On Influential Node Discovery in Dynamic Social Networks

Charu C. Aggarwal[*] Shuyang Lin[†] Philip S. Yu[‡]

Abstract

The problem of maximizing influence spread has been widely studied in social networks, because of its tremendous number of applications in determining critical points in a social network for information dissemination. All the techniques proposed in the literature are inherently static in nature, which are designed for social networks with a fixed set of links. However, many forms of *social interactions* are *transient* in nature, with relatively short periods of interaction. Any influence spread may happen only during the period of interaction, and the probability of spread is a function of the corresponding interaction time. Furthermore, such interactions are quite fluid and evolving, as a result of which the topology of the underlying network may change rapidly, as new interactions form and others terminate. In such cases, it may be desirable to determine the influential nodes based on the dynamic interaction patterns. Alternatively, one may wish to discover the most likely starting points for a *given infection pattern*. We will propose methods which can be used both for optimization of information spread, as well as the backward tracing of the source of influence spread. We will present experimental results illustrating the effectiveness of our approach on a number of real data sets.

Keywords: Social Networks, Influence Analysis

1 Introduction

Social networks have traditionally been studied in the context of static (or slowly evolving) networks such as *Facebook* or *LinkedIn* [22, 23]. This is because the topology of such networks is defined by friendship links which evolve slowly over time. However, many social networks may be defined in the form of *transient interactions* between entities. In such cases, edges may be rapidly added to and deleted from the network, as a result of which the topology of the network may vary drastically over time. Many natural social interactions such as epidemiological networks, email networks or chat networks can be modeled much more naturally from a dynamic perspective. In this paper, we study the significantly more challenging problem of information spread through *transient* social interactions. Some examples of such applications are as follows:

- In an epidemiological network, the interaction between the entities may be very dynamic and transient. In such cases, influence analysis can help provide an understanding of the most potent infection points, and also the most likely infection points for a particular pattern of infection.
- The interactions of a given social network of entities can often be predicted based on past patterns or written calendar records in some cases. These represent future interactions, which can be used in order to model the spread of information.
- In an online chat network, it may sometimes be possible to model periodic patterns of interaction between different entities. This can help determine the key points in the network for information release.

The problem of dynamic information flow analysis is a fundamentally more difficult problem, because of the rapidly changing network pattern of interactions. In such cases, the more influential points are likely to not only be well connected entities, but also need to be connected to other entities in a *temporally strategic way* for maximum influence. The dynamic aspect of the problem makes the problem significantly more challenging, because of the complicated interplay between the structural and temporal aspects of the underlying network.

We will design a stochastic approach to determine the information flow authorities with the use of a *globally optimized forward trace approach*, and a *locally optimized backward approach*. The former approach is more accurate, though it comes at the expense of greater computational costs. We will also present methods in order to determine the most likely release points for a *given pattern* of information spread. We present experimental results illustrating the effectiveness and efficiency of the approach.

This paper is organized as follows. In the remainder of this section, we will discuss related work. In section 2, we introduce the dynamic model for information flow in social networks. In section 3, we will show how to use this model in order to determine the most influential points in the network with the use of a forward algorithm. In section 4, we will present a locally optimized backward trace method for optimizing the dynamic influence. We will also present a method for finding the most likely release points for a given pattern of infection. The experimental results are discussed in section 5. Section 6 contains the conclusions.

[*] IBM T. J. Watson Research Center, Hawthorne, NY, charu@us.ibm.com

[†] University of Illinois at Chicago, Chicago, IL, slin38@cs.uic.edu

[‡] University of Illinois at Chicago, Chicago, IL, psyu@cs.uic.edu

1.1 Related Work The problem of epidemic spread in computer networks has been studied extensively in [5, 12, 11, 13, 14, 15, 19, 20, 21]. Much of this work studies patterns of information flows under which such flows becoming epidemics. Some recent research [21] studies the information propagation problem in context of similar models for computer virus and epidemic spreading [15, 19]. The work in [5, 17, 16, 21] studies the information flows in the context of social networks and other kinds of computer networks. A method for mining the network value of customers for viral marketing model was proposed in [8].

Influential points are closely related to central points of network clusters. Many algorithms have been designed for determining clusters and communities [2, 3] in massive graphs. A number of influence maximization methods for the case of static networks were proposed in [1, 7, 6, 9]. All these previous papers handle only the static case of a fixed network, and are designed to determine the influential nodes on the basis of *steady state* behavior of the network. This is the first paper which addresses the problem of information flow authority determination in dynamic networks *on the basis of transient interactions*. This is a much more challenging and realistic assumption in most real scenarios. Furthermore, this paper examines both the problems of optimizing influence, and that of tracing back from a given pattern of spread.

2 Modeling Influential Nodes in Dynamic Social Networks

We assume that the universal set of nodes over which the social network is defined at time t is denoted by $N(t)$, and the edge set by $A(t)$. The edge set is assumed to be directed, since information flows are specific to direction in the most general case. We note that both the node set $N(t)$ and edge set $A(t)$ are time-dependent. Therefore, the underlying time-dependent graph $G(t)$ is denoted by $(N(t), A(t))$. In addition, we define the set of edges $E(t_1, t_2)$ which appear in the interval (t_1, t_2) as the union of all the edges which appear at any time t in (t_1, t_2). Therefore, we have:

$$E(t_1, t_2) = \cup_{t \in (t_1, t_2)} A(t) \tag{2.1}$$

In many dynamic applications, the edge set $A(t)$ may change drastically over time, as different kinds of transient interactions occur in the network. For example, the meeting of two actors corresponds to the creation of an edge, and their separation corresponds to the deletion of the edge. We assume that the probability of information spread along an edge (i, j) is given by the probability $f_{ij}(\delta t)$, where $f_{ij}(\delta t)$ is an increasing function of δt, and δt is the total amount of time for which the edge was present in the network. For example, a typical example of a transmission function could be:

$$f_{ij}(\delta t) = a \cdot (1 - e^{-\lambda_{ij} \cdot \delta t}) \tag{2.2}$$

In this case, we have $0 \leq a \leq 1$. Therefore, the steady-state probability of transmission is a, though the transmission probability is likely to be much lower for the transient case, when the edge is present in the network only for a small amount of time. The parameter λ_{ij} controls the transmission rate across an edge (i, j). The matrix of transmission functions is denoted by $F(\cdot) = [f_{ij}(\cdot)]$.

The set of nodes from which an incoming edge is incident into node i at time t is denoted by $I(i, t)$. In other words, we have $I(i, t) = \{k : (k, i) \in A(t)\}$. The set of nodes on which the outgoing edges of i are incident are denoted by $O(i, t)$. Therefore, we have $O(i, t) = \{k : (i, k) \in A(t)\}$. We assume a model of information transmissibility, in which a node i which contains a piece of information can transmit it along a transient edge (i, j) lasting for time δt with probability $f_{ij}(\delta t)$. In addition, we also define the incoming and outgoing edges for a *time interval* (t_1, t_2) as the union of all edges which appear in this interval. We define the notations $IN(i, t_1, t_2)$ and $OU(i, t_1, t_2)$, in order to denote the horizon specific incoming and outgoing edges. Therefore, we have $IN(i, t_1, t_2) = \cup_{t \in (t_1, t_2)} I(i, t)$ and $OU(i, t_1, t_2) = \cup_{t \in (t_1, t_2)} O(i, t)$.

