In Search of Influential Event Organizers in Online Social Networks

Kaiyu Feng[1,2] Gao Cong[2] Sourav S. Bhowmick[2] Shuai Ma[3]
[1]LILY, Interdisciplinary Graduate School. Nanyang Technological University, Singapore
[2]School of Computer Engineering, Nanyang Technological University, Singapore
[3]SKLSDE Lab, Beihang University, China
{kfeng002@e., gaocong@, assourav@}ntu.edu.sg, mashuai@buaa.edu.cn

ABSTRACT

Recently, with the emergence of event-based online social services (*e.g., Meetup*), there have been increasing online activities to create, distribute, and organize social events. In this paper, we take the first systematic step to discover *influential event organizers* from online social networks who are essential to the overall success of social events. Informally, such event organizers comprise a small group of people who not only have the relevant skills or expertise that are required for an event (*e.g.,* conference) but they are also able to influence largest number of people to actively contribute to it. We formulate it as the problem of mining *influential cover set* (ICS) where we wish to find k users in a social network G that together have the required skills or expertise (modeled as attributes of nodes in G) to organize an event such that they can influence the greatest number of individuals to participate in the event. The problem is, however, NP-hard. Hence, we propose three algorithms to find approximate solutions to the problem. The first two algorithms are greedy; they run faster, but have no guarantees. The third algorithm is 2-approximate and guarantees to find a feasible solution if any. Our empirical study over several real-world networks demonstrates the superiority of our proposed solutions.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Management

Keywords

Influence Maximization; Event-based social network; Event Organization

1. INTRODUCTION

Recently, there has been increasing popularity and growth of event-based online social services, such as *Plancast* (`www.plancast.com`) and *Meetup* (`www.meetup.com`), in providing platforms for people to create, distribute, and organize social events. One may leverage these platforms to organize a variety of social events such as informal get-together, fund raising for flood victims, and organizing technical conferences. This has led to the germination of *event-based social network* (EBSN) [16] comprising both online and offline social interactions. The availability of such large scale social data paves the way to take a *data-driven* approach to select *influential event organizers* for a social event. Intuitively, such event organizers comprise a small group of people who not only have the relevant skills or expertise that are required for an event but they are also able to influence largest number of people to actively contribute to it. For example, to organize a multidisciplinary conference on sustainability, we may want to choose a small group of program chairs (*e.g.,* three or four) whose expertise covers the topics of the conference and are able to influence the largest number of people in the community to contribute and participate in the conference. Obviously, selection of influential event organizers is critical to the success of an event. Note that influential event organizers can be discovered by leveraging not only online EBSN but also traditional online social networks. For instance, in order to find influential organizers of a conference one may leverage the DBLP network to identify them. In this paper, we take the first systematic step towards discovering such influential event organizers from large-scale online social networks[1].

Specifically, we wish to find k influential users in a social network that together have the required skills to organize an event such that they can influence the greatest number of individuals to join the event. Consequently, we can formulate it as the problem of mining *influential cover set* (ICS) in social networks. Formally, given a social network G, let each node v in G refers to a user and is associated with a set of attributes $\mathcal{A}(v)$, where an attribute may refer to an expertise topic of a user. The ICS problem is to identify a set $\{v_1, \ldots, v_k\}$ of k users in G such that (1) $\bigcup_{i=1}^{k} \mathcal{A}(v_i)$ covers a set of query attributes, denoted by Q, and (2) the set of k users can influence the largest number of other users in G.

At first glance, it may seem that the ICS problem can be addressed by adopting any technique designed to address the traditional influence maximization (IM) problem [11], which aims to find a set of initial users of size k (referred to as *seeds*) so that they eventually influence the largest number of individuals (referred to as *influence spread*) in a social network. Unfortunately, this is not the case as state-of-the-art IM techniques do not meet the attribute covering constraint of the ICS problem. That is, in contrast to the traditional IM problem, the challenge for solving the ICS problem is to ensure that the k selected nodes cover the query attributes while their influence spread is maximized. Furthermore, in our problem setting k is significantly smaller (typically less than 6) compared to the IM problem since, as mentioned above, the number of event

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2612173.

[1]In this paper, we refer to traditional online social network as well as online EBSN collectively using the term "social network".

organizers of a particular social event is typically small. Consequently, it is imperative to design efficient techniques that leverage this unique characteristics of the seed set size.

We show that the ICS problem is NP-hard and propose three algorithms to address it. The first two solutions are greedy algorithms, namely ScoreGreedy and PigeonGreedy, and the third one is an approximation algorithm called PICS. Given an influential cover set query comprising query attributes Q and size k, ScoreGreedy selects k seed nodes in a greedy manner by considering both the number of newly covered attributes and marginal influence increase incurred by a candidate node. The PigeonGreedy algorithm leverages the pigeonhole principle since to cover $|Q|$ attributes with k nodes, we know at least one node in the selected seed set should cover more than $\frac{|Q|}{k}$ attributes. Based on this observation, it chooses the one with the maximum marginal influence increase among the nodes that cover at least $\frac{|Q|}{k}$ attributes.

While the greedy algorithms are efficient, they may not be able to return a seed set that covers all attributes in the query even if such a set exists. This led us to the design of the PICS (**P**artition-based **I**nfluential **C**over **S**et) algorithm, which always returns a seed set that covers all query attributes if such a set exists, and, even better, it is an approximation algorithm with performance guarantee. Specifically, for each partition of the given attributes Q, we find a seed set of size k in which the attributes of each node cover a subset of the partition and they have the largest influence spread. Finally, it returns the seed set with the maximum influence spread. We prove that the algorithm can approximate the optimal solution within a factor of 2.

Despite the benefits of PICS to address the ICS problem, a key limitation is that the number of partitions can be very large when $|Q|$ is large (*e.g.,* 115,975 at $|Q|$= 10), making it computationally expensive. To improve the efficiency, we propose the notion of cover-group to group partitions. We develop an algorithm called PICS+ for pruning the partitions in each cover-group from consideration, which cannot generate the result seed set. We show that this algorithm is instance optimal in pruning unnecessary partitions. In summary, the contributions of this paper are as follows.

- To the best of our knowledge, this is the first work to formulate the influential cover set (ICS) mining problem to discover influential event organizers in online social networks.
- We develop two greedy algorithms, which are efficient but have no performance guarantees, to address the ICS problem. We also present an approximation algorithm that guarantees to return a feasible solution if any .
- By applying the proposed algorithms to real-world datasets, we demonstrate their efficiency and effectiveness in discovering influential cover set.

The remainder of this paper is organized as follows. The related work is reviewed in the next section. Section 3 presents the formal definition of the influential cover set problem. In Sections 4, we present the greedy algorithms. Section 5 presents the approximation algorithm PICS, its approximation ratio, and its optimized variant called PICS+. Section 6 presents the experimental study. Finally, the last section concludes this paper. The key notations used in this paper are given in Table 1.

Table 1: Table of notations.

Notation	Definition
Q	The set of attributes in a query
k	The size of seed set
A	A set of attributes
$\mathcal{A}(v)$	The set of attributes associated to node v
$V(A)$	The set of nodes such that for any $v \in V(A)$, v can cover A
V_Q	The set of nodes such that for any $v \in V_Q$, $\mathcal{A}(v) \cap Q \neq \emptyset$
P	A partition
R	A covergroup
L_i	A list of tuples constructed with respect to $R.r_i$
t	A tuple in a list L
$\mathcal{T}$	A combination
$\sigma_{\mathcal{G}}(S)$	Influence spread of seed set S in $\mathcal{G}$
$\sigma_{\mathcal{G}}(S_1, S_0)^+$	$\sigma_{\mathcal{G}}(S_0 \cup S_1) - \sigma_{\mathcal{G}}(S_0)$, the influence increase of S_1 w.r.t. S_0

2. RELATED WORK

Event Organization and Event-based Social Networks. Lappas *et al.* [13] consider the team formation problem in the presence of a social network of individuals. They aim to find a team with the set of required skills and minimum communication cost. The problem formulation is also adopted by some subsequent work on team formation, *e.g.,* [2, 17]. These techniques aim to form a team to implement a task, but not organize an event; and more importantly they do not consider the influence of users in team formulation. However, influential organizers are very important for the success of online event planning [18].

Recently, Liu *et al.* [16] introduce the notion of event-based social network (EBSN) which comprises online and offline social interactions. Specifically, they analyze various characteristics of EBSN such as network properties, community structures and information flow over this network. They also propose a technique for event recommendation. Our work is orthogonal to this effort as we focus on the online component of the EBSN and present an efficient and scalable technique to find influential event organizers.

Influence Maximization Techniques. There exists a lot of work on the problem of influence maximization in social networks. Kempe *et al.* [11] formulate the problem as a discrete optimization problem, which is adopted by subsequent studies. They prove the influence maximization problem is NP-hard, and propose a greedy algorithm to approximately solve it by repeatedly selecting the node incurring the largest marginal influence increase. Most of subsequent algorithms follow the framework of the greedy algorithm. However, the greedy algorithm framework does not consider the attribute coverage when greedily selecting nodes into the seed set. Hence, the solution returned by such a framework may not satisfy the attribute coverage requirement in our problem setting.

The problem of calculating influence spread of a given seed set itself is intractable (Chen *et al.* [5] prove it to be *#P-hard*). A number of approaches have been proposed to estimate the influence spread. Kempe *et al.* [11] propose to simulate influence spreading process starting from the given seed set for a large number of times, and then use the average value of simulation results to approximate it. However, the simulation based method is computationally expensive. To mitigate this problem, Leskovec *et al.* [14] propose a mechanism called CELF to reduce the number of times required to calculate influence spread. Chen *et al.* propose two fast heuristics algorithms, namely DegreeDiscount [5] and PMIA [4], to select nodes at each step of the greedy algorithm. DegreeDiscount estimates the influence spread for a node using its degree after discounting the degrees of neighbors of the selected nodes in previous steps. PMIA calculates influence spread by employing *local influence arborescences*, which are based on the most probable influence path between two nodes. Jung *et al.* [10] estimate the influence spread using an iterative approach, which is derived from the belief propagation approach. The concept of influence spread path is proposed to estimate the influence of a set of nodes [12, 15].

Other Work on Influence. The problem of building the underlying influence propagation graph from history data has been studied, such as [8,19]. However, this problem is orthogonal to the influence maximization problem, which assumes that the influence graph is known.

3. THE INFLUENTIAL COVER SET (ICS) PROBLEM

In this section, we begin by briefly introducing the *Independent Cascade* (IC) model [11] for influence propagation, which has been widely adopted in addressing the influence maximization problem (e.g., [4,5,12,14,15]). Specifically, we leverage this influence propagation model to define the influential cover set problem.

3.1 Independent Cascade (IC) Model

We refer to a social network as $\mathcal{G}(V, E, w)$, where V denotes a set of nodes and $E = \{(v_i, v_j)|v_i, v_j \in V\}$ is a set of edges. For each edge $(u, v) \in E$, $w(u, v)$ is the propagation probability from u to v, which ranges over $(0, 1]$. In the IC model, (a) each node is either *active* or *inactive*, and (b) a node is only allowed to switch from an inactive state to an active state, but not vice versa.

Given a seed set S and a social network $\mathcal{G}$, the IC model works in an inductive way: Let $S_0 = S$ and S_t denote the set of nodes activated at step t. At step $t+1$, each node $v \in S_t$ has a single chance to activate each currently inactive neighbors u with a probability $w(u, v)$. The propagation process terminates when $S_t = \emptyset$.

