A Preselection Algorithm for the Influence Maximization Problem in Power Law Graphs

Renato S. Melo
Federal University of Paraná
Curitiba, Paraná, Brazil
rsmelo@inf.ufpr.br

Andre L. Vignatti
Federal University of Paraná
Curitiba, Paraná, Brazil
vignatti@ufpr.br

ABSTRACT

The influence maximization problem in social networks seeks out a set of nodes that allows spreading information to the greatest number of members. A greedy algorithm, proposed by Kempe *et al.* [12], finds a solution in which the spread of influence is at least $1 - \frac{1}{e}$ of the optimum. However, some shortcomings of this approach negatively affect the run time of this algorithm. In this work, we propose a methodology to speedup the Kempe's algorithm with focus on power law graphs. The improvement consists of choosing the most promising nodes in advance. To this end, we explore some properties of power law graphs and the relationship between social influence and degree distribution. We have verified by experimental analysis that this preselection reduces the run time while preserving the quality of the solution.

CCS CONCEPTS

• **Theory of computation** → **Social networks**; • **Applied computing** → *Law, social and behavioral sciences*;

KEYWORDS

Influence maximization; Social networks; Power law graphs

ACM Reference Format:
Renato S. Melo and Andre L. Vignatti. 2018. A Preselection Algorithm for the Influence Maximization Problem in Power Law Graphs. In *SAC 2018: SAC 2018: Symposium on Applied Computing , April 9–13, 2018, Pau, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3167132.3167322

1 INTRODUCTION

In social networks, the phenomenon of diffusion of ideas, behaviors and innovations has the property of always beginning with a small group of *early adopters* [13]. From this group, more and more people adopt the same behavior by observing that their friends, neighbors or colleagues have already done so. So an information spreads like an epidemic. A problem arising from the investigation of this type of social influence is the *influence maximization*, which appears in the context of a chain adoption of new behaviors [7]. Informally, the influence maximization problem aims to find a set S of fixed size, such that the influence of S is the largest possible. Our preselection heuristic intends to find a subset of vertices, called set of candidates, in which we can select all the early adopters without losing quality of spread. Thus, it is not necessary to explore every vertex of the graph during the search for the most influential ones.

The influence maximization problem depends on theoretical models for the formal definition, the basic models for information spreading are Linear Threshold (LT) and Independent Cascade (IC) [12], and in this work we consider the IC model. Let us consider a social network as a graph $G = (V, E)$, where V is the set of individuals and E is the set of relationships between these individuals. As is done in [12, 13], the behaviors modeled here are *progressive*, that is, each vertex can assume one of two states, *active* or *inactive*, and can change from inactive to active, but not from active to inactive. At time $t = 0$, a subset S of V is chosen as the set of early adopters. When a vertex v becomes active because of S, we say that v has been influenced by S. From now onward we only consider *directed* graphs, such that for two nodes v and w the influence of v to w is different from the influence of w to v. So in the IC model each edge has an activation probability and the influence spreads through active vertices. Each active vertex can activate independently its inactive neighbors based on the probability at the edges [4]. The adoption process starts from a set S of active nodes and unfolds into discrete time steps. When the vertex v becomes active in step t, it has a chance to activate each inactive neighbor w, with a probability $p_{v,w}$ of success. If v succeeds, w is activated in step $t + 1$, but if v fails it cannot try to activate w in subsequent rounds [12].

Problem 1. *Given a directed weighted graph $G = (V, E)$, an influence model m and an integer $1 \leq k \leq |V|$. Find a subset $S^* \subseteq V$ such that $\sigma_m(S^*) = \max_{S \subseteq V}\{\sigma_m(S)\}$ subject to $|S| = k$.*

The function $\sigma_m : 2^V \to \mathbb{R}$ to be maximized is called *influence function*, where m is an influence model, such as IC and LT. Thus, given a set $S \subseteq V$ of early adopters, $\sigma_m(S)$ denotes the expected number of active vertices at the end of the activation process starting from S [9, 12, 13]. Therefore, the Problem 1 defines the influence maximization problem. This problem is NP-hard both as IC to LT models [12]. To get an approximation guarantee, Kempe et al. [12] have shown that the σ function is submodular and monotone for IC and LT. Due to these properties, a greedy algorithm that iteratively chooses the vertex with greatest marginal gain can be good enough. The Algorithm 1 shows the pseudo code. However, two major sources of inefficiency affect this algorithm. First, the processing time of $\sigma_m(S)$ function is too high, since to get the exact value of σ is a #P-hard problem on LT and IC models [3, 5, 12]. Second, the algorithm makes many calls to σ.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC 2018, April 9–13, 2018, Pau, France
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167322

