$$Q_{t+1}(s_t, a_t, o_t) = (1-\alpha)Q_t(s_t, a_t, o_t) + \alpha\left[r_{t+1} + \gamma V(s_{t+1})\right] \quad (4)$$

where $V(s_{t+1})$ is the expected accumulated reward for the next state. It can be determined as the equilibrium value in $Q(s_{t+1}, a, o)$ with considering both agents' goals. The equilibrium value is derived by MINMAX theorem. The objective becomes finding such mixed strategies of the model itself π_s and the opponent's π_o.

For simplicity, the probability P_a represents the distribution of each action a taken by 1st-player and P_o represents the distribution of each action o taken by 2nd-player given s_{t+1}. The goal in STORM-MM is to find π_a that maximizes its expected accumulated reward while considering the worst case. That is, $\max_{\pi_a} \min_{\pi_o} \sum_a \sum_o p_a p_o Q(s_{t+1}, a, o)$. For the opposite agent, it wants to minimize the opposite's expected accumulated reward derived from the same Q-table. Its goal is to find $\min_{\pi_o} \max_{\pi_a} \sum_a \sum_o p_a p_o Q(s_{t+1}, a, o)$.

Since it has been proved that there exists equilibrium in a two-player zero-sum game, the equality of (5) holds [23].

$$\begin{aligned} V(s_{t+1}) = & \max_{\pi_a} \min_{\pi_o} \sum_a \sum_o p_a p_o (s_{t+1}, a, o) \\ = & \min_{\pi_o} \max_{\pi_a} \sum_a \sum_o p_a p_o (s_{t+1}, a, o) \end{aligned} \quad (5)$$

Since $P_o > 0$, $\max_{\pi_a} \min_{\pi_o} \sum_a \sum_o p_a p_o Q(s_{t+1}, a, o)$ can be reduced to $\max_{\pi_a} \min_o \sum_a p_a Q(s_{t+1}, a, o)$. Then we can transform this minmax problem to an optimization problem which can be solved with a linear programming.

Maximize $V(s_{t+1})$

Subject to

$$\forall_o \max_{\pi_a} \sum_a p_a Q(s_{t+1}, a, o) \geq V(s_{t+1}),$$

$$\forall_a p_a > 0, \text{ and } \sum_a p_a = 1.$$

After solving that, we obtain $V(s_{t+1})$ and π_a. The π_o can be obtained in the similar manner.

The flow of STORM-MM in the training phase is similar to that of STORM-Q except that the Q-function is updated by (5) in the line 7 of Algorithm 1.

The difference between STORM-QQ and STORM-MM is that in STORM-QQ there are two separate agents choosing strategies by separately maximizing the rewards in their own Q-tables; while STORM-MM finds equilibrium with one single Q-table and determines both sides' actions at the same time. Although STORM-QQ and STORM-MM choose their strategies using ϵ-greedy in the training phase, the goal of STORM-QQ is to learn pure strategies as a policy while STORM-MM learns mixed strategies. Thus, in the execution phase, STORM-QQ chooses a strategy by greedy (same as STORM-Q) while STORM-MM samples from the mixed strategy π_a or π_o.

Ideally, the policies learnt by well-trained STORM-QQ and STORM-MM are similar in two-party MRCIM. Since both parties in STORM-QQ simultaneously update their Q-tables, the stable action pair is also at equilibrium when the two Q-tables reach convergence. However, in practice their results might not be the same due to two reasons. First, the equilibrium may not exist in STORM-QQ (and thus may not converge) because it resorts to pure strategy. Second, though it is guaranteed that there exists equilibrium in two-party CIM with mixed strategies (i.e., STORM-MM), it does not guarantee that Q-table shall converge to a correct value. For example, lack of training data and bad initialization can all lead to the convergence in local optimum. If so, STORM-MM might not produce better results.