The *influence spread* of S on $\mathcal{G}$, denoted as $\sigma_{\mathcal{G}}(S)$, is defined as the total number of nodes influenced by S. We define $\sigma_{\mathcal{G}}(S_1, S_0)^+ = \sigma_{\mathcal{G}}(S_1 \cup S_0) - \sigma_{\mathcal{G}}(S_0)$ to represent the marginal influence increase produced by S_1 w.r.t. S_0. However, exact evaluation of influence spread under the IC model is #P-hard [4]. Monte-Carlo (MC) simulations method is proposed to obtain an accurate estimate of the influence spread [11]. By MC simulations method, we can obtain arbitrarily close approximation of influence with high probability.

In addition, heuristic approaches (*e.g.,* [4,5,12,14,15]) have been proposed to approximate the computation.

3.2 Problem Definition

In practice, each node in a social network is usually associated with a set of attributes to describe the node, which can represent the skills or interests of a user. For instance, in *Meetup*, users are associated with a set of interests like music and games. Given a social network $\mathcal{G}$ (or network for brevity), we denote the set of attributes of a node v by $\mathcal{A}(v)$. If attribute $a_i \in \mathcal{A}(v)$, we say that node v *has* (or covers) attribute a_i; otherwise v does not have (or cover) attribute a_i. We say a set of nodes $V' \subseteq V$ cover a set of attributes $A = \{a_1, \ldots, a_m\}$ if for any attribute $a_i \in A$, there exists at least one node $v \in V'$ such that v *has* a_i. We use $V(A)$ to denote the set of nodes such that each node in the set can cover A. Given a set of attributes Q, we define $V_Q \subseteq V$ the candidate set such that for any node $v \in V_Q$, v contains at least one attribute $a_i \in Q$. Hereafter, we refer to a network as $\mathcal{G}(V, E, w, \mathcal{A})$.

Example 1: Given a network containing 4 nodes v_1, v_2, v_3 and v_4. $\mathcal{A}(v_1) = \{a, b, c\}$, $\mathcal{A}(v_2) = \{a, b\}$, $\mathcal{A}(v_3) = \{c, d\}$, and $\mathcal{A}(v_4) = \{a, b\}$. Let $A_1 = \{a\}$. Then $V(A_1) = \{v_1, v_2, v_4\}$. □

It is natural to take the attributes into account in finding a set of most influential event organizers in a social network. As remarked earlier, the traditional influence maximization problem cannot be used to address this problem. Hence, in the following we propose the *influential cover set* problem to address it. Intuitively, the *influential cover set* is to find a set of k nodes in a social network such that the k nodes can cover the query attributes and the influence spread of the selected nodes is maximized.

DEFINITION 1 (INFLUENTIAL COVER SET (ICS)). *Given a set of attributes Q and parameter k, the influential cover set (*ICS*) problem aims to select the k seed nodes S from $\mathcal{G}(V, E, w, \mathcal{A})$:*

$$S = \arg \max_{S \subseteq V, |S|=k} \sigma_{\mathcal{G}}(S) \quad s.t.\ Q \subseteq \bigcup_{s \in S} \mathcal{A}(s), \tag{1}$$

where $\sigma_{\mathcal{G}}(S)$ is the influence spread of S on $\mathcal{G}$, i.e., *the number of nodes influenced by nodes in S, and $\mathcal{A}(s)$ denotes the set of attributes associated with node s.*

Note that $|Q|$ is typically small in our problem setting and hence we treat $|Q|$ as a constant. Furthermore, as mentioned earlier, we usually have $k < |Q|$ because we want to choose a small number k of users that have the required skills Q. Note that this is a challenging problem and our main focus in this paper; in the case of $k \geq |Q|$, the attribute covering constraint is easily satisfiable.

Observe that according to the problem definition, we look for the seed set with exactly k nodes and an empty set will be returned if such k nodes cannot be found, even if a smaller size seed set can already cover all the attributes in the query.

THEOREM 1. *The* ICS *problem is NP-hard.*

PROOF. The traditional influence maximization problem is NP-hard [11]. It can be regarded as a special case of ICS problem that the attribute set of each node can always cover the query Q. Hence the ICS problem is NP-hard. □

Remark Our solutions to the ICS problem are orthogonal to the influence probability of each edge in $\mathcal{G}$, whose interpretation is an application-specific issue [11] and is beyond the scope of this paper. We assume that we have influence probability of each edge in $\mathcal{G}$. In practice, there are different ways to set the probability [3, 7, 10, 11]. For example, we can adopt the approach [3] to learn a set of topic-aware influence probabilities for each edge, from which we determine the probability of each edge for an attribute set Q based on the relevance between query Q and the topics. Alternatively, we can follow other previous work on influence maximization (*e.g.,* [10, 11]).

4. GREEDY SOLUTIONS

In this section, we present two greedy algorithms for mining influential cover set (ICS).

4.1 Score-based Greedy Algorithm

To select seeds in a greedy manner, we need to take both attribute coverage and influence spread into consideration. Specifically, the node that we greedily select in each iteration is supposed to have a high marginal influence increase and cover many uncovered attributes.

To achieve this, we introduce a *ranking function* to measure the score of each node, which will be used to select nodes in the greedy algorithm. Given a set of uncovered attributes Q' and a set of selected seed nodes S', the final score of node v with respect to S' and Q' is a linear interpolation of two factors, namely the score for attribute coverage and the score for influence spread

$$Score(v, S', Q') = \alpha \frac{|\mathcal{F}(v, Q')|}{\max\limits_{u \in V_Q} |\mathcal{F}(u, Q')|} + (1-\alpha) \frac{\sigma_{\mathcal{G}}(\{v\}, S')^+}{\max\limits_{u \in V_Q} \sigma_{\mathcal{G}}(\{u\}, S')^+} \tag{2}$$

where $\alpha \in [0, 1]$ is a parameter used to balance them, $\mathcal{F}(v, Q') = \mathcal{A}(v) \cap Q'$ is the set of attributes that are newly covered by v, and $\sigma_{\mathcal{G}}(\{v\}, S')^+$ is the marginal influence increase of node v with respect to S'. Note that if $\max\limits_{u \in V_Q} |\mathcal{F}(u, Q')|$ (resp. $\max\limits_{u \in V_Q} \sigma_{\mathcal{G}}(\{u\}, S')^+$) is 0, then $\frac{|\mathcal{F}(v,Q')|}{\max\limits_{u \in V_Q} |\mathcal{F}(u,Q')|}$ (resp. $\frac{\sigma_{\mathcal{G}}(\{v\},S')^+}{\max\limits_{u \in V_Q} \sigma_{\mathcal{G}}(\{u\},S')^+}$) is set as 1.

Algorithm 1: PigeonGreedy

Input: $Q, k, \mathcal{G}$
Output: Seed Set S
$S \leftarrow \emptyset, Q' \leftarrow Q$;
while $|S| < k$ **do**
  $L \leftarrow \{u \mid |\mathcal{A}(u) \cap Q'| \geq \lceil \frac{|Q'|}{k-|S|} \rceil, u \in V_Q\} - S$;
  if $L = \emptyset$ **then return** $\emptyset$;
  $v \leftarrow \arg\max_{u \in L} \sigma_{\mathcal{G}}(\{u\}, S)^+$;
  $Q' \leftarrow Q' - \mathcal{A}(v)$;$S \leftarrow S \cup \{v\}$
return S;

Based on the above score, we develop a greedy algorithm to select seed nodes iteratively, called ScoreGreedy. It selects seeds in k iterations. In each iteration, the algorithm selects the node with the maximum score and includes it in the seed set. If all attributes are covered by the k selected nodes, ScoreGreedy returns seed set S as the result; otherwise it returns $\emptyset$.

Note that ScoreGreedy cannot guarantee to find the seed set that can cover all attributes even if such a set exists.

Time complexity. Let $t(n)$ denote the time complexity of evaluating the influence spread of a node set in G, where n is the number of nodes in G. The time complexity of ScoreGreedy is $O(k \cdot |V_Q| \cdot t(n))$.

4.2 PigeonGreedy Algorithm

We propose another greedy algorithm that leverages the pigeonhole principle to select seed nodes in a greedy manner.

LEMMA 1. *If a seed set S with k nodes can cover the attribute set Q, then at least one node in S can cover no fewer than $\lceil \frac{|Q|}{k} \rceil$ attributes.*

Based on Lemma 1, we propose the PigeonGreedy algorithm outlined in Algorithm 1. Algorithm 1 follows the pigeonhole principle. The main idea is to iteratively apply Lemma 1 in a greedy manner. PigeonGreedy computes a set of nodes L which can cover more than $\lceil \frac{|Q'|}{k-|S|} \rceil$ uncovered attributes, where Q' is the set of uncovered attributes. Then it selects the node with maximum marginal influence increase from L. The algorithm invokes this procedure iteratively until k nodes are added to the seed set.

Time complexity. Let n_i be the number of nodes in L when selecting the i-th seed node. Let $t(n)$ denote the complexity of evaluating influence spread for one set in G, where n is the number of nodes in G. Therefore, the time cost of PigeonGreedy is $O(t(n)(\sum_{i \in [1,k]} n_i))$.

THEOREM 2. *PigeonGreedy can always return a seed set consisting of k nodes that covers attributes in the query if such a set exists when $k \geq |Q|$.*

PROOF. When $k \geq |Q|$, $\lceil \frac{|Q'|}{k-|S|} \rceil = 1$. So we can cover at least one attribute in each iteration. Hence PigeonGreedy can always return a seed set satisfying the constraint if such a set exists. □

Remark. Though PigeonGreedy can guarantee to return a seed set covering all attributes if such a set exists when $k \geq |Q|$, it does not hold when $k < |Q|$. We illustrate this with an example.

Example 2: Consider the graph in Figure 1. Let $Q = \{a, b, c, d, e, f\}$. Let $\mathcal{A}(v_1) = \{a, f\}$, $\mathcal{A}(v_2) = \{a, b, c, d\}$, $\mathcal{A}(v_3) = \{a, b, c, d, e\}$, and $\mathcal{A}(v_4) = \{b, c, d\}$. Suppose a user wants to select 2 nodes to cover all the attributes in Q. When PigeonGreedy is employed,

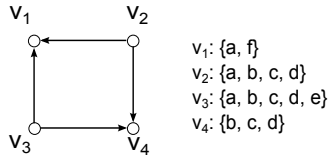


Figure 1: An example of the ICS problem.

we first select a node with maximum influence from the nodes that can cover no less than $\frac{6}{2} = 3$ attributes. Assume that v_2 produces larger influence spread than v_3. So we add v_2 to the seed set. Now $\{e, f\}$ are still uncovered and we can only choose one more node. But there exists no node that can cover both $\{e, f\}$. Hence the algorithm returns an empty set. However, there exists a seed set of 2 nodes, $\{v_1, v_3\}$, which cover all attributes in Q. □

The two algorithms proposed here exploit different greedy strategies. As we shall see later, ScoreGreedy usually has a better performance when k is small while PigeonGreedy can usually find a seed set with larger influence when k is large. However, neither of them can guarantee to return a seed set if such a set exists.