Algorithm 1: GREEDY

Input: $G = (V, E), k \in \mathbb{N}, \sigma_m$
Output: Seed set S

```
1 begin
2     S = ∅
3     while |S| ≤ k do
4         u = argmax_{w∈V\S}{σ_m(S ∪ {w}) − σ_m(S)}
5         S ← S ∪ {u}
```

Related Work: To overcome the inefficiency of the Algorithm 1, several works propose improvements and reduction of the computational cost. Two algorithms, CELF [14] and CELF++ [10], stand out for providing good results using Monte Carlo simulations. The main idea of the CELF algorithm is that the marginal gain of a vertex at a given iteration can not be greater than its gain in the previous iterations. The algorithm maintains a list of vertices sorted by the marginal gain in a non-increasing order. CELF++ proposes new settings to CELF. The central idea is that, if the last selected vertex is still the first on the sorted list, then the marginal gain of such vertex does not need to be recomputed. Arora et al. [2] explains that besides the Monte Carlo based methods, there are well-known heuristics that use a method called *Score Estimation* to deal with the influence function, for instance SIMPATH [11] and LDAG [5]. Moreover, recent studies show good results using a technique known as *Reverse Reachable* sets [2], which has provided algorithms as efficient as the heuristics, but with the plus of having approximation factor guarantee, for example TIM+ [18] and IMM [17]. We can think of algorithms for the influence maximization problem as having two phases, (i) the influence function estimation and (ii) the seed selection, where Monte Carlo simulations, Score Estimation and Reverse Reachable techniques address the first one. In this work, we use a preselection strategy in order to improve the performance of the algorithms that deal with the second phase.

While most of the studies focus on proposing algorithm to find the set of early adopters or to estimate the influence function, for instance [3, 4, 9–11, 14, 17, 18], we try to take advantage of knowing the topology of most of the large scale social networks, which follows a power law degree distribution. In such networks there are few vertices with a large number of neighbors, called *hubs*, and many with low degree [6, 8, 15]. Liu et al. [15] shows that only a few out-neighbors of the hubs have considerable influence, while many of these neighbors contribute little to the marginal gain. These findings suggest that there is a relation between the degree distribution and the reach of an information that spreads along the network. We explore these findings to recognize and rule out the less probable influencer nodes.

In summary, the contribution of the paper is twofold. An efficient heuristic to select the more promising vertices and, as an application of such strategy, we present an algorithm to select the early adopters in power law graphs. The main contribution is the PRESELECTOR algorithm which chooses a subset of the vertices based on its degree, where the objective is to decrease the number of evaluated vertices by the greedy algorithm. Such strategy allied with the CELF optimization lead us to the second contribution, the PREVALENTSEED, that makes less calls to the σ function. Experimentally, this approach reduces up to 57% the CELF's run time.

The rest of the paper is organized as follows. Initially, we introduce the algorithm to select the most promising vertices to become early adopters. Then we show some theoretical analysis that have been carried out on run time and quality results. Next, we present the algorithm that chooses the early adopters using the preselection combined to a lazy forward update scheme. Lastly, in the experiments section, some observations are described about the empirical results achieved on real world power law graphs.

2 OPTIMIZATION BY PRESELECTION

Instead of computing the marginal gain over the whole set of vertices at each iteration to select those of greatest marginal gain, we only travel a subset of nodes. This subset will be called *set of candidates*, and it is selected by a heuristic (Algorithm 2) in advance. In order to reduce the number of calls to σ, we discard the nodes that may have small marginal gain before processing them in fact. Thus, we compute the marginal gain only for the more promising nodes, so we avoid multiple calls to the σ's estimation by choosing correctly such nodes. Our strategy to select the candidates is fundamentally based on the following criteria. We assume that a vertex will not have high marginal gain if their out-neighbors already can be influenced by a node of higher degree. So the search chooses nodes that can cover the greatest number of non-covered nodes. A vertex is covered when it has at least one in-neighbor that already was chosen as candidate in the iterative process. The procedure stops when there are no more uncovered vertices.