One of the common steps that all algorithms in this paper need to repeatedly perform is to determine the transmission probabilities of the edges over different time intervals. Specifically, we need to compute the transmission probability of an edge over an arbitrary time interval (t_1, t_2). We denote the transmission probability along edge (i, j) between the time period (t_1, t_2) by $p_{ij}(t_1, t_2)$. We note that this transmission probability can be derived from the length of the period between (t_1, t_2) for which the edge (i, j) was present. We note that since the edge (i, j) may occur multiple times in the interval (t_1, t_2), we need to compute the overall transmission probability for that period. Let δt_1, δt_2, $\ldots$, δt_r be the time periods in (t_1, t_2), at which the edge (i, j) was present.[1] Then, the probability of transmission is given by $1 - \prod_{m=1}^{r}(1 - f_{ij}(\delta t_m))$. For example, an interesting special case is the memory less function when $f_{ij}(t)$ is $1 - e^{-\lambda_{ij} t}$. In such a case, the transmission probability evaluates to $1 - e^{-\sum_{m=1}^{r} \lambda_{ij} \cdot \delta t_m} = f_{ij}(\sum_{m=1}^{r} \delta t_m)$. In other words, we simply need to evaluate the function over the sum of the periods in (t_1, t_2) over which the edge is present.

The matrix of transmission probabilities in period (t_1, t_2) is denoted by $P(t_1, t_2)$ We denote the corresponding matrix of transmission probabilities by $P(t) = [p_{ij}(t_1, t_2)]$. Each value of $p_{ij}(t_1, t_2)$ is defined with the use of the underlying parameters. We note that this matrix is extremely

[1] For simplicity in this example, we assume that the edge (i, j) is not present at time t_1. In the event that edge (i, j) is present at time t_1, then its transmission probability would be defined by $f_{ij}(t_1 + \delta t_1 - t_a) - f_{ij}(t_1 - t_a)$. Here $t_a < t_1$ is the time of arrival of the edge (i, j), which continues to be present at time t_1. The modeling in this paper does not assume any such simplifying assumption.

sparse, because it is often overlaid on very sparse graphs such as social networks. We note that if $\pi(i, t_1)$ be the probability that a given node i contains a given piece of information at time t_1, then the probability that the the adjacent node j contains the information $\mathcal{I}$ at time t_2 is given by the probability $\pi(i, t_1) \cdot p_{ij}(t_1, t_2)$.

The problem of influence analysis in *dynamic networks* can be modeled in two ways, depending upon the goals of the analysis:

- We wish to pick the k points at time t_1, which would result in the maximum influence at time t_2.
- We wish to pick the most likely points of influence at time t_1 for a particular pattern of influence at t_2.

In this paper, we will study both cases. The first problem of influence maximization is defined as follows:

PROBLEM 1. *Determine the set S of k data points in the time interval $(t_0, t_0 + h)$ at which release of the information bits $\mathcal{I}$ at time t_0 would maximize the expected number of nodes over which $\mathcal{I}$ is spread at time $t_0 + h$.*

Such a problem can be very challenging in a rapidly changing network. This is because the spread probabilities may need to be potentially recomputed continuously as new social connections (edges) arrive in the network.

Let $\pi(i, t)$ be the probability that a particular piece of information is available at node i at time t. We note that the node i contains the information at time t_2, if it either already contained the information at time t_1 or if it transmits the information between times t_1 and t_2. Therefore, we have:

$$\pi(i, t_2) = \pi(i, t_1) + \\ +(1 - \pi(i, t_1)) \cdot (1 - \prod_{j \in IN(i, t_1, t_2)} (1 - \pi(j, t_1) \cdot p_{ji}(t_1, t_2)))$$

The above equation can be explained as follows. The probability at time period t_2 is expressed as the sum of the probabilities of two events: **(1)** In the first case, the node already contains the information at time t_1. The probability of this is $\pi(i, t_1)$, which is the first term above. **(2)** In the second case, the node does not contain the information at time t_1 with probability $(1 - \pi(i, t_1))$, but the information is transmitted to it by one of its neighbors. In the second term above, the expression $\prod_{j \in IN(i, t_1, t_2)} (1 - \pi(j, t_1) \cdot p_{ji}(t_1, t_2))$ represents the probability that none of the incoming edges of node i in the period (t_1, t_2) transmit to it. Therefore, the probability of transmission by at least one of the neighboring nodes is given by $1 - \prod_{j \in IN(i, t_1, t_2)} (1 - \pi(j, t_1) \cdot p_{ji}(t_1, t_2))$.

The generic problem requires us to determine a set S of k nodes, such that if the information is released at the time t_0, the total amount of information spread is maximized at time $t_0 + h$. Therefore, Problem 1 can be formally restated as follows:

Problem 1 (Formal Restatement): *Determine the set S of nodes at which to release the information at time t_0, which maximizes the probabilistic spread $\sum_{i \in N(t_0+h)} \pi(i, t_0+h)$.*

```
Algorithm DynInfluenceVal(Initial Set: $S$,
      Transmission Matrix: $F(\cdot)$, Time Horizon: $(t_0, t_0 + h)$)
begin
  for each $i \in S$ set $\pi(i, t) = 1$ and 0 otherwise;
  Divide $(t_0, t_0 + h)$ into time periods $t_0 \ldots t_r = t_0 + h$;
  $j = 0$;
  repeat
    Compute edge set $E(t_j, t_{j+1})$ and transmission matrix $P(t_j, t_{j+1})$;
    for each $i$ do
      Compute $\pi(i, t_{j+1})$ from $\pi(\cdot, t_j)$ using dynamic update equations;
    $j = j + 1$;
  until($j = r$);
  return( $\sum_i \pi(i, t_r)$);
end
```

Figure 1: Evaluating Influence for Fixed Node Set

3 Dynamic Influential Node Discovery

In order to determine the most influential nodes, we first design a method for evaluating the influence of a *given* set of k nodes S, which is referred to by the subroutine $DynInfuenceVal$. The input to this algorithm is the set S of nodes, a time-interval $(t_0, t_0 + h)$, and a transmission matrix F, which contains the time-dependent functions for influence spread. We note that our earlier equations for dynamically updating the probabilities at a given time-period from those in the previous time-period can be very useful for this purpose. However, we cannot apply this equation *at one time* between periods t_0 and $t_0 + h$, as this would not approximate the *proper sequence* of infections among nodes, because of the transient edges occurring between t_0 and $t_0 + h$. In order to obtain the most accurate possible result, the time interval $(t_0, t_0 + h)$ needs to be discretized into infinitesimally small intervals, and the probabilistic derivation of $\pi(i, t_2)$ needs to be constructed on the basis of the edges in this period. However, this is unlikely to be practical for real applications, and we need to discretize into much larger intervals as an approximation.

One possibility for discretization is to use equal length intervals. However, this is not very useful in cases where the structure of the network evolves at different rates over time. This is because the rate of edge-additions and deletions may vary over time. Ideally, if the edges in the network change more rapidly, then the discretization interval should be smaller. Therefore, we opt for a *non-uniform* discretization methodology, in which we successively generate the time-intervals based on the level of structural change in the underlying network. The approach is to start at the time-interval t_0, and then generate $t_1, t_2, \ldots t_r = t_0 + h$ *suc-*

cessively, in which the time-point t_{q+1} is generated from t_q based on the level of evolution in the interval (t_q, t_{q+1}). In order to do so, we define the structural evolution of the graphs $G(t_a)$ and $G(t_b)$ at two time periods t_a and t_b, where $t_b \geq t_a$.

DEFINITION 1. (STRUCTURAL EVOLUTION) *The structural evolution evolution level $Q(t_a, t_b)$ of graph $G(t_a) = (N(t_a), A(t_a))$ to the graph $G(t_b) = (N(t_b), A(t_b))$ is defined as follows:*

$$Q(t_a, t_b) = 1 - \frac{|A(t_a) \cap A(t_b)|}{|A(t_a) \cup A(t_b)|} \tag{3.3}$$

We note that the value of $Q(t_a, t_b)$ lies between 0 and 1. The greater the value of $Q(t_a, t_b)$, the greater the level of evolution from t_a to t_b. Furthermore, as $t_b - t_a$ increases, the value of $Q(t_a, t_b)$ will typically increase, though this may not always be the case because of noise, especially after $t_b - t_a$ already becomes large.