5. APPROXIMATION SOLUTIONS

In this section, we present the *Partition-based Influential Cover Set* (PICS) algorithm which addresses the limitation of the aforementioned greedy approaches by guaranteeing finding a seed set that covers all attributes in a query if such seed set exists. Furthermore, to reduce unnecessary computation in PICS, we propose an optimized variant of PICS which we refer to as PICS+.

5.1 Terminology

We first introduce some terminologies to facilitate exposition.

DEFINITION 2 (PARTITION). *Given a set Q of attributes and a positive integer k, $P = \{A_1, \ldots, A_m\}$ $(0 \leq m \leq k)$ is a* partition *of Q if and only if*

(1) for each $i \in [1, m]$, attribute set $A_i \subseteq Q$ is not empty;

(2) for any $i \neq j$ $(i, j \in [1, m])$, $A_i \cap A_j = \emptyset$.

(3) $\bigcup_{i \in [1,m]} A_i = Q$, i.e., $A_1, \ldots, A_m$ together cover Q.

We call $\bigcup_{A_i \in P} A_i$ the **attributes** *of P. Given two sets of attributes Q_1 and Q_2, $Q_1 \subseteq Q_2$, and a* partition *P' of Q_1, we say that P' is a* **partial** partition *of Q_2.*

Example 3: Consider an attribute set $Q = \{a, b, c, d, e\}$ and an integer $k = 3$. Then we have the following: (1) $P_1 = \{\{a, b, c\}, \{d, e\}\}$ is a partition; (2) $P_2 = \{\{b, c, d\}, \{a, e\}\}$ is a partition; (3) $P_3 = \{\{a, b, c\}, \{c, d\}, \{e\}\}$ is not a partition since $\{a, b, c\} \cap \{c, d\} \neq \emptyset$; (4) $P_4 = \{\{a, b\}, \{d, e\}\}$ is not a partition since c is not covered by P_4; And (5) $P_5 = \{\{a, b\}, \{c\}, \{d\}, \{e\}\}$ is not a partition since P_5 contains more than 3 attribute sets. □

LEMMA 2. *For a set Q of attributes and a positive integer k, the number of all possible* partitions *of Q is bounded by $\sum_{i=1}^{|Q|} \{{|Q| \atop i}\}$, where $\{{|Q| \atop i}\}$ is a stirling number of the second type,* i.e., $\{{|Q| \atop i}\} = \frac{1}{i!} \sum_{j=0}^{i} (-1)^{i-j} \binom{i}{j} j^{|Q|}$.

PROOF. There are two cases to consider: $k \leq |Q|$ or $k > |Q|$.

(1) When $k \leq |Q|$, it is easy to know that there are $\sum_{i=1}^{k} \{{|Q| \atop i}\} \leq \sum_{i=1}^{|Q|} \{{|Q| \atop i}\}$ partitions. Recall that the stirling number of the second type $\{{|Q| \atop i}\}$ is the number of ways to partition a set of $|Q|$ elements into i nonempty subsets [9].

Algorithm 2: PICS

Input: $Q, k, \mathcal{G}$
Output: Seed Set S

1 $S \leftarrow \emptyset$;
2 **if** $|V_Q| < k$ **then return** $\emptyset$
3 **for** $(j \leftarrow 0; j < k; j \leftarrow j + 1)$ **do**
4 $\quad SP[j] \leftarrow \arg\max_{u \in V_Q - S'} \sigma_{\mathcal{G}}(u, S')^+$;
5 $\quad S' \leftarrow S' \cup \{SP[j]\}$;
6 **for** *each* partition $P = \{A_1, \ldots, A_m\}$ *of* Q **do**
7 $\quad S_{\overline{P}} \leftarrow \emptyset$; $j \leftarrow 0$;
8 $\quad$ **while** $|S_{\overline{P}}| < k - |P|$ **do**
9 $\quad\quad$ For each $A_i \in P$, remove attributes covered by $S_{\overline{P}}$, and remove A_i from P if all attributes in A_i are covered;
10 $\quad\quad S_{\overline{P}} \leftarrow S_{\overline{P}} \cup \{SP[j]\}$; $j \leftarrow j + 1$;
11 $\quad$ **if** $\sigma_{\mathcal{G}}(S_{\overline{P}}) + upper(P) > \sigma_{\mathcal{G}}(S)$ **then**
12 $\quad\quad S_P \leftarrow$ PartitionSelect$(S_{\overline{P}}, \mathcal{G}, \{V(A_{i_1}), ..., V(A_{i_h})\})$;
13 $\quad\quad$ **if** $\sigma_{\mathcal{G}}(S_P \cup S_{\overline{P}}) > \sigma_{\mathcal{G}}(S)$ **then**
14 $\quad\quad\quad S \leftarrow S_P \cup S_{\overline{P}}$;
15 **return** S;

(2) When $k > |Q|$, there are at most $|Q|$ attribute sets in a partition by its definition. Then similar to (1), we know that there are $\sum_{i=1}^{|Q|} \{{}_{i}^{|Q|}\}$ possible partitions.

By (1) and (2) above, we have the conclusion. □

5.2 The PICS Algorithm

We now give the main intuition behind PICS. Recall an attribute set A_i in a partition P corresponds to a set of nodes $V(A_i)$ that each contains A_i. Clearly, if for every attribute set A_i of partition P, we select a node from $V(A_i)$ to form a seed set, it is guaranteed that such a seed set covers all attributes in Q. Since the number ($|P|$) of attribute sets of P may be less than k, to get a seed set with k nodes, we may need to further select nodes from V_Q. Hence, in PICS, we first select nodes from V_Q since $V(A_i) \subseteq V_Q$, and thus the nodes selected from V_Q are more likely to provide a better influence spread. Then the attributes covered by the nodes selected from V_Q can be removed from P because they have already been covered; we select nodes from $V(A_i)$ for each remaining A_i in P. Among the seed sets selected from every partition, we choose a seed set with the maximum influence spread.

More specifically, for each partition P, PICS aims to find a seed set with the maximum influence spread that covers all attributes in Q as follows: (1) We greedily select a set of $k - |P|$ nodes, denoted by $S_{\overline{P}}$, from V_Q (without any constraint of attribute coverage); (2) For each attribute set $A_i \in P$, we remove from it the attributes that have been covered by a node in $S_{\overline{P}}$; If all attributes in A_i have been covered, we remove A_i from P, and greedily choose a node from V_Q and add it to $S_{\overline{P}}$; We repeat this step until $|S_{\overline{P}}| + |P| = k$ and no attribute in P is covered by $S_{\overline{P}}$; and (3) The set of remaining attribute set actually forms a partial partition of Q, denoted by P'. For each attribute set $A_i \in P'$, we greedily select a node v_i from $V(A_i)$ to cover A_i; these selected nodes form a set, denoted by S_P. We call the set $S_{\overline{P}}$ *free set*, and the set S_P *constrained set*. (4) Finally, PICS returns the seed set with the largest influence among the seed sets of all partitions.

Algorithm 2 outlines the PICS algorithm. It takes as input a set of attributes Q, an integer k, and a social network $\mathcal{G}$. The result seed set is stored by S, which is initialized by an empty set (line 1). In the rare case that $|V_Q| < k$, it is impossible to find a seed set consisting of k nodes to cover all attributes, and the algorithm returns an empty set as the result (line 2). We greedily select k nodes from V_Q and store them in an array SP (lines 3–5), and then for each partition P we use array SP to populate free set $S_{\overline{P}}$, which records the set of nodes selected from V_Q (lines 7–10). For each $A_i \in P$, we remove from it attributes covered by $S_{\overline{P}}$ and remove it from P if all attributes in A_i are covered (line 9), *i.e.,* we can select a node from V_Q rather than $V(A_i)$. Next, PICS selects a node for each remaining attribute set in partition P to ensure that all attributes of Q are covered; all the selected nodes form constrained set S_p. Let $A_{i_1}, \ldots, A_{i_h}$ be the remaining attribute sets in P. Before finding S_p, we first apply an optimization (discussed later) to check if partition P can be pruned (line 11). If P cannot be pruned, PICS invokes PartitionSelect to find constrained set S_p with the node sets that can cover the remaining attribute sets of partition P as the arguments (line 12). Result S is updated if $S_p \cup S_{\overline{P}}$ has a higher influence (lines 13–14). When all partitions are considered, S is returned as the final result.

Function PartitionSelect$(S_{\overline{P}}, \mathcal{G}, \{V_1, \ldots, V_m\})$

Output: Seed Set S_P

1 $S_P \leftarrow \emptyset$;
2 **for** *each* $V_i \in \{V_1, \ldots, V_m\}$ **do**
3 $\quad$ **if** $V_i - S_P = \emptyset$ **then return** $\emptyset$;
4 $\quad v \leftarrow \arg\max_{u \in (V_i - S_P)} \sigma_{\mathcal{G}}(\{u\}, S_P \cup S_{\overline{P}})^+$;
5 $\quad S_P \leftarrow S_P \cup \{v\}$
6 **return** S_P;

The PartitionSelect procedure takes as input a set of attributes Q, an integer k, a network $\mathcal{G}$, and a group of node sets $\{V_1, \ldots, V_m\}$. It first checks whether V_i contains at least one node that has not been added to S_P yet. If not, empty set is returned as the result. Otherwise, it picks the node with maximum marginal influence increase from $V_i - S_P$ (line 4). It repeats this until every node set V_i provides one node to S_P.

Example 4: Reconsider Example 2. Q has many partitions and $P_1 = \{\{a, b, c, d, e, f\}\}$ is a partition. When processing P_1 in PICS, we add $SP[0]$ to $S_{\overline{P_1}}$ due to $k - |P_1| = 1 > 0$. We assume $SP[0] = v_1$. We then remove attributes of v_1 from each attribute set in P_1 and P_1 becomes $\{\{b, c, d, e\}\}$. Now $k = |P_1| + |S_{\overline{P_1}}|$ and free set $S_{\overline{P_1}} = \{v_1\}$. Assume that $\sigma_{\mathcal{G}}(S_{\overline{P_1}}) + upper(P_1) > \sigma_{\mathcal{G}}(S)$ and we invoke PartitionSelect to compute constrained set S_{P_1}. Now P_1 only contains one attribute set $\{b, c, d, e\}$ and $\{v_3\}$ is the node set that can cover it. PartitionSelelct returns $\{v_3\}$ as S_{P_1}. Therefore the seed set for P_1 is $\{v_1, v_3\}$. □

Next, we compute an upper bound of influence spread for any constrained set S_P selected for set P, which can be a partition or a partial partition (if any A_i is removed from P at line 12).

DEFINITION 3 (UPPER BOUND OF INFLUENCE SPREAD). *The upper bound of* P w.r.t. $S_{\overline{P}}$ *is defined by*

$$upper(P, S_{\overline{P}}) = \sum_{A_i \in P} \max_{v \in V(A_i)} \sigma_{\mathcal{G}}(\{v\}, S_{\overline{P}})^+$$

LEMMA 3. *We have* $upper(P, S_{\overline{P}}) \geq \sigma_{\mathcal{G}}(S_P, S_{\overline{P}})^+$ *for any* S_P *returned by function* PartitionSelect.

PROOF. Since the function $\sigma_{\mathcal{G}}()$ is submodular, we have

$$\sigma_{\mathcal{G}}(S_P, S_{\overline{P}})^+ \leq \sum_{v \in S_P} \sigma_{\mathcal{G}}(\{v\}, S_{\overline{P}})^+ \leq \sum_{A_i \in P} \max_{v \in V(A_i)} \sigma_{\mathcal{G}}(\{v\}, S_{\overline{P}})^+$$

□

Based on Lemma 3, if the upper bound together with the influence of $S_{\overline{P}}$ is smaller than the known best result $\sigma_{\mathcal{G}}(S)$, we simply prune the partition P to avoid finding a seed set for it.