4.5 Complexity Analysis

We first analyze the time complexity of STORM in training and testing processes. One episode in training includes node selections and CTL propagation of T rounds. In Algorithm 1, one round of node selection includes five parts:

- $O(|A|)$: ϵ-greedy strategy determination in Line 5
- $\max_i O(x_i)$: executing a candidate strategy of choosing nodes in Line 6 and 7
- $O(|E| + |V|)$: performing propagation in Line 8
- $\max_j O(y_j)$: computing the state in Line 9
- $O(1)/O(|A|)$: updating the Q-table by STORM-QQ /STORM-MM in Line 10

Thus, for one episode in training, the time complexity is $O(T(\max_i x_i + \max_j y_j) + |E| + |V|)$ since the complexity of overall propagation is still $O(|E| + |V|)$ and A is a constant.

It is not hard to realize that the key to efficiency lies in candidate strategy executions $\max_i O(x_i)$ and state computations $\max_j O(y_j)$. In the experiment, we choose four candidate strategies, $Degree$, $Weight$, $Blocking$, and $SubGreedy$, which is bounded by $O(|E|)$. Note that the greedy strategy might not be a suitable basis strategy if efficiency is a concern, since its worst case cost is $O(|V||E|)$. Next, the computation of states varies a lot, depending on the complexity of each state. In our setting, Features 8 and 9 dominate the computation. Their computational complexity of the expected reward after propagating two rounds is $O(|E|)$. Hence, we have $\max_j O(y_j) = O(|E|)$.

Since $O(|E|) > O(|V|)$, the complexity of the training phase is $O(kT|E|)$, where k is the predefined number of episodes. For testing, since there is only one episode and the Q-table is not being updated, the complexity for testing is merely $O(k|E|))$, which is generally very efficient.

For space complexity, we need to store Q-tables with complexities of $|S||A|$ and $|S||A|^2$ for STORM-QQ and STORM-MM, respectively, where S is the set of the states which have been visited. In our experiment, $|A|$ is 4 .

5. EXPERIMENTS

To demonstrate the effectiveness of the proposed framework for MRCIM, we have designed experiments to test the following three hypotheses:

- H1. When the opponent's strategy is given, STORM-Q can be trained to optimize its performance against the opponents. Furthermore, while training and testing on the opponents of the same strategy, our model can do a better job than testing on opponents with different strategies to the training opponent.
- H2. When the opponent's strategy is unknown but available for training, our solution can gradually learn how to beat the opponent.
- H3. When the opponent's strategy is unknown and not available for training against, our solution can still yield better results than other solutions, including fixed strategies and meta-strategies such as voting.

5.1 Experimental Setting

Table 2: Statistics of real-world networks

Name	Node#	Edge#	Weight	Full graph
Ca-HepTh	9,877	51,971	given	Full
Ca-GrQc	5,242	28,980	given	Full
P2P-Gnutella	6,301	20,777	random	Full
Facebook	4,039	88,234	random	Full
Cit-HepPh	620	827	given	Subgraph

We choose four kinds of networks for evaluation, including social network, collaboration network, peer to peer network, and citation network with their details listed in Table 2. All datasets are downloaded from the Stanford Large Network Dataset Collection website [25]. Facebook is a network with social circles from the popular website Facebook.com. Ca-GrQc and Ca-HepTh are collaboration networks of Arxiv. P2P-Gnutella is a snapshot of Gnutella peer to peer network taken on August 8, 2002. Cit-HepPh is a subgraph of Arxiv High Energy Physics paper citation networks, which is created using Breadth-First-Search on the original citation network. We subsample Cit-HepPh to a much smaller graph since we want to use this to perform exhaustive search on all combinations of strategies to evaluate the scenario of unknown opponent strategy. Since the edge-weights are not given in the original datasets, we define the edge weight in Ca-HepTh, Ca-GrQc and Cit-HepPh as the number of co-authored papers, normalized by the sum of edge weight incident to a node. The weights of the other graphs, P2P-Gnutella and Facebook, are assigned randomly. The thresholds in the CLT model are set randomly.

An episode in our model is defined as the parties choosing seed nodes and influence being propagated for $T = 7$ rounds. In each round, the occupation status of nodes in the network is represented through the defined states. Our model then learns which strategy is the best to apply in each state. We use four heuristic algorithms, Degree, Weight, Blocking and SubGreedy, as candidate strategies.