We compute the time t_{q+1} from t_q by computing the level of evolution between these periods, and using a threshold η in order to quantify this evolution. The value η is referred to as the *evolution threshold*. The time period t_{q+1} is generated from t_q as follows:
The value of t_{q+1} is determined as the smallest time at which $Q(t_q, t_{q+1})$ is greater than η.
The only exception to the above rule is the case for which t_{q+1} is larger than the end of the horizon $t_0 + h$. In that case, we set t_{q+1} to $t_0 + h$, and terminate the discretization process. We note that the use of smaller values of η results in a larger number of intervals. While this is more accurate, it will come at the expense of the greater amount of computation, because a greater granularity of the network requires us to compute the probabilistic influence analysis over a larger number of periodic intervals.

We will describe the approach for finding the dynamic spread in a given time interval $(t_0, t_0 + h)$ with the use of the afore-mentioned discretization of intervals. The first step is to construct the new set of discretized intervals, which are divided at the time points $t_0, t_1, \ldots t_r$. We start off with the counter $j = 0$, and set the value of $\pi(i, t_0)$ to 1 if i is in S and 0 otherwise. The value of j is incremented in each iteration as the edges in the next temporal interval are processed. In each iteration, we process the time-interval (t_j, t_{j+1}) and update $\pi(i, t_{j+1})$ from $\pi(i, t_j)$ with the use of the afore-mentioned dynamic update equations. We first generate the edge set $E(t_j, t_{j+1})$ and the transition probabilities $P(t_j, t_{j+1})$. These edge sets and transition probabilities are used in conjunction with the dynamic update equations in order to generate the probabilities $\pi(\cdot, t_{j+1})$ from $\pi(\cdot, t_j)$. This process is repeated for each of the discretized time periods from 1 through r. The overall algorithm is described in Figure 1 and is denoted by $DynInfluenceVal$.

```
Algorithm ForwardInfluence(Transmission Matrix: $F(\cdot)$,
          Time Horizon: $(t_0, t_0 + h)$, #InfluencePoints: $k$);
begin
  for each node $i \in N(t_0)$ do
     compute $DynInfuenceVal(\{i\}, F(\cdot), (t_0, t_0 + h))$;
  $S$ =Initial set of $k$ authority nodes with the
    highest $DynInfuenceVal(\{i\}, F(\cdot), (t_0, t_0 + h))$;
  for each node $i$ in $N(t_0) - S$ in descending
    order of $DynInfuenceVal(\cdot)$ do
  begin
    Find a node $j$ in $S$, such that
      $DynInfuenceVal(S \cup \{i\} - \{j\}, F(\cdot), (t_0, t_0 + h))-$
        $DynInfuenceVal(S, F(\cdot), (t_0, t_0 + h))$ is
      as large (positive) as possible;
    if $DynInfuenceVal(S \cup \{i\} - \{j\}, F(\cdot), (t_0, t_0 + h)) \geq$
         $DynInfuenceVal(S, F(\cdot), (t_0, t_0 + h))$ then
       begin; $S = S \cup \{i\} - \{j\}$; $NoReplace = 0$; end
     else $NoReplace = NoReplace + 1$;
    if $NoReplace \geq Min\_Iter$ then break;
  end
  return($S$);
end
```

Figure 2: Forward Algorithm

3.1 Forward Influence Algorithm The forward-influence algorithm uses a greedy approach in combination with forward temporal analysis in order to determine the most influential points in the network. The algorithm is denoted as *ForwardInfluence* and is described in Figure 2. The input to the algorithm is the matrix of transmission functions $F(\cdot)$, the time horizon $(t_0, t_0 + h)$, and the number of influence points k which need to be determined. The overall algorithm works by maintaining a current set S of influential points and continually improving it over time with the use of successive replacement. The algorithm starts off with a set S of k nodes, which are picked on the basis of the flow value from those individual nodes. Specifically, for each node $i \in N(t_0)$, we evaluate its dynamic influence in the interval $(t_0, t_0 + h)$ with the use of our previous algorithm *DynInfluenceVal*. The initial set S is the set of the nodes with the largest flow value from the entire node set.

The first step is to the evaluate the dynamic influence spread from each individual node in $N(t_0)$ with the use of the procedure *DynInfluenceVal*. We note that the *DynInfluenceVal* procedure is inherently a procedure which uses forward analysis for computing the total flow value. The individual nodes in $N(t_0) - S$ are sorted in reducing value of the influence spread. Subsequently, the algorithm proceeds iteratively in which we pick each successive node i from $N(t_0) - S$, and determine whether it is possible to replace some node $j \in S$ in order to increase the total influence spread in $(t_0, t_0 + h)$. We check each node $j \in S$ and determine the maximum influence *increase* on replacing the node j with i. In the event that at least one such node $j \in S$

exists for which the increase is positive, then we replace the node j with i. In the event that no such node exists, then we do not perform the replacement. We maintain a counter which tracks the number of consecutive times by which the replacement is not performed and increment it by one, if a replacement is not performed. We check if the replacement is not performed for at least *min_iter* consecutive iterations. If this is the case, then we terminate and report the current set of nodes S. On the other hand, if the termination criterion is not met, then we move on to the next node in $N(t_0) - S$ in the sorted order and check it with the different nodes for replacement. This process is continually repeated, until either all nodes in $N(t_0) - S$ have been visited, or no exchange is performed for at least *min_iter* consecutive iterations. The overall algorithm is illustrated in Figure 2. The forward algorithm essentially starts with a set S of nodes and continuously improves it over time with this iterative procedure. The use of a sorted order in order to perform the iterative exchange ensures that only a small number of iterations are typically required to termination.

4 Backward Influence Algorithms

The forward influence algorithm is somewhat slow in practice because it requires us to repeatedly estimate the influence spread from different starting points. Therefore, we will design faster approximations which use backward analysis in order to determine approximate release points. In *backward analysis*, we use a desired result at $t_0 + h$ in order to trace the influence results back to the appropriate nodes. One advantage of this class of methods is that it can not only be used in order to determine the (approximately) optimal influence points, but it can also be used to *retrospectively determine the most likely release points for a specific pattern of influence* or determine the best release points for a *desired pattern of influence*. Such an analysis can be very useful in a variety of dynamic applications, in which it is costly to release information at too many nodes, and it is desirable to influence only a subset of the nodes. Therefore, in the next subsections, we will propose two variations of the backward influence algorithm which are applicable to different scenarios:

- **Retrospective Version:** We attempt to determine the nodes which result in a specific pattern of influence.
- **Maximization Version:** This version is similar to the problem addressed by the forward algorithm.

Although the maximization version of the backward algorithm is the same problem as what we have already discussed, we will take the unusual step of presenting the retrospective version of the backward algorithm first because of the common notations and mathematical influence equations which it shares with the forward maximization algorithm. This allows for ease in exposition. We note however, that the applications of these methods are for different scenarios, and only the latter is directly comparable to the forward influence algorithm.

4.1 The Backward Influence Algorithm: Retrospective Version In this section, we will present the retrospective version of the backward algorithm. In this version, we assume that the pattern of influence is already known, or a specific pattern of influence is desired. Let us assume that the set of the nodes which are influenced (or desired to be influenced) at the time $t_0 + h$ is denoted by $D \subseteq N(t_0 + h)$.