THEOREM 3. *Given a network $\mathcal{G}(V, E, w, \mathcal{A})$, a set of attributes Q, and an integer k, PICS can always find a set of k nodes S that cover all attributes in Q, if such a set exists.*

PROOF. Assume that S is a seed set that covers all attributes in Q, which implies $|V_Q| \geq k$. We can find m ($\leq k$) nodes $\{v_1, \ldots, v_m\}$ from S such that $\cup_{i\in[1,m]}\mathcal{A}(v_i) = Q$ and for each node v_i there exists an attribute $a_i \in Q$ that is covered by v_i but not other $m-1$ nodes. The m nodes correspond to a partition $P = \{A_1, \ldots, A_m\}$ such that $v_i \in V(A_i)$, *i.e.,* $V(A_i) \neq \emptyset$. Since PICS considers all possible partitions, the partition P is guaranteed to be identified. For the partition P, it is guaranteed that PICS will generate set $S_{\overline{P}}$ due to $|V_Q| \geq k$. Since $V(A_i)$ is not empty for each A_i in P, PICS can select one node from $V(A_i)$. Hence, PICS can find S covering all attributes if such S exists. □

Time Complexity. Let $t(n)$ denote the time complexity of evaluating the influence spread of a node set in G, where n is the number of nodes in G. Given a partition $P = \{A_1, \ldots, A_m\}$, it takes $O((k)|V_Q| \cdot t(n))$ to select a seed set for a partition. Since there are at most $\sum_{i=1}^{|Q|}\{{}_i^{|Q|}\}$ partitions, the time complexity of PICS is $O((k)|V_Q| \cdot t(n) \cdot \sum_{i=1}^{|Q|}\{{}_i^{|Q|}\})$.

5.3 Approximation Ratio Analysis of PICS

When MC simulations are adopted to compute the influence spread, algorithm PICS has the following performance guarantee.

THEOREM 4. *If MC simulations are adopted to compute the influence spread, algorithm PICS provides an approximation bound of $1/2-\phi$, where ϕ is an arbitrarily small positive rational number.*

PROOF. By Lemma 2, we know that PICS finishes in polynomial time when $|Q|$ is bounded by a constant. We next show PICS has an approximation bound of $1/2 - \phi$.

First let $S^{OPT} = \{s_1, \ldots, s_k\}$ be an optimal solution that the set of k seed nodes has the maximum $\sigma_\mathcal{G}(S^{OPT})$. Also let partition $P_{OPT} = \{A_1, \ldots, A_m\}$ ($m \leq k$) such that for each $A_i \in P_{OPT}$, there must exist an $s_i \in S^{OPT}$ such that $s_i \in V(A_i)$, and, moreover, $s_i \neq s_j$ for A_i and A_j ($i \neq j \in [1, m]$). Note that algorithm PICS can always find partition P_{OPT}. Let $S = S_{P_{OPT}} \cup S_{\overline{P_{OPT}}}$ be the seed set selected from P_{OPT} by PICS, where $S_{P_{OPT}}$ is the constrained set and $S_{\overline{P_{OPT}}}$ is the free set. Without loss of generality, we assume that $S_{P_{OPT}}$ has g nodes, where $g \leq m$. We denote $P' = \{A'_1, ..., A'_g\}$ as the set of remaining attributes after removing the attributes covered by the free set. Apparently, for each $A'_i \in P'$, we can find a different $A_j \in P_{OPT}$ such that $A'_i \subseteq A_j$ and $V(A_j) \subseteq V(A'_i)$. Thus for each $A'_i \in P'$, we can find a different $s_i \in S^{OPT}$ such that $s_i \in V(A'_i)$. We use $S^{OPT}_{P_{OPT}}$ to denote such nodes and S^{OPT} is divided into $S^{OPT}_{P_{OPT}}$ and $S^{OPT}_{\overline{P_{OPT}}}$ such that $S^{OPT}_{P_{OPT}} = \{s_1, ..., s_g\}$ and $S_{P_{OPT}} = \{u_1, ..., u_g\}$, where $s_i \in V(A'_i)$ and $u_i \in V(A'_i)$ for any $i \in [1, g]$. We also assume that $S^{OPT}_{\overline{P_{OPT}}} = \{s_{g+1}, ..., s_k\}$ and $S_{\overline{P_{OPT}}} = \{u_{g+1}, ..., u_k\}$.

Since the influence function $\sigma_\mathcal{G}()$ is submodular and monotone, and $\forall S \subseteq V, t \in V, \sigma_\mathcal{G}(t, S)^+ \geq 0$, we have:

$$\begin{aligned}\sigma_\mathcal{G}(S^{OPT}) &\leq \sigma_\mathcal{G}(S) + \sum_{s\in S^{OPT}-S} \sigma_\mathcal{G}(\{s\}, S)^+ \\ &\leq \sigma_\mathcal{G}(S) + \sum_{s_i\in S^{OPT}} \sigma_\mathcal{G}(\{s_i\}, S)^+\end{aligned} \tag{3}$$

Let S_i be the set of nodes after the i-th seed node is added to the seed node set. For any node $v \in V, \sigma_\mathcal{G}(\{v\}, S_i)^+ \geq \sigma_\mathcal{G}(\{v\}, S)^+$. It follows that $\sigma_\mathcal{G}(S^{OPT}) \leq \sigma_\mathcal{G}(S) + \sum_{s_i\in S^{OPT}} \sigma_\mathcal{G}(\{s_i\}, S_{i-1})^+$.

When selecting the i-th seed node u_j, there are two cases to consider. (1) When the i-th node is from the free set, PICS selects the node with maximum approximate marginal influence increase from V_Q. Then the node that PICS selects has a larger influence than any node in $S^{OPT}_{\overline{P_{OPT}}}$. (2) When the i-th selected node is from the constrained set, it is the node with maximum marginal influence increase in $V(A'_j)$. Since $s_j \in V(A'_j)$, we know that the u_j can produce a higher marginal influence increase than s_j. Thus, for each node u_i in S, there's a different node s_i in S^{OPT} such that u_i has a larger marginal influence increase than s_i. By MC simulations, we can obtain arbitrarily close approximations to the exact influence with a high probability. Hence, for any $\varepsilon > 0$, there is a $\gamma > 0$ such that by using $(1+\gamma)$-approximate value of $\sigma_\mathcal{G}()$, we have

$$\sigma_\mathcal{G}(\{u_j\}, S_{i-1})^+ + \varepsilon \geq \sigma_\mathcal{G}(\{s_j\}, S_{i-1})^+ \tag{4}$$

By Equation 4, we have the following.

$$\begin{aligned}\sigma_\mathcal{G}(S^{OPT}) &\leq \sigma_\mathcal{G}(S) + \sum_{s_i\in S^{OPT}} (\sigma_\mathcal{G}(\{s_i\}, S_{i-1})^+ + \varepsilon) \\ &\leq \sigma_\mathcal{G}(S) + \sum_{u_i\in S} \sigma_\mathcal{G}(\{u_i\}, S_{i-1})^+ + k\varepsilon \\ &= \sigma_\mathcal{G}(S) + \sigma_\mathcal{G}(S) + k\varepsilon = 2\cdot\sigma_\mathcal{G}(S) + k\varepsilon\end{aligned} \tag{5}$$

That is, $\sigma_\mathcal{G}(S) \geq \frac{1}{2}\cdot\sigma_\mathcal{G}(S^{OPT}) - \frac{k}{2}\varepsilon$ for both cases.

Let S_{max} be the seed set returned by Algorithm PICS. Since PICS returns the seed set with maximum approximate influence spread, $\sigma_\mathcal{G}(S_{max}) \geq \sigma_\mathcal{G}(S) - \varepsilon$. Let $\phi = (\frac{k}{2}+1)\varepsilon$ and ϕ can be an arbitrarily small positive rational value by changing ε. Then we have $\sigma_\mathcal{G}(S) \geq \frac{1}{2}\cdot\sigma_\mathcal{G}(S^{OPT}) - \phi$.

Put these together, we have the conclusion. □

Next we discuss the approximation ratio when using heuristic methods to approximate the influence spread for PICS, such as DegreeDiscount [5], MIP [4], ISP [15], and IRIE [10]. Though these heuristic methods cannot provide a theoretical bound for influence spread, they perform well in practice in terms of both efficiency and influence spread, and they are also submodular and monotone.

Let $\mathcal{H}_\mathcal{G}$ be the class of heuristic algorithms for influence approximation that are submodular and monotone, and let $\hat{\sigma}_\mathcal{G} \in \mathcal{H}_\mathcal{G}$ be one of the heuristic methods in $\mathcal{H}_\mathcal{G}$. Now assume that $\hat{\sigma}_\mathcal{G}$ is employed to compute the influence spread. We have the following result.

THEOREM 5. *$\hat{\sigma}_\mathcal{G}(S) \geq 1/2\hat{\sigma}_\mathcal{G}(S_{OPT})$, where S is a seed set returned by PICS based on $\hat{\sigma}_\mathcal{G}$, and S_{OPT} is an optimal seed set whose influence $\hat{\sigma}_\mathcal{G}(S_{OPT})$ is maximum.*

PROOF. Since $\hat{\sigma}_\mathcal{G}$ is submodular and monotone, by the proof of Theorem 4 we have

$$\hat{\sigma}_\mathcal{G}(S_{OPT}) \leq \hat{\sigma}_\mathcal{G}(S) + \sum_{s_i\in S_{OPT}} \hat{\sigma}_\mathcal{G}(\{s_i\}, S_{i-1})^+,$$

where S_i is the set of nodes after the i-th seed node is added to the seed set. For $s_i \in S_{OPT} = \{s_1, \ldots, s_k\}$ and $u_i \in S = \{u_1, \ldots, u_k\}$, we also have $\hat{\sigma}_\mathcal{G}(\{s_i\}, S_{i-1})^+ \leq \hat{\sigma}_\mathcal{G}(\{u_i\}, S_{i-1})^+$. Therefore, it follows that

$$\begin{aligned}\hat{\sigma}_\mathcal{G}(S_{OPT}) &\leq \hat{\sigma}_\mathcal{G}(S) + \sum_{u_i\in S} \hat{\sigma}_\mathcal{G}(\{u_i\}, S_{i-1})^+ \\ &= \hat{\sigma}_\mathcal{G}(S) + \hat{\sigma}_\mathcal{G}(S) = 2\hat{\sigma}_\mathcal{G}(S)\end{aligned} \tag{6}$$

□

5.4 Optimized PICS Algorithm

Observe that the PICS algorithm needs to enumerate all partitions, which can be inefficient. To alleviate this problem, we leverage the notion of cover-groups so that the partitions can be reorganized by their cover-groups. Consequently, as we shall see later, such organization will enable us to develop a Threshold-Algorithm-flavored [6] approach to avoid unnecessary enumeration of partitions. We also show that our optimized algorithm is instance optimal in pruning unnecessary partitions.

5.4.1 Cover-group

We first introduce the notion of cover-group and then elaborate on how we can group partitions using cover-group.