- Degree: choosing a node with maximum number of free neighbors as the seed.
- Weight: choosing a node with maximum sum of free out-edge weight as the seed.
- Blocking: choosing among the blocking nodes (i.e. neighbor nodes of the opponent) whose sum of free out-edge weight is greatest.
- SubGreedy: A greedy strategy that chooses a node which, after being activated, yields the most activated nodes after two rounds of propagation, assuming the opponent takes no action. The reason to choose SubGreedy is that it is much more efficient than greedy which requires performing propagation till no new node can be activated.

We want to test how our model performs against different types of opponents. The above four strategies are the natural opponents we would like to compete with. Thus, we have Degree, Weight, Blocking, and SubGreedy as four fixed-strategy components. We also compare ourselves with two meta-strategies which also use a fusion of these four candidate strategies. The 'Random' meta-strategy randomly chooses one of these four strategies to select the seed. The 'Voting' meta-strategy let these four strategies vote for a node as the seed node.

If not specifically mentioned, we have the following training setting for each STORM model. During training phase, since the cost of data simulation is low, we independently train five models each running 500 episodes and choose the best one from the the results as our final model. It can be considered as a validation process for learning to eliminate the negative effects of poor initialization. The other parameters, initial learning rate α, initial ϵ-greedy probability ϵ , the decay of learning rate d, and discount factor γ are setted as 0.5, 0.8, 0.998, and 0.98, respectively. All experiments are conducted on a machine with Intel Core i7 3.4GHz 6 cores 64 bit processor and 64G RAM.

5.2 Experimental Results

In the first experiment, we assume that the opponent's strategy is given. We separately train STORM-Q against the given six opponents to obtain six different models. Then for each model we test it against six different opponents. Conceivably if testing on the strategy not yet trained against, the performance can be degraded. Table 3 shows the results in five networks. Due to space limitation, we only show the results of three networks with the average results of all five networks. Each row represents our model trained with a certain strategy (in bracket). Each column represents an opponent. The value inside represents the difference between activated nodes to the opponent in competition, coming from averaging five repetitions of different threshold settings. In our experiment we always train our model as the 1st-player to act first, although there is no reason it cannot be trained as the 2nd-player. Note that the learnt strategy for the 1st-player and 2nd-player can be different, thus the absolute score here might be slightly biased since the 1st player has some edge in the competition by choosing first. Nevertheless, the goal here is to show that our framework learns well with given opponents' strategies compared with training against arbitrary strategies. It generally outperforms the 2nd player if trained with the right strategy.

We can see that the score in the diagonal cells is usually the highest except for Random. This implies training

(a) P2P-Gnutella

Ours \ Opponent	Degree	Weight	Blocking	SubGreedy	Random	Voting
ST-Q(Degree)	**102.5**	26.0	24.0	-27.0	42.5	44.5
ST-Q(Weight)	75.0	96.5	90.5	-32.5	67.5	94.5
ST-Q(Blocking)	91.0	60.0	**123.0**	-52.0	48.0	46.0
ST-Q(Subgreedy)	75.0	60.5	88.0	**9.0**	54.5	85.0
ST-Q(Random)	87.0	**100.0**	91.5	-29.0	**70.5**	68.0
ST-Q(Voting)	67.5	21.0	71.5	-51.0	47.5	**106.5**

(b) Ca-HepTh

Ours \ Opponent	Degree	Weight	Blocking	SubGreedy	Random	Voting
ST-Q(Degree)	**112.5**	11.0	94.8	-48	20.3	39.5
ST-Q(Weight)	101.0	**67.8**	119.0	9.0	74.8	70.8
ST-Q(Blocking)	100.3	30.0	**129.5**	-12.8	42.8	59.5
ST-Q(Subgreedy)	94.5	55.0	105.3	**17.0**	**94.8**	65.3
ST-Q(Random)	107.8	62.0	121.0	6.8	71.5	63.8
ST-Q(Voting)	85.5	44.0	113.0	-14.8	54.0	**86.5**

(c) Cit-HepPh

Ours \ Opponent	Degree	Weight	Blocking	SubGreedy	Random	Voting
ST-Q(Degree)	**89.6**	5.8	91.0	2.0	51.8	27.2
ST-Q(Weight)	60.6	**35.2**	97.2	17.4	57.4	47.2
ST-Q(Blocking)	59.2	3.2	**146.4**	2.6	44.6	17.6
ST-Q(Subgreedy)	77.2	32.6	92.0	**30.6**	43.0	34.6
ST-Q(Random)	75.2	34.6	116.6	32.8	**71.2**	42.4
ST-Q(Votin)g	56.6	29.0	107.0	16.6	50.8	**49.6**