As before, we assume that the time-interval (t_0, t_0+h) is discretized into intervals with the use of segmentation points at $t_0, t_1 \dots t_r = t_0 + h$. Since the goal is to find the most likely release points, we will work the probabilities back from $t_0 + h$ to t_0 using these discretized intervals, and the afore-mentioned relationship which relates two successive discretized intervals t_m and t_{m+1}:

$$\pi(i, t_{m+1}) = \pi(i, t_m) + (1 - \pi(i, t_m)) \cdot (1 - \prod_{j \in IN(i, t_m, t_{m+1})} (1 - \pi(j, t_m) \cdot p_{ji}(t_m, t_{m+1})))$$

We start off by setting $\pi(i, t_r) = \pi(i, t_0 + h)$ to $1 - \epsilon$ if $i \in D$, and to ϵ if $i \notin D$. Here ϵ is the *smoothing probability* and is typically a small value in $(0, 1)$. The idea of the smoothing probability is to construct a set of probability values at time $t_0 + h$, which are highly biased towards the set D. A natural question arises as to why a smoothing probability is needed at all, and we do not simply use the deterministic probability values at time $t_0 + h$ by setting $\epsilon = 0$. We note that if we set $\epsilon = 0$, the system of equations above would result in identical solutions for $\pi(i, t_m)$ for all the different values of m. This does not help in distinguishing the different nodes *on the basis of transmission*, which is our goal in this paper. The use of a small smoothing value of ϵ helps in distinguishing the different nodes on the basis of transmission. The idea is to start off with $\pi(i, t_r)$ and use it to derive $\pi(i, t_{r-1})$. The latter is used to derive $\pi(i, t_{r-2})$. The process is repeated for $\pi(i, t_{r-3}) \dots \pi(i, t_0)$. The process of generating $\pi(i, t_m)$ from $\pi(i, t_{m+1})$ is referred to as a *major iteration*, and is achieved by solving the system of equations above. As we will discuss below, the process of solving for $\pi(i, t_m)$ from $\pi(i, t_{m+1})$ in each major iteration is a challenge in of itself, as it requires us to solve a non-linear system of equations. Once we have generated the set of probabilities for $\pi(i, t_0)$, we have an estimate for the most likely set of nodes which have resulted in the corresponding pattern of influence. In that case, we report the k nodes with the highest value of the probability $\pi(i, t_0)$ as the set of nodes at which the information should be released. The overall algorithm is

```
Algorithm RetrospectiveSources(Transmission Matrix: $F(\cdot)$,
   Influence Pattern: $D$, Time Horizon: $(t_0, t_0 + h)$,
         #InfluencePoints: $k$);
begin
  Discretize the interval $(t_0, t_0 + h)$ into
    $r$ different points $t_0, t_1 \ldots t_r = t_0 + h$;
  for each $i \in D$ set $\pi(i, t_r) = 1 - \epsilon$ and $\epsilon$ otherwise;
  for $m = r - 1$ down to 0 do
  begin { Start Major Iteration }
    Generate the system of probabilities $\pi(\cdot, t_m)$ from $\pi(\cdot, t_{m+1})$ using
      the relationships of probabilities between successive intervals with
      the use of multiple minor iterations;
  end; { End Major Iteration }
  return $k$ nodes with largest value of $\pi(i, t_0)$;
end
```

Figure 3: Backward Trace: Retrospective Version

illustrated in Figure 3. The input parameters to the algorithm include the transmission matrix $F(\cdot)$, the influence pattern D, the time horizon, and the number of influence points k. We note that the influence pattern D is an additional input to the algorithm, as compared to the forward algorithm.

One challenge in the approach discussed above is the derivation of $\pi(i, t_m)$ from $\pi(i, t_{m+1})$, since the system of equations discussed relating the two highly non-linear in terms of unknown variable $\pi(i, t_m)$, and it is linear only in terms of the known variable $\pi(i, t_{m+1})$. The solution is to use *successive approximations* for the non-linear part of the equation in order to solve for $\pi(i, t_m)$. Thus, each major iteration of generating $\pi(i, t_m)$ from $\pi(i, t_{m+1})$ contains a set of *minor iterations* in which the value of $\pi(i, t_m)$ is successively refined. We denote the sth approximation (corresponding to the sth minor iteration) of $\pi(i, t_m)$ by $\pi^s(i, t_m)$. In the sth minor iteration, an improved approximation $\pi^{s+1}(i, t_m)$ is generated from $\pi^s(i, t_m)$. In order to derive $\pi(i, t_m)$ from $\pi(i, t_{m+1})$, we start off by setting the value of $\pi^1(i, t_m)$ to $\pi(i, t_{m+1})$ as the first approximation. This approximation is improved in successive iterations, with the use of the following relationship between $\pi^s(i, t_m)$, $\pi^{s+1}(i, t_m)$ and $\pi(i, t_{m+1})$ in the sth iteration:

$$\pi(i, t_{m+1}) = \pi^{s+1}(i, t_m) + (1 - \pi^{s+1}(i, t_m)) \cdot (1 - \prod_{j \in IN(i, t_m, t_{m+1})} (1 - \pi^s(j, t_m) \cdot p_{ji}(t_m, t_{m+1})))$$

It is important to remember that the only unknown variable in the sth iteration is $\pi^{s+1}(i, t_m)$, since $\pi(i, t_{m+1})$ is known and the value of $\pi^s(i, t_m)$ is known from the previous iteration. We also note that the non-linear part of the above equation, which is denoted by $(1 - \prod_{j \in IN(i, t_m, t_{m+1})}(1 - \pi^s(j, t_m) \cdot p_{ji}(t_m, t_{m+1})))$ is expressed in terms of the (known) approximation $\pi^s(\cdot, \cdot)$ from the sth iteration. This results in a simple linear equation in $\pi^{s+1}(i, t_m)$, which can be easily solved. For brevity, let us denote the non-linear part of the relationship between $\pi^s(\cdot, \cdot)$ and $\pi^{s+1}(\cdot, \cdot)$ by $\gamma^s(i, t_m)$.

(4.4)
$$\gamma^s(i, t_m) = (1 - \prod_{j \in IN(i, t_m, t_{m+1})} (1 - \pi^s(j, t_m) \cdot p_{ji}(t_m, t_{m+1})))$$

Therefore, we can express the relationship between $\pi^{s+1}(\cdot, \cdot)$ and $\pi^s(\cdot, \cdot)$ as follows:

(4.5)
$$\pi(i, t_{m+1}) = \pi^{s+1}(i, t_m) + (1 - \pi^{s+1}(i, t_m)) \cdot \gamma^s(i, t_m)$$

This provides as simple solution for $\pi^{s+1}(i, t_m)$ in the sth minor iteration from known variables derived in the $(s-1)$th minor iteration. In other words, we have:

$$\pi^{s+1}(i, t_m) = \frac{\pi(i, t_{m+1}) - \gamma^s(i, t_m)}{1 - \gamma^s(i, t_m)} \tag{4.6}$$

We repeat the above iterative solution continuously, until the solution converges. Specifically, we compute the value $\frac{\sum_{i \in N(t_m)} |\pi^{s+1}(i, t_m) - \pi^s(i, t_m)|}{\sum_{i \in N(t_m)} \pi^s(i, t_m)}$, which reduces successively as the probabilities are approximated better. We terminate when the value falls below 0.01. We note that since the above system of equations is approximate, it is possible for some of the estimated probabilities to be mildly negative. In such cases, we set these probability values to 0, and normalize the other probability values, so that they sum to 1. The overall algorithm *RetrospectiveSources* is illustrated in Figure 3.

4.2 Backward Influence Algorithm: Maximization Version While the afore-mentioned approach is designed for optimizing the distribution to a *known* ending pattern of probabilities, we do not know what this known ending pattern *should be* in order to optimize for the highest influence spread. In this subsection, we will propose such an approach. This serves the same function as the forward algorithm. except that the results are achieved much more efficiently from a computational perspective.

In order to *maximize* the spread of influence, we formulate each iteration as an *optimization* problem. The objective function of this optimization problem maximizes the *increase* in the probabilities from time period t_m to t_{m+1}. Our approach is to determine the set of probabilities $\pi(i, t_m)$ for each time period which maximize the increase in the expected number of infections from time period t_m to time period t_{m+1}. Thus, each such set of probabilities is a *local optimization problem* between interval t_m and t_{m+1} about the value of $\pi(i, t_m)$ which leads to the maximum increase. The local optimizations over the different intervals can be combined in order to determine the most influential nodes for information release. Specifically, let $\pi^*(i, t_m)$ be the optimum probability of node i at time period t_m which leads to the *maximum* increase between the time-periods t_m and t_{m+1}.