DEFINITION 4 (cover-group). *Consider a set of attributes Q and a partial* partition $P = \{A_1, A_2, \ldots, A_m\}$ *of Q, and let $r_i = |A_i|$ ($i \in [1, m]$) be the number of attributes of A_i. The multiset $R = \{r_1, r_2, \ldots, r_m\}$ is called the* cover-group *of P.*

Example 5: Consider $Q = \{a, b, c, d, e\}$. (1) $P_1 = \{\{c, d\}, \{e\}\}$ is a partial partition of Q and its cover-group is $\{2, 1\}$. (2) $P_2 = \{\{c, e\}, \{d\}\}$ is also a partial partition of Q and it cover-group is $\{2, 1\}$. Observe that although P_1 and P_2 are 2 different partial partitions of Q, they have the same cover-group. □

Recall that for each partition P the seed set returned by the PICS algorithm comprises two parts: free set $S_{\overline{P}}$ and constrained set S_P. Nodes in $S_{\overline{P}}$ are selected from V_Q. Nodes in constrained set S_P are selected based on P', a partial partition of Q, which comprises the remaining attribute sets after removing attributes covered by nodes in $S_{\overline{P}}$. Hereafter, when we say partial partition of a partition, we refer to the partial partition like P'.

We group partitions in two steps. First, we group partitions according to their free sets ($S_{\overline{P}}$), *i.e.,* partitions with the same free set will be put into the same group. This means that for partitions in the same group, their partial partitions contain the same number of attribute sets and covers the same set of attributes. We generate at most $\min(k, |Q|)$ groups in this step because there are at most $\min(k, |Q|)$ free set.Second, for partitions in each group generated in the first step, we further group them based on their partial partitions' cover-groups, *i.e.,* if their partial partitions' cover-groups are the same, they form a group.

Observe that each group has a distinct cover-group and can be uniquely identified by its cover-group, which is shared by all the partitions in the group. In addition, for each group R, its partitions share the same free set, denoted by $S_{\overline{R}}$. For group R, the set of attributes that are not covered by nodes in $S_{\overline{R}}$ are denoted by Q_R; they actually form the attribute sets of the partial partitions of partitions in the group.

Example 6: Consider a set of attributes $Q = \{a, b, c, d, e\}$, $k = 4$ and two partitions of Q, $P_1 = \{\{a, b\}, \{c\}, \{d, e\}\}$ and $P_2 = \{\{a, b, c\}, \{d, e\}\}$. Assume that the first two nodes selected from V_Q can cover $\{c, d\}$. Then for both P_1 and P_2, their free set is the same, and their partial partition is $\{\{a, b\}, \{e\}\}$. Hence, the two partitions are in the same group. □

Given Q, the number of cover-groups is no more than the number of integer partitions of $|Q|$, which is much smaller than the number of partitions [1] (*e.g.,* at $|Q| = 10$, the number of cover-groups is bounded by 42 while the number of partitions can be 115,975). To enumerate all the cover-groups, we enumerate cover-groups for each size. To enumerate cover-groups of size i, we first select $k - i$ nodes from V_Q greedily, which form the free set. Let Q_R be the set of attributes not covered by the $k - i$ nodes in the free set. Attributes in Q_R form partial partitions of size i.

Table 2: Examples for UA.

$List_1(r_1 = 3)$	$List_2(r_2 = 2)$	$List_3(r_3 = 1)$
$t_1 : [\{abc\}, \{v_1\}, 40]$	$t_2 : [\{de\}, \{v_4, v_5\}, 60]$	$t_3 : [\{f\}, \{v_3, v_6\}, 70]$
$t_4 : [\{abe\}, \{v_2\}, 30]$	$t_5 : [\{ab\}, \{v_1, v_2\}, 40]$	$t_6 : [\{d\}, \{v_4, v_5\}, 60]$
$t_7 : [\{bcf\}, \{v_3\}, 20]$	$t_8 : [\{bc\}, \{v_1, v_3\}, 40]$	$t_9 : [\{e\}, \{v_2, v_4, v_5\}, 60]$
$t_{10} : [\{cde\}, \{v_4\}, 10]$	$t_{11} : [\{ac\}, \{v_1\}, 40]$	$t_{12} : [\{a\}, \{v_1, v_2\}, 40]$
	$t_{13} : [\{ae\}, \{v_2\}, 30]$	$t_{14} : [\{b\}, \{v_1, v_2, v_3\}, 40]$
	$t_{15} : [\{be\}, \{v_2\}, 30]$	$t_{16} : [\{c\}, \{v_1, v_3, v_4\}, 40]$
	$t_{17} : [\{bf\}, \{v_3\}, 20]$	
	$t_{18} : [\{cf\}, \{v_3\}, 20]$	
	$t_{19} : [\{cd\}, \{v_4\}, 10]$	
	$t_{20} : [\{ce\}, \{v_4\}, 10]$	

Every way of decomposing $|Q_R|$ as the sum of i positive integers corresponds to a cover-group. In case of $Q_R = \emptyset$ and $i = 0$, the only cover-group is $\emptyset$.

5.4.2 The Upper bound Algorithm (UA)

Observe that each element $R.r_i$ of cover-group R may correspond to many attribute sets, each of which contains $R.r_i$ number of uncovered attributes and may correspond to many nodes that contain the attribute set. For each element $R.r_i$, we order its corresponding attribute sets in descending order of their influence upper bounds to form a list. This enables us to access partial partitions under the cover-group by scanning these lists, and design an algorithm called *Upper bound Algorithm* (UA) to prune enumerating partial partitions, based on the no random access (NRA) algorithm [6] and compute a seed set for each enumerated partial partition. After we process all cover-groups, the seed set with maximum influence spread is returned as the final result.

We now proceed to detail how to generate lists for cover-group R. Each element $R.r_i$ ($1 \le i \le m$) corresponds to a set of attribute sets, each of which contains $R.r_i$ attributes which are not covered by any node in $S_{\overline{R}}$. For each element $R.r_i$, we build a list L_i for its set of attribute sets. Each tuple t in list L_i is associated with three types of information: an attribute set A_t that comprises r_i attributes in Q_R, a set of nodes V_t each of which can cover A_t, and the maximum marginal influence increase of nodes in V_t w.r.t. $S_{\overline{R}}$, *i.e.,* $\max_{v \in V_t} \sigma_{\mathcal{G}}(\{v\}, S_{\overline{R}})^+$. We denote the three types of information of a tuple t by $Attr(t)$, $Node(t)$, and $Inf(t)$, respectively. The tuples in list L_i are ranked in descending order of their maximum marginal influence increase $Inf(t)$. We say $t_1 = t_2$ if and only if (1) $Attr(t_1) = Attr(t_2)$, (2) $Node(t_1) = Node(t_2)$, and (3) $Inf(t_1) = Inf(t_2)$.

Example 7: Consider a network $\mathcal{G}(V, E, w, \mathcal{A})$, where $V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$, and $\mathcal{A}(v_1) = \{a, b, c\}$, $\mathcal{A}(v_2) = \{a, b, e\}$, $\mathcal{A}(v_3) = \{b, c, f\}$, $\mathcal{A}(v_4) = \{c, d, e\}$, $\mathcal{A}(v_5) = \{d, e\}$, and $\mathcal{A}(v_6) = \{f\}$. Let $Q = \{a, b, c, d, e\}$, $k = 3$, cover-group $R = \{3, 2, 1\}$, and $S_{\overline{R}} = \emptyset$. Let the marginal influence increase of v_1, v_2, v_3, v_4, v_5 w.r.t. $S_{\overline{R}}$ be $40, 30, 20, 10, 60, 70$, respectively. We construct $List_1, List_2, List_3$ as shown in Table 2. □

We can also divide a tuple t into several smaller tuples $t_1, t_2, \ldots, t_m$, where (a) $Attr(t_i) = Attr(t)$ ($1 \le i \le m$), (b) $\cup_{i \in [1,m]} Node(t_i) = Node(t)$, and (c) $Inf(t_i)$ is computed within its $Node(t_i)$.

Next, based on the lists of a cover-group R, we present our NRA-inspired UA approach to pruning the enumeration of partitions in R while finding the partition with the maximum influence. Note that NRA, which is proposed to solve the well-known top-k problem, cannot be directly adopted here. Specifically, our goal is to prune the enumeration of partition without missing the result. In contrast to the setting for finding top-k objects, a tuple in a list L_i can be combined with multiple tuples in any other list L_j ($j \neq i$) of R to cover Q_R. Hence, the NRA algorithm (and its variants)

cannot be used directly. To facilitate the exposition of UA, we first define the notion of combination.

DEFINITION 5 (COMBINATION). *Given a* cover-group R *and a set of tuple lists* $\{L_1, \ldots, L_{|R|}\}$*, we call a group of* p $(p \leq |R|)$ *tuples* $t_1, \ldots, t_p$*, each from a distinct list in* $\{L_1, \ldots, L_{|R|}\}$*, a* combination*, denoted by* $\mathcal{T}$*, if* $Attr(t_i) \cap Attr(t_j) = \emptyset$ *for any* $i, j \in [1, p], i \neq j$*. A* combination $\mathcal{T}$ *is called a full* combination *if* $|\mathcal{T}| = |R|$.

We say two combinations $\mathcal{T}_1 = \mathcal{T}_2$ if for each tuple t_i in $\mathcal{T}_1$, there exists a tuple t_j in $\mathcal{T}_2$ such that $t_i = t_j$.

Example 8: Consider Table 2. Here (1) $\{t_1, t_2\}$ is a combination, (2) $\{t_1, t_5\}$ is not a combination because $Attr(t_1) \cap Attr(t_5) = \{ab\}$, (3) $\{t_1, t_4\}$ is not a combination because the tuples come from a single list $List_1$, and (4) $\{t_1, t_2, t_3\}$ is a full combination. □

Our algorithm UA does sorted access in parallel to each of the $|R|$ sorted list L_i. At each depth d (when d tuples have been accessed under sorted access in each list), we maintain (1) all the generated combinations ; (2) the influence values associated with the tuples at depth d in the lists; and (3) the upper bound of influence for each combination $\mathcal{T}$. Note that if two combinations are equal, we only maintain one of them. For a tuple from a list L_i $(1 \leq i \leq m)$ at depth d, we check whether it can be combined with any existing combination to form a new combination.

Once a combination becomes a full combination, $Attr(t)$ for all its tuple t together forms a partial partition of Q_R. We then compute a upper bound influence value using Definition 3. If the upper bound is better than the best seed set so far, we compute the seed set for the combination to check if it has a larger influence, and prune it, otherwise. For each combination, we maintain the upper bound of its marginal influence increase using Definition 6.

DEFINITION 6 (UPPER BOUND FOR combination). *Given* R*,* $S_{\overline{R}}$*, and depth* d*, let* $x_1^d, \ldots, x_m^d$ *be the influence values associated with the tuples at depth* d *in lists* $L_1, \ldots, L_m$*, respectively. Consider a* combination $\mathcal{T}$*, and assume that* $L_{i_1}, \ldots, L_{i_j}$ $(j = |R| - |\mathcal{T}|)$ *are the set of lists from which we have not selected any tuple for* $\mathcal{T}$*. The upper bound marginal influence increase w.r.t.* $S_{\overline{R}}$ *of* $\mathcal{T}$ *at depth* d *is* $upper(\mathcal{T})^d = \sum_{t \in \mathcal{T}} Inf(t) + x_{i_1}^d + \cdots + x_{i_j}^d$.