(d) Average of Score in Six networks

Ours \ Opponent	Degree	Weight	Blocking	SubGreedy	Random	Voting
ST-Q(Degree)	**85.2**	22.1	73.3	-10.8	37.0	27.0
ST-Q(Weight)	56.3	**58.9**	77.4	3.8	51.5	42.7
ST-Q(Blocking)	55.7	32.3	**108.2**	-12.8	30.8	28.6
ST-Q(Subgreedy)	55.9	37.1	73.6	**18.5**	50.0	45.5
ST-Q(Random)	70.8	46.4	95.9	11.5	**66.4**	44.9
ST-Q(Voting)	61.1	40.1	81.9	-1.4	42.2	**60.3**

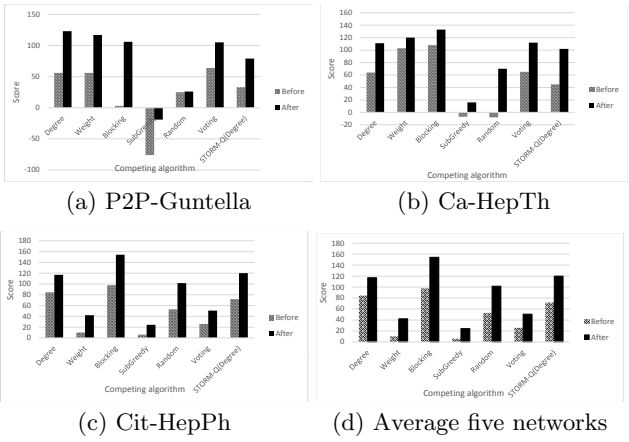
Table 3: Each row represents STORM-Q trained with a certain strategy (in bracket). Each column represents an opponent.

a model against the strategy that will be used to compete later on yields more effective results. Since Random strategy yields unpredictable choices , it is generally harder to train a good strategy against. Also we find that although SubGreedy seems to be the strongest strategy, training against it might not yield a killer model against all opponents. This experiment Our findings from this experiment confirms hypotheses H1.

In the second experiment, we want to test how our model performs when the opponent's strategy is unknown but available for training. We focus on verifying H2. To do so, we use STORM-Q to train against some unknown strategy for 500 episodes in five networks, and then report the results of our model against it before and after training in Figure 5. We average the outcomes of five networks in Figure 5(d).

The backslash and solid bars are the scores of STORM-Q against different strategies before and after training with these strategies. We notice that no matter which strategy the unknown opponent uses, can STORM-Q eventually learn an effective model to play against it, achieving higher score after training. Even against the strongest strategy SubGreedy, STORM-Q improves the score after training.

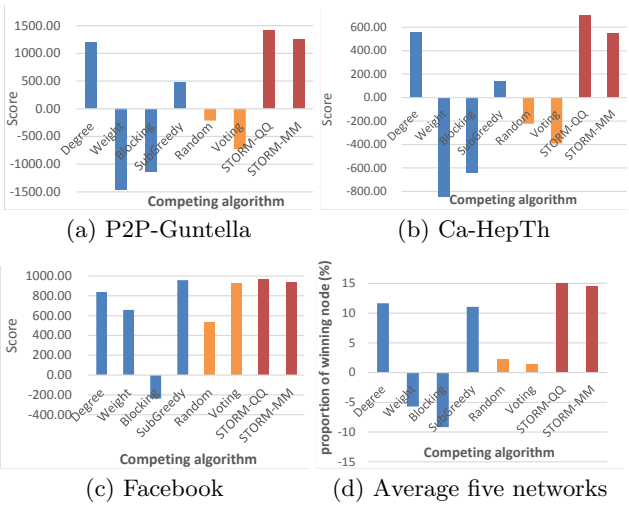
In the third experiment, we test the scenario that the opponent's strategy is unknown and unavailable for training.