We note that each of these solutions is a *local optimization* which does **not** relate the probabilities between the different periods. Furthermore, we note that we cannot directly control the probabilities in the later periods, though they are correlated with the initial release probabilities. Therefore, in order to account for this, we use an adjustment of the probability values in successive intervals.

We subtract from $\pi^*(i, t_j)$ the *increase in* influence values which result as a transmission from t_{j-1}. This is because if the node i already receives a lot of influence increase because of infection from previous iterations, we do not need to add s large a component from t_j. Ideally, we should use the increase in influence from *all previous iterations*, but such adjustments become less useful for earlier and earlier iterations. Therefore, we just use the increase from the last iteration as an approximation. Let $\Delta^*(i, t_j)$ be in the increase in influence value from t_{j-1} to t_j with the use of $\pi^*(i, t_{j-1})$ as the probability at t_{j-1}. This is the value which is subtracted from $\pi^*(i, t_j)$. The value of $\Delta^*(i, t_j)$ can be computed as follows:

(4.7)
$$\Delta^*(i, t_j) = (1-\pi^*(i, t_{j-1}))\cdot \prod_{q\in IN(i,t_{j-1},t_j)} (1-\pi^*(q, t_{j-1}))$$

We note that the above relationship can be obtained by simple rearrangement of the infection equation introduced earlier. Then, the adjusted probabilities are given by $\pi^*(i, t_j) - \Delta^*(i, t_j)$. Therefore, the initial release probability $\pi(i, t_0)$ is defined as a composite of these release probabilities as follows:

$$\pi(i, t_0) = \sum_{j=0}^{r-1} \cdot(\pi^*(i, t_j) - \Delta^*(i, t_j)) \tag{4.8}$$

Next, we describe how to determine the locally optimized probabilities $\pi^*(i, t_m)$ at time period t_m which results in the maximum increase in the underlying probability. This increase for node i is equal to $\pi(i, t_{m+1}) - \pi(i, t_m)$. The overall increase for all nodes in $N(t_{m+1})$ is given by $\sum_{i\in N(t_{m+1})}(\pi(i, t_{m+1} - \pi(i, t_m)))$. Therefore, the objective function $F(\overline{\pi_m}, t_m)$ of this problem can be written as follows:

$$\begin{aligned} F(\overline{\pi_m}, t_m) &= \sum_{i\in N(t_{m+1})} (\pi(i, t_{m+1}) - \pi(i, t_m)) \\ &= \sum_{i\in N(t_{m+1})} (1 - \pi(i, t_m)) \cdot (1 - \prod_{j\in IN(i,t_m,t_{m+1})} (1 - \\ &\qquad -\pi(j, t_m) \cdot p_{ji}(t_m, t_{m+1}))) \end{aligned}$$

We note that the second condition is directly derived from the transmission equation discussed earlier. The objective function is simply the increase in the expected number of infections from time period t_m to t_{m+1}. We note that $F(\overline{\pi_m}, t_m)$ is now expressed purely in terms of the probabilities at period t_m, since the transmission relationships have been used in a backward way in order to eliminate the probabilities at time period t_{m+1}. Therefore, the expression for $F(\overline{\pi_m}, t_m)$ is a non-linear objective function expressed in terms of $\pi(i, t_m)$ which needs to be maximized. We further note that the value of each $\pi(i, t_m)$ lies in the range $(0, 1)$. If we set all the values of $\pi(i, t_m)$ to 0 (or to 1), the value of the objective function $F(\overline{\pi_m}, t_m)$ is 0, because such probabilities at time t_m lead to no improvement of the objective function value at time t_{m+1}. Clearly, the optimum value of $F(\overline{\pi_m}, t_m)$ is achieved for values of $\pi(i, t_m)$ which are such that at least some components are non-zero. Therefore, the key question is how we can determine the value of the optimum vector $\overline{\pi_m}$, which maximizes this increase. We achieve this by using a steepest gradient ascent method.

As before, we use an iterative approach in which the notation $\overline{\pi_m^s}$ denotes the value of the probability vector in the sth iteration. We start off by setting each $\pi^0(i, t_m)$ to $m/|N(t_m)|$. We determine the gradient of the objective function at this point and determine the step length (along this gradient) which leads to the maximum increase in the objective function value, while retaining the constraint that each $\pi(i, t_m)$ lies in the range $(0, 1)$. This step length is determined by binary search. This approach is used to estimate $\pi^{s+1}(i, t_m)$ from $\pi^s(i, t_m)$. In each iteration, we determine the (partial) gradient $\nabla F(\overline{\pi_m^s}, t_m)$ with respect to each element of vector $\overline{\pi_m^s}$ which is not already either 0 or 1. We define the partial gradient as follows:

Definition 2. (Partial Gradient) *The partial gradient $\nabla^p F(\overline{\pi_m^s}, t_m)$ of the objective function $F(\overline{\pi_m^s}, t_m)$ is defined as a vector containing one element for each node in the data. The ith element of this vector is defined as follows:*

- *If $0 < \pi^s(i, t_m) < 1$, then the value of the ith element is the partial derivative of $F(\overline{\pi_m^s}, t_m)$, with respect to $\pi(i, t_m)$ and evaluated at $\pi^s(i, t_m)$.*
- *if $\pi^s(i, t_m)$ is 0 or 1, then the value of the ith element is set to 0.*

The value of $\pi^{s+1}(i, t_m)$ is derived from $\pi^s(i, t_m)$ by a step along this vector of partial gradients which maximizes the improvement in the objective function value, subject to the constraint that the probability vectors must always lie in $[0, 1]$. Thus, for step length a, we have:

$$\overline{\pi_m^{s+1}} = \overline{\pi_m^s} + a \cdot \nabla^p F(\overline{\pi_m^s}, t_m) \tag{4.9}$$

The value of a is determined by binary search. In order to perform the binary search, we start at a small step length and keep doubling until either the $\pi \in (0, 1)$ constraints are violated, or the doubling did not lead to a greater improvement than the second-last iteration. We use the step length in the second-last iteration as the one to use. The process of using partial gradients instead of gradients ensures that once an element of the probability vector reaches either 0 or 1, it is no longer modified. This process continues until the quality of the objective function can no longer be improved.

5 Experimental Results

We tested both the maximization and retrospective versions of the influence analysis algorithm. We used two real data sets in order to validate the effectiveness of our approach. The following performance measures were used.

5.1 Performance Measures and Baselines We used different performance measures for the maximization and retrospective versions of the algorithm. For the forward and backward maximization versions of the algorithm, we used the total influence flow value as the effectiveness measure. The influence flow value for a particular starting set of k nodes is calculated by simulating the the total flow for that set of starting nodes and corresponding dynamic set of edges. This simulation is performed by a sequential use of the infection equation on the dynamic set of edges. The larger the total flow value, the better the quality of the result.

For the case of the retrospective version of the backward algorithm, we used the plot of precision versus recall in order to determine the effectiveness results. In order to generate the precision-recall tradeoff, we picked a ground truth influence set of fixed size, and simulated a final infection pattern at the different nodes. The infection pattern was simulated by using the probabilities of infection of edges over the entire period of infection in the network. The final infection pattern was used in conjunction with the retrospective algorithm in order to estimate the source infection points in comparison to the ground truth. The estimated set was compared to the ground truth in order to generate the precision and recall. The precision was defined as the percentage of the *found set* which was truly a part of the most influential set. The recall was defined as the percentage of the ground truth which was a part of the most influential set determined by the algorithm. Specifically, if S_G and S_F be the ground-truth sets and found sets respectively, then the precision and recall were defined as follows:

$$\text{Precision} = \frac{|S_G \cap S_F|}{|S_F|}$$

$$\text{Recall} = \frac{|S_G \cap S_F|}{|S_G|}$$

In order to test the retrospective algorithm, we work with a fixed ground-truth k_0 and vary the value of the algorithm input parameter k in order to determine the most influential nodes. Clearly, for larger values of k the recall (typically) increases, and precision reduces. The tradeoff curve between precision and recall was used as the effectiveness measure.