Example 9: Consider the lists in Table 2. Let $\mathcal{T} = \{t_1, t_2\}$ and $List_3$ be the list from which no tuples are selected for $\mathcal{T}$. Assume that we access tuples at depth 4. Then we have $x_3^4 = 40$. Hence $upper(\mathcal{T})^4 = inf(t_1) + inf(t_2) + x_3^4 = 40 + 60 + 40 = 140$. □

If the upper bound of marginal influence increase of $\mathcal{T}$ is smaller than the currently best marginal influence increase, we can safely prune the combination, *i.e.,* any full combination extended from it cannot provide a better result. The drop condition is given in Definition 7. When no combination is left, our algorithm UA halts.

DEFINITION 7 (DROP-CONDITION). *Given a set of lists* L_1*, ...,* L_m*, a set of nodes* $S_{\overline{R}}$*, and a* combination $\mathcal{T}$*, let* Inf_{max} *be the currently best influence spread of any known seed set. We drop* $\mathcal{T}$ *at depth* d *if either of the following conditions is satisfied:*

1. $upper(\mathcal{T})^d + \sigma_{\mathcal{G}}(S_{\overline{R}}) < Inf_{max}$*; or*
2. *There exists a list* L_i *from which we have not selected any tuple for* $\mathcal{T}$ *such that the end of* L_i *is reached at depth* d.

Example 10: Consider the lists in Table 2. Suppose $S_{\overline{R}} = \emptyset$, $\sigma_{\mathcal{G}}(S_{\overline{R}}) = 0$, and the currently best influence spread of known seed sets is 165. (1) Assume that we access tuples at depth 3, and $\mathcal{T}_1 = \{t_1, t_2\}$ is a combination. Its upper bound influence at depth 3 is $upper(\mathcal{T}_1)^3 = Inf(t_1) + Inf(t_2) + x_3^3 = 40 + 60 + 60 = 160$. It is smaller than the known best influence 165, and thus we drop $\mathcal{T}_1$ at depth 3. (2) Assume that we access tuples at depth 5, and $\mathcal{T}_2 = \{t_2, t_3\}$ is a combination. The end of list $List_1$ is reached at depth 5, and thus $\mathcal{T}_2$ can be dropped. □

Algorithm 3: UA

Input: $\mathcal{G}, S_{\overline{R}}, L_1, \ldots, L_m, S$
Output: Seed set S

```
1  d ← 0, C ← {∅};
2  repeat
3      for each T ∈ C do
4          if T satisfies the drop-condition then
5              remove T from C;
6      for tuple t_i^d in each list L_i at depth d do
7          for each T ∈ C do
8              T' = T ∪ {t_i^d};
9              if T' is a full combination then
10                 S_T' ← PartitionSelect(S_{k-m}, G, Nodes(T'));
11                 if σ_G(S_T' ∪ S_R̄) > σ_G(S) then
12                     S ← S_T' ∪ S_R̄;
13             else
14                 C ← C ∪ {T'};
15     d ← d + 1;
16 until C = ∅;
17 return S
```

Algorithm 3 presents the pseudo code of UA. It takes as input (1) network $\mathcal{G}$, (2) the node set $S_{\overline{R}}$ consisting of $k - |R|$ nodes selected from V_Q, (3) the set of lists $L_1, \ldots, L_{|R|}$, and (4) the seed set S with maximum influence spread known so far. In UA, we use the following variables: d indicates the depth of lists that UA is accessing; and C keeps track of all active combinations. At depth d, we first check each combination in C and remove it from C if it satisfies the drop-condition(line 3–5). Then for tuple t_i^d in each list L_i, UA checks each combination in C (lines 6–14). Specifically, it adds t_i^d to each combination to get a new combination (line 8). If a new combination is a full combination, then UA invokes PartitionSelect to compute $S_{\mathcal{T}'}$, where argument $Nodes(\mathcal{T})$ represents the set of node sets for combination $\mathcal{T}$, each node set corresponding to $Nodes(t)$, $t \in \mathcal{T}$; Otherwise, we add $\mathcal{T}'$ to C (line 14). We update the best known seed set S if the influence of $S_{\mathcal{T}'} \cup S_{\overline{R}}$ is larger than $\sigma_{\mathcal{G}}(S)$(lines 11–12). We increase d to access tuples at next level (line 15) until C is empty. Finally, we return S.

Example 11: Consider UA runs on the three lists in Table 2: Initially, C comprises an $\emptyset$. At depth 1, it computes upper bound influence for $\emptyset$, representing a combination, which is $40 + 60 + 70 = 170$. (1) Assume that the currently known best influence $\sigma_{\mathcal{G}}(S)$ is 180. UA first remove $\emptyset$ from C because of the drop-condition and then halts. (2) Assume that $\sigma_{\mathcal{G}}(S)$ is 150. UA will find 7 combinations: $\mathcal{T}_1 = \{t_1\}$, $\mathcal{T}_2 = \{t_2\}$, $\mathcal{T}_3 = \{t_1, t_2\}$, $\mathcal{T}_4 = \{t_3\}$, $\mathcal{T}_5 = \{t_1, t_3\}$, $\mathcal{T}_6 = \{t_2, t_3\}$ and $\mathcal{T}_7 = \{t_1, t_2, t_3\}$. Since $\mathcal{T}_7$ is a full combination, UA invokes PartitionSelect to compute a seed set for $Nodes(\mathcal{T}_7)$. Assume the seed set is $\{v_1, v_5, v_6\}$ and its influence is 165. We update S and now $\sigma_{\mathcal{G}}(S) = 165$. Then at depth 2,UA next checks the combinations in C. At depth 2, $upper(\emptyset)^2 = 130$, $upper(\mathcal{T}_1)^2 = 140$, $upper(\mathcal{T})^2 = 150$, $upper(\mathcal{T}_3)^2 = 160$, $upper(\mathcal{T}_4)^2 = 140$, $upper(\mathcal{T}_5) = 150$, $upper(\mathcal{T}_6) = 160$. The upper bound of every combination in C is smaller than the best known influence 165, and thus all of them are dropped at depth 2 and C is empty. Therefore the algorithm terminates. □

We next consider the optimality of Algorithm UA.

Algorithm 4: PICS+

Input: $\mathcal{G}, Q, k$
Output: Seed set S
1 $S \leftarrow \emptyset$;
2 **if** $|V_Q| < k$ **then return** $\emptyset$;
3 **foreach** j *from* 1 *to* k **do**
4 $\quad SP[j] \leftarrow \arg\max_{u \in V_Q - S'} \sigma_{\mathcal{G}}(u, S')^+; S' = S' \cup \{SP[j]\}$;
5 **foreach** cover-group R **do**
6 $\quad S_{\overline{R}} = \cup_{i \in [1, k-|R|]} \{SP[i]\}$;
7 $\quad$ Build lists $L_1, \ldots, L_{|R|}$;
8 $\quad S \leftarrow \texttt{UA}(\mathcal{G}, S_{\overline{R}}, L_1, \ldots, L_{|R|}, S)$;
9 **return** S;

THEOREM 6. *Given a* cover-group R *and a set of nodes* $S_{\overline{R}}$*, let* $\mathbb{D}$ *be the class of all generated* $|R|$ *lists* $L_1, \ldots, L_{|R|}$ *for* R*. Let* $\mathbb{A}$ *be the class of correct algorithms that run on every* $\{L_1, \ldots, L_m\}$ *using sorted access to deterministically find the full* combination *with maximum marginal influence increase with respect to* $S_{\overline{R}}$*, which is computed by* PartitionSelelct*. Apparently, algorithm* UA *is in* $\mathbb{A}$*. We say* UA *is instance optimal over* $\mathbb{A}$ *and* $\mathbb{D}$*, i.e. for any algorithm* $\mathscr{A} \in \mathbb{A}$ *and any* $\{L_1, \ldots, L_m\} \in \mathbb{D}$*,* $cost(\mathsf{UA}, \{L_1, \ldots, L_m\}) \leq c \times cost(\mathscr{A}, \{L_1, \ldots, L_m\})$*, where* $cost(\mathsf{UA}, \{L_1, \ldots, L_m\})$*(resp.* $cost(\mathscr{A}, \{L_1, \ldots, L_m\})$*) is the number of tuples seen by* UA*(resp.* $\mathscr{A}$*) and* c *is a constant.*

PROOF. Assume that UA halts at depth d on $D_1 = \{L_1, \ldots, L_m\}$. We will show that $\mathscr{A}$ must get to depth d in at least one list in D_1; otherwise $\mathscr{A}$ cannot always return the correct answer. For contradiction, we assume that $\mathscr{A}$ can always return the correct answer and it does not get to depth d in any list in D_1. We will show the contradiction by constructing a $D_2 \in \mathbb{D}$ on which $\mathscr{A}$ cannot find the correct answer.

Since UA halts at depth d on D_1, there exists a combination $\mathcal{T}$ such that $upper(\mathcal{T})^{d-1} > \sigma_{\mathcal{G}}(S)$. We construct $D_2 = \{L'_1, \ldots, L'_m\}$ as follows: L'_i is identical to L_i up to depth $d-1$ (that is, for each $i \in [1, m]$, the top $d-1$ tuples in L'_i are the same as in L_i). Run UA on D_2. There also exists a combination $\mathcal{T}'$ such that $\mathcal{T}'$ and $\mathcal{T}$ contain the same attribute sets. Let $L'_{j_1}, \ldots, L'_{j_x} (x = |R| - |\mathcal{T}'|)$ be the set of lists from which we have not selected any tuple for $\mathcal{T}'$. We let the attribute sets in the tuples at depth d in $L'_{j_1}, \ldots, L'_{j_x}$ be the particular attribute sets such that $\mathcal{T}'$ can become a full combination $\mathcal{T}_{full}$ if we expand it with these tuples. Let $Inf(t_{j_i})$ for tuple t_{j_i} at depth d in L'_{j_i} be $x_{j_i}^{d-1}$, where $x_{j_i}^{d-1}$ is the influence part of the tuple at depth $d-1$ in list L_{j_i}. So $upper(\mathcal{T}_{full}) > Inf_{max}$. Assume the seed set of $\mathcal{T}$ has a larger marginal influence increase than Inf_{max}. Since the top $d-1$ tuples in each list in D_2 are the same as those in D_2, and $\mathscr{A}$ does not get to depth d in any lists, the seed sets returned by $\mathscr{A}$ on D_1 and D_2 are the same. So $\mathscr{A}$ cannot find the correct answer on D_2, contradicting that $\mathscr{A}$ is a correct algorithm. Therefore $\mathscr{A}$ must get to depth d in at least one list in D_1. $cost(\mathsf{UA}, L_1, \ldots, L_m) \leq m \cdot cost(\mathscr{A}, L_1, \ldots, L_m)$, where m is the size of R and is smaller than constant $|Q|$. □

5.4.3 The PICS+ Algorithm

Finally, we present the PICS+ to process all cover-groups, for each of which we use UA to find a seed set for partial partitions under the cover-group and to update the known best seed set. Note that some cover-group may not contribute to the final seed set (*i.e.,* it is pruned) as illustrated in Example 11. The pseudo code of PICS+ is shown in Algorithm 4. In PICS+, we use S to record the best known seed set. It first selects k nodes greedily from V_Q and store them in an array SP (lines 3–4). Then for each cover-group, it uses array SP to populate $S_{\overline{R}}$, which stores the nodes selected from V_Q (line 6). The algorithm next constructs lists $L_1, \ldots, L_{|R|}$ for each element in the cover-group R (line 7). It invokes UA to update the best seed set S (line 8). After all cover-groups are considered, the algorithm returns S as the final result.