(a) P2P-Guntella (b) Ca-HepTh

(c) Cit-HepPh (d) Average five networks

Figure 5: Score of STORM-Q training against different strategies before and after training



(a) P2P-Guntella (b) Ca-HepTh

(c) Facebook (d) Average five networks

Figure 6: Average scores of each competing algorithms against six strategies and an average score over five networks

It can also be regarded as that the opponents have freedom on which strategy to choose. Thus the design goal of the model becomes to optimize the influence given smart opponents. To evaluate such a goal, we design two scenarios.

The first scenario requests each competitor to compete one by one against the six aforementioned strategies (i.e. four fix strategies and two meta-strategies) and average the results. Here again we only show the results of three graphs and the average results of all graphs in Figure 6. The bar in Figure 6(d) represents the average score normalized by the node number in a network. Ideally an agent that obtains the better score can be considered as more capable of handling different types of opponent strategy.

In Figure 6, we observe that the best strategies among four basis strategies for networks with different topology vary significantly. In P2P-Gnutella and Ca-HepTh networks, Degree is the most effective strategy among four candidate strategies. However, SubGreedy yields the best performance in Facebook network. The results again reassure the need of a topology-dependent strategy. Though the most effective strategy is different for different networks, STORM-QQ and

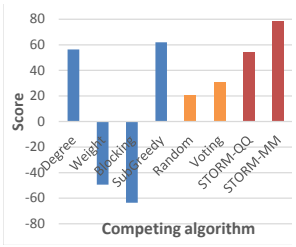


Figure 7: The average performances of the competing algorithms against all possible combinations of candidate strategies

STORM-MM generally perform better and both are able to learn different policies given different networks. Figure 6(d) reveals apparent advantage of our models while averaging the results of five networks.

In the second scenario, we provide a thorough evaluation on the generalization of the model. Using a smaller graph Cit-HepPh, we tried all possible combinations of candidate strategies in T rounds (in our experiments, $T = 7$, that sums up to 4^7 combinations of fused strategies). We let each competing algorithm play against all combinations and average their outcomes. The result is shown in Figure 7. It reveals that STORM-MM is the best strategy that has best chance to defeat different opponent strategies.

In summary, the third experiment verifies H3. We consider many types of strategies to model the possible unknown opponents' strategies. When the opponent's strategy is unknown and not available for training, STORM-QQ and STORM-MM generally yields better performance than fixed strategies and other types of meta-strategies.

5.3 Discussion

STORM can support more than two parties though we apply the simple setting of two parties in Sections 4 and 5. STORM updates the reward and makes decisions according to two factors: the responses of other parties and the influence propagation outcome. However, those factors are modeled as the environment's feedback from RL point of view. That is to say, no matter how many parties are there, their actions and consequent effects can be modeled jointly. STORM-Q constructs a Q-table in which the values represent the difference between its reward and the sum of other parties' rewards. Thus there is no extra cost of learning for more than two parties. The same reason applies to STORM-QQ, in which each party constructs its own Q-table. The complexity does not grow with the number of parties (not even linearly). Only constant loading is added for node selections and propagation.

STORM also supports the selection of more than one seed node each round by letting each basis strategy in STORM pick the top k nodes as the seed nodes. Similarly, the budget constraints can be implemented in each basis strategy, while the whole learning process remains the same.

6. CONCLUSION

The goal of this paper is not to design a new strategy for IM, instead we design a framework that generalizes a fused strategy that allows us to combine existing strategies to produce better results. To our knowledge, this is the first ever proposal that brings a marriage of reinforcement

learning and game theory to network influence maximization. Among the scenarios we have discussed, the more practical yet challenging is the situation where the opponent is unknown and unavailable to train against. We thus design a game-theoretic, learning-based framework to handle such a situation with theoretical guarantee to reach Nash equilibrium. Note that STORM can be exploited to handle single party IM as well. We can simply assume the opponents exploit the strategy of 'doing nothing' to learn a fused strategy adaptive to different networks.

Furthermore, STORM can be applied to solve more general IM task. It can learn from different diffusion models, even when the models are unknown as long as available during training. It can be modified to consider a situation that the network is not fully observed and can only be obtained through sampling (i.e., exploration during seed selection). In this situation, we have to model the expected gain of exploration in STORM.