In addition, we also reported the running times of the different algorithms for different variations of parameters. Since the primary goal of this paper is to design the first algorithm for *dynamic influence* algorithm, we used a version of the forward algorithm in which the evolution threshold was set to 1 as a baseline. We note that this special case of our forward dynamic algorithm actually reduces to a static baseline algorithm which is equivalent to the *RankedReplace* algorithm [1]. The goal here was to determine the advantages of using a dynamic algorithm for this problem over a direct application of previously designed static algorithms.

5.2 Data Sets We used two data sets in order to test the approach.

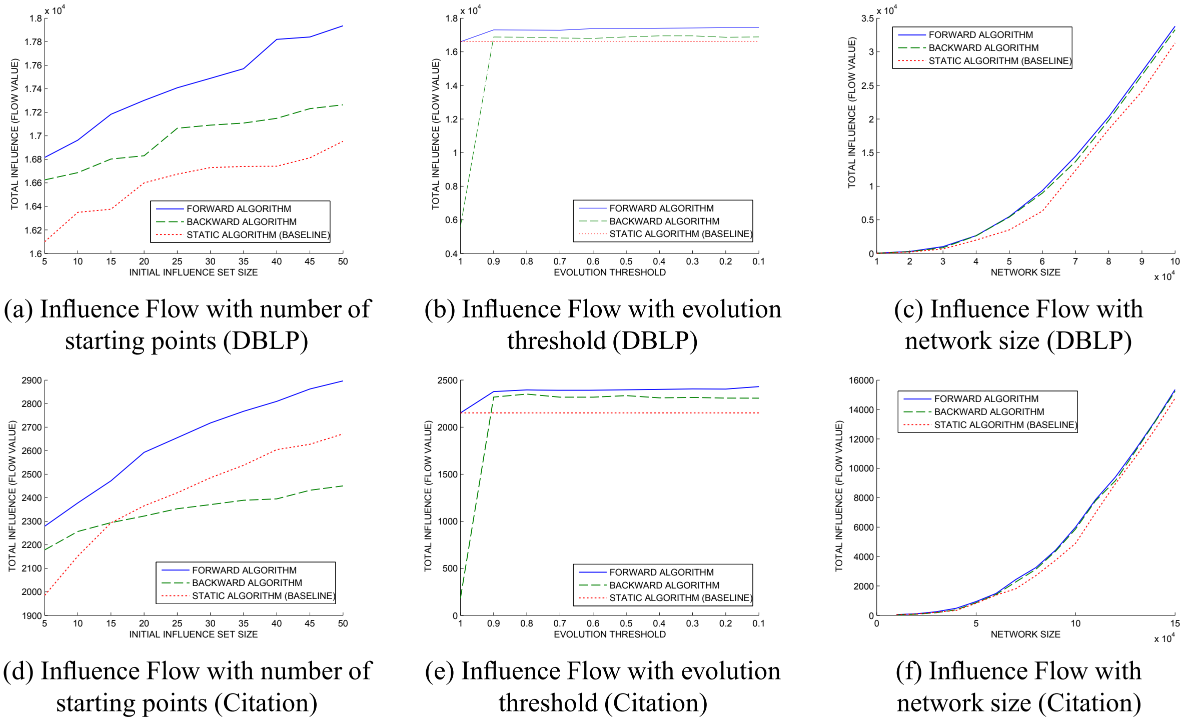
DBLP Data Set: This data set[2] corresponds to the DBLP data, in which the nodes correspond to authors and the edges correspond to the co-authorship interaction between authors. The temporal order of the edges is derived from the year of publication. The time that an edge exists is generated from an exponential distribution with a mean of one year. The probability of information spread along an edge is given by the Equation 2.2, in which λ_{ij} represents the infection parameter, and a was set to 0.8. For each edge, the parameter for the transmission rate λ_{ij} is generated by picking a random value from the uniform distribution in $[0, 1]$, and then dividing it by the time for which the edge is present in the network. The data set contained 934,672 nodes and 8,850,502 edges, and it spans from 1938 to 2011 in terms of time.

Arnetminer Citation Data Set: This data set[3][18] corresponds to a graph structure of citations from Arnetminer, in which nodes correspond to authors, and edges correspond to citation relation. For every citation, there exists an edge between the first author of the paper which is cited to the first author of the paper in which the citation occurs. The time of arrival of an edge is determined by the time at which the citation took place. The method for generating the time of edge presence and the transmission probabilities was the same as the DBLP data set. The data set contained 916,979 nodes and edges, and it spans in time from 1960 to 2011.

5.3 Effectiveness Results for Influence Maximization In this section, we will present the effectiveness results of the maximization versions by varying different parameters. We used a static baseline which was a special case of our forward algorithm, when no dynamic analysis was used, and the evolution threshold was set to 1. This corresponds to the *RankedReplace* algorithm discussed in [1]. In Figures 4(a) and (d), we have illustrated the variation in influence

[2] http://dblp.uni-trier.de/xml/

[3] http://arnetminer.org/citation



(a) Influence Flow with number of starting points (DBLP)

(b) Influence Flow with evolution threshold (DBLP)

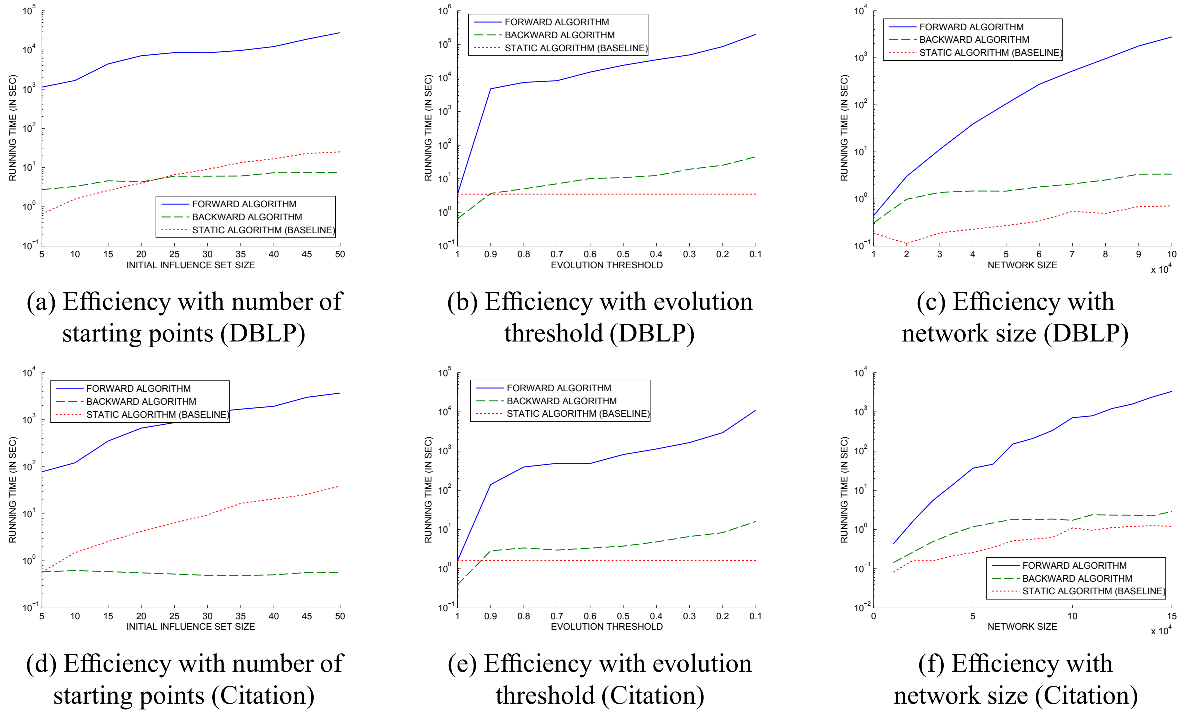
(c) Influence Flow with network size (DBLP)