Correctness. This can be verified by showing that the seed set returned by PICS+ is the same as the one returned by PICS.

Table 3: Dataset properties.

Property	FX	PC	DBLP	MeetUp
# of nodes	38,834	76,665	874,305	1,013,453
# of edges	164,093	1,702,058	9,415,206	34,410,754
# of distinct attr.	37,036	103,289	89,975	64,721
avg. # of attr. per node	47	9	27	11

6. PERFORMANCE STUDY

6.1 Experimental Setup

Datasets. We use 4 datasets in our experiments and their properties are given in Table 3. MeetUp is crawled from an event-based social network *meetup.com* from July 2013 to Oct 2013. Each node represents a user and the interests specified by the user form the attributes of each node. Two users are connected by an edge if they appear in the same group at least three times. PC[2] is a real-life event-based social network from *plancast.com*. Each node represents a user and each edge represents a subscription between two users. We group the events into clusters and the clusters containing the events that a user attended form the attributes of the node. DBLP is a real-life network from DBLP. Each node represents an author, and the attributes of a node comprise the words in the author's publication titles after removing stopping words and stemming. An edge represents the co-author relationship between two authors. Lastly, FX[3] is a dataset of movie ratings from *flixster.com* in 2008. Each node represents a user and the movies that have been rated by the user form the attributes of each user.

Query. A query contains an integer k and a set of attributes Q. We use a pair of integers (x, y) to denote a group of queries where $x = k$ and $y = |Q|$. Attributes in Q are randomly generated such that there exists a set of k seed nodes to cover all attributes in Q. Each group comprises 30 queries.

Evaluated algorithms. The following algorithms are evaluated. (a) ScoreGreedy algorithm, denoted by SG; (b) PigeonGreedy algorithm, denoted by PG; and (c) PICS+ algorithm. Recall that SG and PG may not always return a seed set that can cover all attributes in the query even if such a set exists. In contrast, PICS+ guarantees to return a seed set that can cover all attributes in the query if such a set exists. The parameter α in SG is set at 0.6 since it usually achieves the best *success rate* (defined below) according to our experiments. We use a state-of-the-art method, IRIE [10], to approximate the influence spread. We also evaluate the performance when MC simulation is used to compute the influence.

Note that there is no existing algorithm that addresses the proposed ICS problem. Nevertheless, since one of our goals is to demonstrate that the seed set generated by our proposed techniques cannot be found by existing influence maximization (IM) approaches, we confine ourselves to compare the PICS+ against a state-of-the-art IM technique, IRIE. All algorithms are implemented in C++. All experiments are run on a Windows PC with Intel Xeon 2.66 GHZ 6-core CPU and 24 GB memory.

Performance measures. We consider three measures. (a) The *success rate* = a/b, where b is the number of queries in a group,

[2] http://lsna2012.net76.net/ebsn/
[3] http://www.cs.sfu.ca/~sja25/personal/datasets/

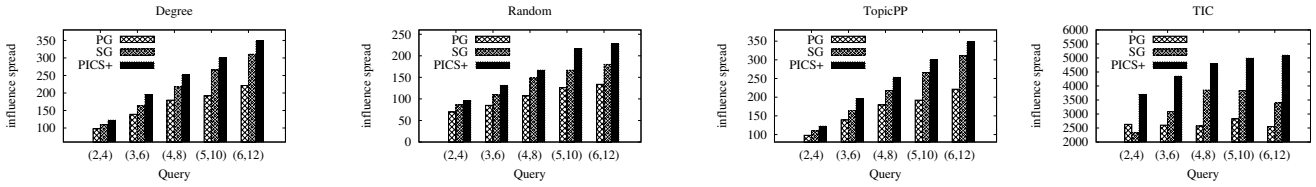


Figure 2: Influence spread under influence probabilities generated by four methods.

Table 4: Success rate of SG and PG.

Dataset	Method	(2,4)	(3,6)	(4,8)	(5,10)	(6,12)
FX	SG	0.87	0.77	0.87	0.83	0.87
	PG	0.77	0.67	0.7	0.57	0.6
PC	SG	0.47	0.53	0.53	0.5	0.43
	PG	0.63	0.36	0.33	0.23	0.1
DBLP	SG	0.47	0.37	0.53	0.47	0.5
	PG	0.5	0.17	0.23	0.27	0.27
MeetUp	SG	0.33	0.2	0.13	0.2	0.03
	PG	0.47	0.0	0.03	0.03	0.03

Table 5: Comparison with an IM technique: Jaccard similarity (JC) and attribute coverage rate (ACR).

Dataset		(2,4)	(3,6)	(4,8)	(5,10)	(6,12)
FX	JC	0.033	0.02	0.01	0.011	0.021
	ACR	3.3%	1.1%	0.8%	2.3%	1.7%
PC	JC	0.033	0.033	0.040	0.054	0.031
	ACR	2.5%	1%	12.5%	10%	0.6%
DBLP	JC	0.044	0.033	0.056	0.079	0.096
	ACR	25.8%	33.9%	31.3%	34%	37.2%
MeetUp	JC	0.0	0.0	0.005	0.0	0.0
	ACR	0%	0.5%	3%	1%	0.8%

and a is the number of queries in the group for which an algorithm is able to find a seed set (satisfying attribute constraints). (b) The influence spread of the seed set. We apply 20,000 Monte Carlo simulations and the average number of influenced nodes is used as the influence spread of the seed set. (c) The runtime of algorithms.

Propagation probability. As remarked earlier, our solutions to the ICS problem are orthogonal to the techniques for generating influence probability. Hence, we consider four methods of generating propagation probability. (a) Degree. The probability on edge (u, v) is set to be $\frac{1}{N_{in}(v)}$, where $N_{in}(v)$ is the in-degree of v. (b) Random. The probability on edge (u, v) is randomly selected from $\{0.1, 0.01, 0.001\}$. (c) TopicPP. We first compute a $p_{u,v}$ for each edge (u, v) as $p_{u,v} = \max\left(\frac{|\mathcal{A}(u)|\cdot|\mathcal{A}(v)|\cdot|\mathcal{A}(u)\cap\mathcal{A}(v)|}{|Q|^3}, 0.001\right)$. Then the probability on edge (u, v) is set to be $\frac{p_{u,v}}{\sum_{(s,v)\in E} p_{s,v}}$. (d) TIC. We adopt the TIC model proposed in [3] and learn the propagation probability from the action log of FX. For each movie a, we can compute the probability $p^a_{u,v}$ that u activates v according to the model. Given a query $Q = \{a_1, ..., a_q\}$, the weight of edge (u, v) is defined as $1-\prod_{a_i\in Q}(1 - p^{a_i}_{u,v})$. Note that the remaining three datasets do not provide historical action logs information.

We study the effects of the propagation probability generated by four methods on influence spread. We run 5 groups of queries, $(2, 4)$, $(3, 6)$, $(4, 8)$, $(5, 10)$, and $(6, 12)$. Figure 2 shows the average influence spreads of the three evaluated algorithms on FX. We can see that the relative performance of the four algorithms in terms of influence spread is consistent over the different methods of generating propagation probabilities. The relative performance of the four algorithms in terms of success rate and efficiency is also consistent, irrespective of methods of generating the propagation probability.

We also investigated the impact of adjusting the propagation probability between two nodes based on their degrees. Our results show that it does not significantly affect the relative performances of the algorithms. We omit the detailed results due to space constraint. *In the sequel, the propagation probability is generated by TopicPP.*

Table 6: Success rate of SG and PG (Varying $|Q|$).

		|Q|							
Dataset	Method	8	9	10	11	12	13	14	15
FX	SG	0.83	0.9	0.83	0.93	0.9	0.8	0.83	0.63
	PG	0.63	0.83	0.57	0.8	0.73	0.63	0.57	0.5
PC	SG	0.7	0.6	0.5	0.27	0.33	0.57	0.2	0.37
	PG	0.63	0.7	0.23	0.13	0.2	0.27	0.03	0.13
DBLP	SG	0.77	0.53	0.47	0.23	0.3	0.17	0.23	0.13
	PG	0.67	0.3	0.27	0.2	0.1	0.03	0.1	0.0
MeetUp	SG	0.43	0.26	0.2	0.23	0.33	0.03	0.2	0.1
	PG	0.53	0.26	0.03	0.2	0.2	0.13	0.13	0.0

Table 7: Success rate of SG and PG (Varying k).

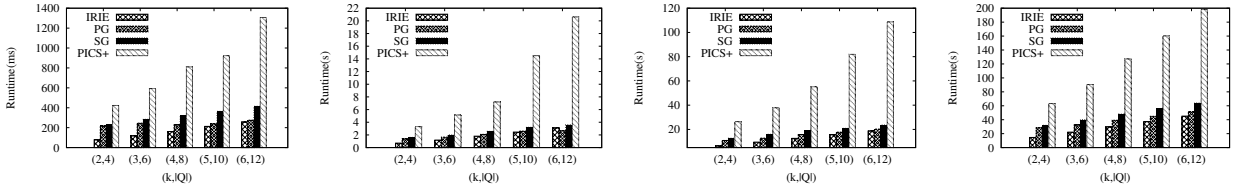
		k		
Dataset	Method	5	6	$7-13$
FX	SG	0.83	1.0	1.0
	PG	0.56	0.96	1.0
PC	SG	0.5	0.93	1.0
	PG	0.23	0.83	1.0
DBLP	SG	0.47	0.93	1.0
	PG	0.27	0.73	1.0
MeetUp	SG	0.2	0.47	1.0
	PG	0.03	0.6	1.0

6.2 Experimental Results

Comparison with IM *techniques.* First, in this set of experiments we compare the seed sets generated by our algorithm (PICS+) and a traditional IM technique (we use the IRIE method) for each query. Specifically, we measure the (1) Jaccard similarity between the seed sets returned by these two techniques, (2) the *attribute coverage rate* of the seed sets returned by traditional IM technique, which is defined as $\frac{|\bigcup_{s\in S}\mathcal{A}(s)\cap Q|}{|Q|}$, and (3) the runtime of these two techniques. Table 5 reports the results. Observe that there is very low similarity between the seed sets returned by these two techniques. We can also observe that only a small portion of attributes in the query can be covered by the seed sets returned by the traditional IM technique. Hence we conclude that traditional IM techniques are not suitable for addressing the ICS problem. The runtime of IRIE and PICS+ is reported in Figure 3. As expected, IRIE is faster because it does not take into account the attribute covering constraint.

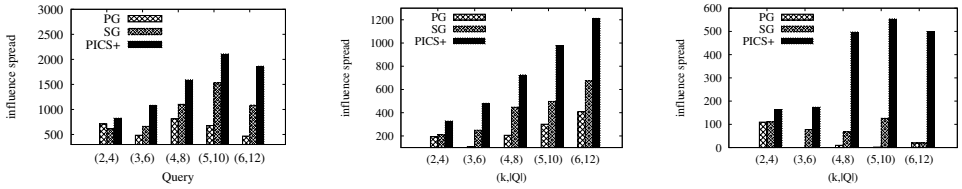
Overall Performance. Next, we evaluate the performance of the three proposed methods. We run 5 groups of queries, $(2, 4)$, $(3, 6)$, $(4, 8)$, $(5, 10)$, and $(6, 12)$ for each dataset. Table 4 shows the success rates of SG and PG in finding a seed set covering all the attributes in a query for each query group containing 30 queries. Note that PICS+ guarantees to find such a seed set, and thus its success rate is 1. The success rates of SG and PG often are around 0.5, which means they fail on half of 30 queries in a group. On some query groups, the success rate of PG can be as low as 0.17. This indicates that the greedy algorithms may not be a good choice for the ICS problem, especially when k is smaller than $|Q|$.