Currently STORM is trained given one specific environment setting including single network topology, single diffusion model and single opponent strategy. In the future we would like to investigate whether it is possible to quickly transfer a learnt model to different settings, rather than learning from scratch. We also want to investigate the partial-observed MDP (POMDP) algorithms to handle the situation when the diffusion or selection outcomes are stochastic.

7. ACKNOWLEDGMENTS

This work was supported by Ministry of Science and Technology, National Taiwan University and Intel Corporation under Grants MOST 103-2911-I-002-001 and NTU-ICRP-104R7501 and grant MOST 103-2221-E-001-038-MY2, 102-2923-E-002-007-MY2, 102-2221-E-002-170, 103-2221-E-002-104-MY2.

8. REFERENCES

[1] P. Domingos and M. Richardson, "Mining the network value of customers," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2001.

[2] M. Richardson and P. Domingos, "Mining knowledge-sharing sites for viral marketing," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2002.

[3] D. Kempe, J. Kleinberg, and E. Tardos, "Maximizing the spread of influence through a social network," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2003.

[4] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2010.

[5] A. Goyal, W. Lu, and L. V. S. Lakshmanan, "Simpath: An efficient algorithm for influence maximization under the linear threshold model," in *IEEE International Conference on Data Mining*, 2011.

[6] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance, "Cost-effective outbreak detection in networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2007.

[7] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2009.

[8] K. Jung, W. Heo, and W. Chen, "Irie: Scalable and robust influence maximization in social networks," in *IEEE International Conference on Data Mining*, 2012.

[9] S. Bharathi, D. Kempe, and M. Salek, "Competitive influence maximization in social networks," in *Workshop on Internet and Network Economics*, 2007.

[10] J. Kostka, Y. A. Oswald, and R. Wattenhofer, "Word of mouth: Rumor dissemination in social networks," in *International Colloquium on Structural Information and Communication Complexity*, 2008.

[11] J. Tsai, T. H. Nguyen, and M. Tambe, "Security games for controlling contagion," in *Association for the Advancement of Artificial Intelligence*, 2012.

[12] A. Borodin, Y. Filmus, and J. Oren, "Threshold models for competitive influence in social networks," in *Workshop on Internet & Network Economics*, 2010.

[13] C. Budak, D. Agrawal, and A. E. Abbadi, "Limiting the spread of misinformation in social networks," in *International World Wide Web Conference*, 2011.

[14] W. Chen, A. Collins, R. Cummings, T. Ke, Z. Liu, D. Rincon, X. S. snd Yajun Wang, W. Wei, and Y. Yuan, "Influence maximization in social networks when negative opinions may emerge and propagate," in *SIAM International Conference on Data Mining*, 2011.

[15] X. He, G. Song, W. Chen, and Q. Jiang, "Influence blocking maximization in social networks under the competitive linear threshold model," in *SIAM International Conference on Data Mining*, 2012.

[16] T.-T. Kuo, S.-C. Hung, W.-S. Lin, S.-D. Lin, T.-C. Peng, and C.-C. Shih, "Assessing the quality of diffusion models using real-world social network data," in *International Conference on Technologies and Applications of Artificial Intelligence*, Nov 2011.

[17] C.-H. Tsai, J.-K. Lou, W.-C. Lu, and S.-D. Lin, "Exploiting rank-learning models to predict the diffusion of preferences on social networks," in *International Conference on Advances in Social Network Analysis and Mining*, 2014.

[18] Goyal, Amit, Bonchi, Francesco, Lakshmanan, and L. V. S., "A data-based approach to social influence maximization," *Proc. VLDB Endow.*, 2011.

[19] H.-H. Chen, Y.-B. Ciou, and S.-D. Lin, "Information propagation game: a tool to acquire human playing data for multiplayer influence maximization on social networksg," in *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2012.

[20] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[21] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge University, 1989.

[22] C. Watkins and P. Dayanr, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[23] J. von Neumann and O. Morgenstern, "Theory of games and economic behavior," in *Princeton University Press*, 1944.

[24] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *International Conference on Machine Learning*, 1994.

[25] [Online]. Available: http://snap.stanford.edu/data/