(d) Influence Flow with number of starting points (Citation)

(e) Influence Flow with evolution threshold (Citation)

(f) Influence Flow with network size (Citation)

Figure 4: Effectiveness Results for DBLP and Citation Data set



(a) Efficiency with number of starting points (DBLP)

(b) Efficiency with evolution threshold (DBLP)

(c) Efficiency with network size (DBLP)

(d) Efficiency with number of starting points (Citation)

(e) Efficiency with evolution threshold (Citation)

(f) Efficiency with network size (Citation)

Figure 5: Efficiency Results for DBLP and Citation Data set

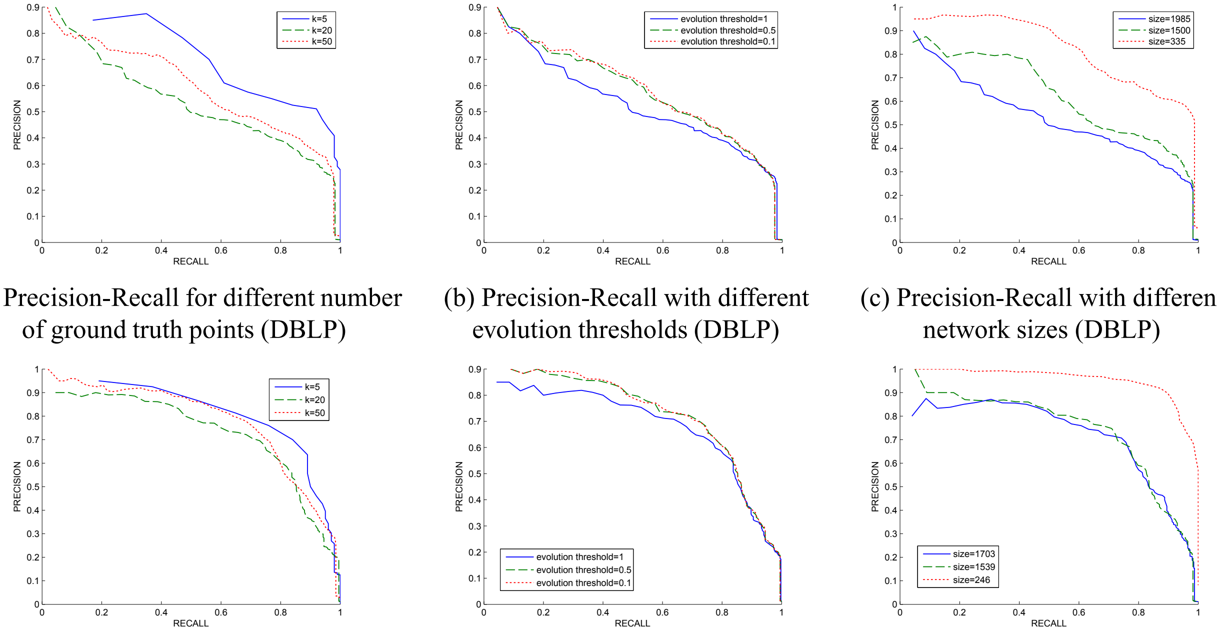
flow value with increasing number of seed starting points for the *DBLP* and *Citation* data sets respectively. The evolution threshold η was fixed to 0.9 in both these cases. The number of starting points are illustrated on the X-axis, and the total influence flow value is illustrated on the Y-axis. In each case, the total influence flow value increases with the number of starting points for both data sets. In each case, the forward algorithm performed the best. The backward algorithm was superior to the static baseline for the case of the *DBLP* data set throughout the entire range of the X-axis. As we will see later, the backward algorithm has a tremendous efficiency advantage over the forward algorithm. Therefore, the two algorithms have different advantages in terms of effectiveness and efficiency. For the case of the *Citation* data set, the backward algorithm did not perform as well, though it will still superior to the static algorithm for a portion of the X-axis range.

We also tested the influence flow value with different values of the evolution threshold η. A smaller value of the evolution threshold results in a better discretization of the network and therefore creates more accurate results. On the other hand, this accuracy comes at the expense of slower running times. The results for different values of the evolution threshold for the *DBLP* and *Citation* data sets are illustrated in Figures 4(b) and (e) respectively. The influence set size k was set to 20 for both algorithms. In each case, the *evolution threshold* is illustrated on the X-axis in *decreasing order* (or better granularity), and the influence flow value is illustrated on the Y-axis. The performance of the static algorithm is a horizontal line in this case, because the evolution threshold parameter is relevant only to the dynamic algorithms. It is evident that the effectiveness of both the forward and backward algorithms increase with lower evolution threshold, because of the accuracy advantages of a more fine grained temporal representation of the network. As in the previous case, the forward algorithm is superior to the backward algorithm in terms of the total influence spread. Both algorithms are superior to the static algorithm over a majority of the range of the evolution threshold, and the forward algorithm is always superior.

We also tested the effectiveness of the method with networks of different size. In order to generate networks of different sizes, we sampled nodes from the *DBLP* and *Citation* networks respectively. The results for the *DBLP* and *Citation* networks are illustrated in Figures 4(c) and (f) respectively. In the both cases, the influence set size k was set to 5, and the evolution threshold η was set to 0.9. The network size is illustrated on the X-axis, and the influence flow value is illustrated on the Y-axis. It is evident that the flow value increases with increasing network size because of the greater conductance of a larger network. As in the previous algorithms, the forward algorithm performed the best, followed by the backward algorithm, and then the static baseline algorithm. Thus, our methods provide a definite advantage over the use of a purely static influence maximization approach.

5.4 Efficiency Results for Influence Maximization All experiments were executed on 2.7GHz computer with Intel Pentium Dual Core Processor, 4GB memory, and running Windows 7 Home Premium. The program was implemented in Java. We tested the efficiency of the different methods with increasing influence set size. The results for the *DBLP* and *Citation* networks are illustrated in Figures 5(a) and (d) respectively. The evolution threshold η was set to 0.9. The number of starting points are illustrated on the X-axis, and the running time is illustrated on the Y-axis. It is evident that the backward algorithm is significantly more efficient as compared to the forward algorithm. It is important to note that the Y-axis is drawn on a logarithmic scale, and the backward algorithm is between 2 and 3 orders of magnitude faster than the forward algorithm. For example, for an influence set size of 50, the forward algorithm requires more than 27519 seconds, whereas the backward algorithm requires only about 7 seconds. Thus, while the backward algorithm does not provide quite as accurate results as the forward method, it provides an excellent and fast approximation with the use of the gradient-based method. The backward algorithm is also faster than the static baseline algorithm in most cases, and unlike the other algorithms, its efficiency is essentially insensitive to the influence set size. This is because the gradient-based method uses an interior point approach in which the efficiency of each iteration does not depend upon the influence set size, and the the number of iterations also does not seem to depend upon the influence set size.