Figure 4 shows the influence spread of the three algorithms on PC, DBLP and MeetUp (Figure 2 gives results on FX). If SG or PG cannot find a seed set to cover attributes in some query, it returns an empty set whose influence spread is 0. We can see that the influence spread of PICS+ is consistently better than the other two methods.



(a) FX (b) PC (c) DBLP (d) MeetUp

Figure 3: Runtime vs. query groups.



(a) PC (b) DBLP (c) MeetUp

Figure 4: The influence spread vs. query groups.

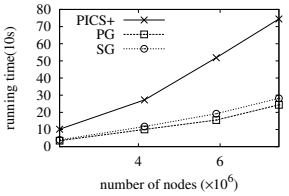


Figure 5: Runtime vs. graph size.

Figure 3 shows the runtime of different algorithms. Although PICS+ can finish in reasonable time for the largest dataset, it is less efficient than SG or PG as expected.

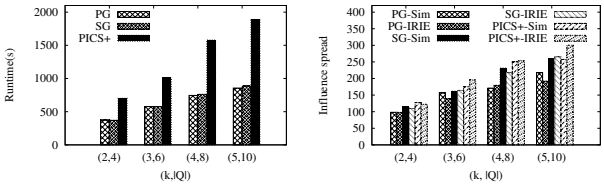
Effect of $|Q|$. This set of experiments is conducted to study the effect of $|Q|$. We set $k = 5$ and vary $|Q|$ from 8 to 15. Table 6 shows the success rates of SG and PG. We observe that SG and PG may fail to return a seed set for a significant portion of queries. We also observe that the success rate usually drops as $|Q|$ increases, as expected. This is because it is more difficult to cover all the attributes as $|Q|$ increases.

Figure 6 shows the influence spread with different $|Q|$. PICS+ outperforms the other two methods consistently; When $|Q|$ gets larger, the disparity between PICS+ and the other two methods become larger. In addition, the influence spread usually drops as $|Q|$ increases on `PC`, `DBLP`, and `Meetup` . This is because when $|Q|$ is small, it is relatively easier to cover the attributes in Q, and thus we have better chances to choose nodes with high influence while satisfying the attribute constraint. On `FX`, the influence spread is less affected by $|Q|$ since nodes in `FX` can cover many attributes and it is easier for SG and PG to find a good seed set. Although PICS+ is usually 2-3 times slower than SG and PG, PICS+ still can finish in reasonable time even for the largest network. The runtime of all three methods increases slightly with the increase of $|Q|$. Due to the space limitation, we do not show the runtime.

Effect of k. We evaluate the effect of k by varying it from 5 to 13 while fixing the query attributes in Q and $|Q|$ at 10. As shown in Figure 7, PICS+ consistently achieves better influence than the other two methods. However, as k increases, the gap becomes smaller. This is because when k gets larger, it becomes easier to cover all attributes and the constraints of attributes are weakened. Note that in our problem typically $k < |Q|$ as the number of desired event organizers is small. We also observe that SG outperforms PG when $k < |Q|$, and they perform similarly when $k \geq |Q|$.

PICS+ is usually 2-3 times slower than SG and PG that take similar time. The runtime of PICS+ is almost not affected by k, but SG and PG become slower as k increases. Due to the space limitation, we do not give the runtime results.

The success rate of SG and PG is given in Table 7. As expected, the success rate approaches to 1 when the value of k approaches $|Q|$ since it becomes relatively easier to cover attributes in Q. However, we need to temper the observation. Here we use the same set of query attributes for different k values, and each query Q is generated such that Q can be covered in the case of $k = 5$, which means that those queries are easy queries for $k > 5$ (their attributes are easier to be covered for $k > 5$). This also explains why success rate of SG and PG can be 1 at $k = 7$. Nevertheless, the results indicate that when $k \geq |Q|$, SG and PG are a better choice. However, as remarked earlier, k is usually small in our target applications.



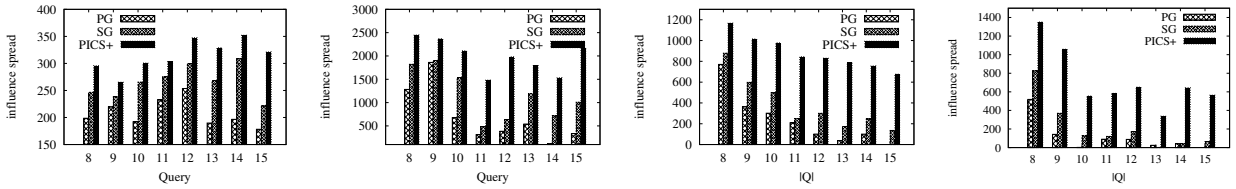
(a) Running time (b) Influence spread

Figure 8: Computing influence by MC simulations.

Comparison between PICS *and* PICS+. We compare the efficiency of PICS+ and PICS. Note that they return the same results and thus have the same influence spread (their success rate is 1). Figure 9 reports the runtime of the two methods for processing 4 groups of queries, $(2, 4), (3, 6), (4, 8), (5, 10)$ on `FX`. Note the y-axis is in logarithmic scale. We observe that PICS+ is orders of magnitudes faster than PICS, especially when $|Q|$ becomes larger. The results on the other datasets are qualitatively similar.

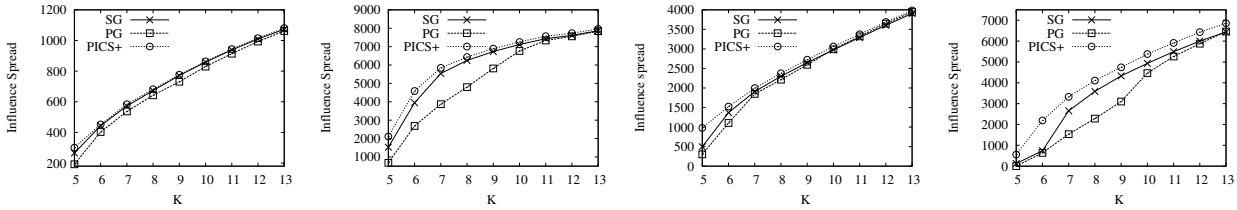
Computing Influence by MC Simulations. This experiment is to study whether the relative performance of the three methods is similar when MC Simulations are used to estimate influence. In terms of the success rate, the results using MC simulations are almost the same as those using IRIE reported in Table 4. Figure 8 shows the runtime and influence spread of the three methods on dataset `FX`. Experiment for group $(6, 12)$ cannot finish in 5 hours and we terminate it and do not get its runtime. We observe that the relative influence spread of the three methods using MC simulations is similar as that using IRIE. Actually, MC simulations achieve similar influence spread as does IRIE. In terms of runtime, the relative performance of the three methods is also similar; however, by comparing Figure 8a and Figure 3a, we observe that the runtime using MC simulations is much longer than that using IRIE. The results on the other datasets are qualitatively similar.

Scalability with graph size. We extract subgraphs from `MeetUp`. On each graph, we run 5 groups of queries, $(2, 4), (3, 6), (4, 8), (5, 10), (6, 10)$. We only report the runtime for $(4, 8)$ in Figure 5. The trend on other groups are similar. We find that all the three algorithms scale well with the graph size.



(a) FX (b) PC (c) DBLP (d) MeetUp

Figure 6: Influence spread vs. $|Q|$ ($k = 5$).



(a) FX (b) PC (c) DBLP (d) MeetUp

Figure 7: Influence spread vs. k ($|Q| = 10$).

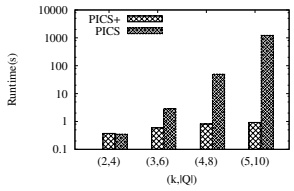


Figure 9: Runime of PICS and PICS+.

7. CONCLUSIONS

The quest for organizing successful social events has become more pressing due to the rapid growth of event-based online social services. In this paper, we introduce the problem of discovering influential event organizers from an online social network. We formulate it as the problem of mining influential cover set (ICS), which is intractable. Hence, we propose two greedy algorithms, ScoreGreedy and PigeonGreedy, to find approximate solutions to the problem. Although they are efficient, they have no guarantee to find a feasible solution. This led us to propose an approximation algorithm called PICS+ that guarantees to find a feasible solution if any. The experimental results on real-world datasets demonstrate that PICS+ guarantees to find a feasible solution if any within reasonable time. Although the two greedy algorithms run faster, they fail to find feasible solution for a significant portion of queries.

Acknowledgments. This work was supported in part by a Singapore MOE AcRF Tier 2 Grant (ARC30/12), a Singapore MOE AcRF Tier 1 Grant RG 66/12, a Singapore MOE AcRF Tier-1 Grant RG24/12, Interactive and Digital media Programme Office(IDMPO) and National Reseaerch Foundation (NRF) hosted at Media Development Authority(MDA) under Grant No.: MDA/IDM/2012/8/8-2 VOL 01. Ma was supported in part by NSFC grant (No. 61322207), 973 program grant (No. 2014CB340304), MOST grant (No. 2012BAH46B04), SRF for ROCS, SEM,

8. REFERENCES

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. 1964.

[2] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi. Online team formation in social networks. In *WWW*, 2012.

[3] N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. In *ICDM*, 2012.

[4] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. KDD, pages 1029–1038, 2010.

[5] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *SIGKDD*, pages 199–208, 2009.

[6] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[7] A. Goyal, F. Bonchi, and L. V. Lakshmanan. A data-based approach to social influence maximization. *VLDB*, 5(1):73–84, 2011.

[8] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250, 2010.

[9] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 1994.

[10] K. Jung, W. Heo, and W. Chen. Irie: Scalable and robust influence maximization in social networks. In *ICDM*, 2012.

[11] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, 2003.

[12] J. Kim, S. K. Kim, and H. Yu. Scalable and parallelizable processing of influence maximization for large-scale social networks? In *ICDE*, 2013.

[13] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *SIGKDD*, pages 467–476, 2009.

[14] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *SIGKDD*, pages 420–429, 2007.

[15] B. Liu, G. Cong, D. Xu, and Y. Zeng. Time constrained influence maximization in social networks. In *ICDM*, pages 439–448, 2012.

[16] X. Liu, Q. He, Y. Tian, W.-C. Lee, J. McPherson, and J. Han. Event-based social networks: linking the online and offline social worlds. In *SIGKDD*, pages 1032–1040, 2012.

[17] A. Majumder, S. Datta, and K. Naidu. Capacitated team formation problem on social networks. In *SIGKDD*, pages 1005–1013, 2012.

[18] R. Raj, P. Walters, and T. Rashid. *Events Management: Principles and Practice*. SAGE Publications Ltd, 2 edition, Feb. 2013.

[19] K. Saito, M. Kimura, K. Ohara, and H. Motoda. Selecting Information Diffusion Models over Social Networks for Behavioral Analysis. In *ECML/PKDD*. 2010.