We also tested the efficiency for different values of the evolution threshold. The efficiency results for the *DBLP* and *Citation* data sets are illustrated in Figures 5(b) and (e) respectively. The influence set size k was set to 20. The evolution threshold is illustrated (in decreasing order) on the X-axis, and the running time is illustrated on the Y-axis. As in the case of the effectiveness results, the static baseline is shown as a horizontal line, because it does not depend upon the evolution threshold. As in the previous chart, the backward algorithm was more than 2 orders of magnitude efficient than the forward algorithm over the entire range of the evolution threshold. Both the forward and backward algorithms require more time with reducing value of the evolution threshold (or higher temporal granularity). This is because the use of higher temporal granularity resulted in a larger number of iterations for both algorithms. While the static algorithm was more efficient than the backward algorithm for high temporal granularities, it is also important to note that the backward algorithm always provided more accurate results than the static algorithm in this range. Thus,



(a) Precision-Recall for different number of ground truth points (DBLP)

(b) Precision-Recall with different evolution thresholds (DBLP)

(c) Precision-Recall with different network sizes (DBLP)

(d) Precision-Recall for different number of ground truth points (Citation)

(e) Precision-Recall with different evolution thresholds (Citation)

(f) Precision-Recall with different network sizes (Citation)

Figure 6: Results for Retrospective Algorithm

the forward and backward algorithms provide different tradeoffs in terms of effectiveness and efficiency.

Finally, we tested the efficiency with increasing network size. The results for the *DBLP* and *Citation* data sets are illustrated in Figures 5(c) and (f) respectively. The influence set size k was set to 5, and the evolution threshold was set to 0.9. The network size is illustrated on the X-axis and the running time is illustrated on the Y-axis. It is clear that larger networks require more running times. This is natural to expect, because larger networks require more steps at each stage of the processing. The backward algorithm is significantly more efficient than the forward algorithm, and this is especially the case for larger networks. The static baseline is only marginally more efficient than the backward algorithm. Thus, the backward algorithm provides an excellent and efficient alternative, which retains its effectiveness, and can also be utilized efficiently for larger networks.

5.5 Precision-Recall Results for Retrospective Algorithm We also tested the retrospective algorithm for its precision-recall tradeoff. The retrospective algorithm determines the most likely source set for a given pattern of attack. In order to generate the precision-recall tradeoff, we picked a ground truth influence set of fixed size, and simulated a final infection pattern at the different nodes. The infection pattern was simulated by using the probabilities of infection of edges over the entire period of infection in the network. The final infection pattern was used in conjunction with the retrospective algorithm in order to estimate the source infection points in comparison to the ground truth. We varied the size of the influence set determined by the algorithm.

With increasing size of this determined influence set (for a fixed ground truth size), the precision reduces but the recall increases. We plotted this tradeoff by varying the influence set size reported by the algorithm. We tested the approach for three different ground truth sizes corresponding to $k = 5$, $k = 20$ and $k = 50$ respectively. The results for the *DBLP* and *Citation* data sets are illustrated in Figures 6(a) and (d) respectively. The recall is illustrated on the X-axis and the precision is illustrated on the Y-axis. The evolution threshold η was set to 0.1 in this case. It is evident from the figures that the the precision-recall tradeoff is superior for smaller ground truth sizes. This is because smaller ground truth sizes correspond to instances which are easier to solve in terms of tracing back to the source of the infection. The absolute precision-recall tradeoff was also quite strong, especially for the *Citation* data set, in which a precision of 0.70 was achieved at a recall point of 0.84.

We also tested the precision recall trade-off with different values of the evolution threshold. The ground truth size k was set to 20. The results for different evolution thresholds for the *DBLP* and *Citation* data sets are illustrated in Figures 6(b) and (e) respectively. The ground truth set size k was set to 20. It is evident that the precision-recall tradeoff was better for the use of lower evolution thresholds. This corresponds to a better granularity of representation. A higher granularity of temporal representation improves the effectiveness of the algorithm.

Finally, we tested the precision-recall tradeoff for different network sizes. The results for the *DBLP* and *Citation* data sets are illustrated in Figures 6(c) and (f) respectively. The ground truth influence set size k was set to 20, and the evolution threshold η was set to 0.1. The precision-recall tradeoff was usually better for smaller and sparser networks because it was easier to trace an infection pattern to a particular set of sources in such cases. For larger and denser networks, many different sets of starting points result in a similar infection pattern. Therefore, the problem is fundamentally more difficult in those cases. However, in all cases a very robust precision-recall tradeoff was achieved by the algorithm. This suggests the consistent effectiveness of our approach.

6 Conclusions and Summary

This paper presents methods for influential node discovery in dynamic networks. While the problem of influential node discovery has been widely studied in the case of static networks, the problem of influence analysis in a *dynamic case* with *transient and rapidly evolving interactions* is a much more difficult and challenging problem. The patterns of social interactions between a pair of entities may rapidly evolve and change over time. This paper designs a first set of methods for *temporally sensitive* algorithms in influence analysis. We present experimental results presenting the advantages of the approach over the static scenario.

Acknowledgements

Research of the first author was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

Research of the second and third authors was supported in part by NSF through grants IIS 0905215, DBI-0960443, CNS-1115234, IIS-0914934, OISE-1129076, and OIA-0963278, and Google Mobile 2014 Program.

References

[1] C. Aggarwal, A. Khan, and X. Yan, *On Flow Authority Discovery in Social Networks*, SDM Conf., (2011), pp. 522–533.

[2] C. Aggarwal, and H. Wang, Managing and Mining Graph Data, *Springer*, (2010).

[3] C. Aggarwal, Social Network Data Analytics, *Springer*, (2011).

[4] N. Berger, C. Borgs, J. T. Chayes, and A. Saberi, *On the spread of viruses in the internet*, SODA, (2005).

[5] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos, *Epidemic thresholds in real networks*, ACM Trans. on Inf. Systems and Security, 10(4), (2008).

[6] W. Chen, Y. Wang, and S. Yang, *Efficient Influence Maximization in Social Networks*, KDD Conf., (2009).

[7] W. Chen, C. Wang, and Y. Wang, *Scalable influence maximization for prevalent viral marketing in large-scale social networks*, KDD Conference, (2010).

[8] P. Domingos, and M. Richardson, *Mining the network value of customers*, ACM KDD Conference, (2001).

[9] D. Kempe, J. Kleinberg, and E. Tardos, *Maximizing the Spread of Influence in a Social Network*, ACM KDD Conf., (2003).

[10] J. Kleinberg, *Authoritative sources in a hyperlinked environment*, JACM 46(5), (1999) pp. 604-632.

[11] J. M. Kleinberg, *The flow of on-line information in global networks*, SIGMOD Conf., (2010).

[12] G. Kossinets, J. M. Kleinberg, and D. J. Watts, *The structure of information pathways in a social communication network*, ACM KDD Conf., (2008).

[13] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. S. Glance, *Cost-effective outbreak detection in networks*, ACM KDD Conf., (2007).

[14] J. Leskovec, M. McGlohon, C. Faloutsos, N. Glance, and M. Hurst, *Cascading Behavior in Large Blog Graphs*, SDM Conf., (2007).

[15] M. E. J. Newman, S. Forrest, and J. Balthrop, *Email networks and the spread of computer viruses*, Phys. Rev. E 66, 035101, (2002).

[16] M. E. J. Newman, *The spread of epidemic disease on networks*, Phys. Rev. E 66, 016128, (2002).

[17] X. Song, C.-Y. Lin, B. L. Tseng, and M.-T. Sun, *Modeling and predicting personal information dissemination behavior*, ACM KDD Conf., (2005).

[18] J. Tang, D. Zhang, and L. Yao, *Social Network Extraction of Academic Researchers*, ICDM, (2007).

[19] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, *Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint*, SRDS, (2003), pp. 25-34.

[20] C. Wang, J. C. Knight, and M. C. Elder, *On computer viral infection and the effect of immunization*, ACM Annual Comp. Security App. Conf., (2000).

[21] F. Wu, B. Huberman, L. Adamic, and J. Tyler, *Information Flow in Social Groups, Physica A:*, Statistical and Theoretical Physics, (337)1–2, 1, (2004) pp. 327-335.

[22] `http://www.facebook.com`

[23] `http://www.linkedin.com`