2.1 Set of Candidates

The set of candidates, selected by Algorithm 2, is defined as $C \subseteq V$. Initially, $C = \emptyset$, and nodes are added iteratively during the execution of the algorithm. For every $v \in V$, the set of out-neighbors of v is denoted by $Out(v) = \{w \in V \text{ such that } (v, w) \in E\}$. Similarly, the out-neighborhood of the set C is denoted as $Out(C) = \{(u, v) \in E \text{ such that } u \in C \text{ and } v \notin C\}$. To simplify the pseudo-code, we denote by $D = C \cup Out(C)$ the set of vertices covered by C.

Algorithm 2: PRESELECTOR

Input: $G = (V, E)$
Output: Set C of candidates

```
1 begin
2     Sort the vertices v_1, v_2, ..., v_|V| in decreasing order by
        out-degree
3     C ← ∅; D ← ∅
4     for i ← 1 to |V| do
5         if Out(v_i) ⊈ D and Out(v_i) ≠ ∅ then
6             C ← C ∪ {v_i}
7             D ← D ∪ {v_i} ∪ Out(v_i)
```

In Algorithm 2, each node v_i is selected as candidate when it has at least one out-neighbor that is still uncovered. Theorem 2.1 determines an upper bound for the PRESELECTOR's run time.

Theorem 2.1. *Let G be a directed graph with n vertices and m edges. Algorithm 2 ends in $O(n+m)$ steps.*

Proof. At the first line, if we use an efficient algorithm like the *counting sort* to sort the vertices, this task will be performed in linear time on the number of vertices. We can use this algorithm because the out-degrees of the nodes are values from 1 to $n-1$. After, in the loop of the lines 4-7, the trickiest operation is the condition that depends on whether the set D contains $Out(v_i)$, for all $v_i \in V$. We can verify such condition in constant time using a *hash table* to store the elements of D. Thus, let $\delta^+(v_i)$ be the out-degree of v_i. No more than $\delta^+(v_i) \cdot O(1)$ steps are needed to check each v_i. Thus, directly by the "*Handshake*" lemma, the loop demands $\sum_{v_i \in V} \delta^+(v_i) = O(m)$ comparisons. Finally, the total time for preselection is $O(n) + O(m) = O(n+m)$, where $O(n)$ is the time to sort the list of vertices. □

2.2 Analysis of the Preselection Process

The preselection process is biased in favor of nodes of high degree, since it consists in discarding nodes in which all its out-neighbors are covered by higher-degree vertices. So we want to show that when a vertex v does not belong to C after preselection, v tends to have a low marginal gain in comparison to the selected nodes. By this way, we can avoid unnecessary computation, for the marginal gain of v, during the greedy search for the k nodes of the highest marginal gain. For this purpose we use three results to argue about. First, for each vertex v in which all its out-neighbors are already candidates, we can activate v plus $Out(v)$ in order to improve the spread of active nodes. But, in number of activated nodes, activate v has the same effect of activating only $Out(v)$. For this reason, it is not needed to activate v since only its neighbors are sufficient. The Lemma 2.2 provides a demonstration of this statement. Second, if all the out-neighbors of a vertex v are covered by the set of candidates, putting v together with C as active nodes can increase the probabilities of such neighbors being activated, but such increase is low and limited. Thereby, as shown in Lemma 2.3, v can be left aside. The last result obtained from this analysis is the Theorem 2.4, which says that for each $v \notin C$, the set C has the possibility of activate all the nodes that v would activate.

To analyze the quality of the Preselector's output, we have to assume some simplifications. First, we consider the IC model with activation probabilities p equals on each edge. Second, to make some calculus, we use a more simple influence function called *direct influence* instead of σ itself. Not making these simplifications implies in compute the exact value of σ, which is not the goal of this analysis, since to get this value is a #P-hard problem. The Definitions 1 and 2 describe the concept of direct influence.

Definition 1 (Direct influence of a vertex). *Let v be an active vertex. We call direct influence of v the number $inf(v)$ of vertices in $Out(v)$ activated by v.*

Definition 2 (Direct influence of a set). *Let A be a set of active nodes. We denote $inf(A)$ the direct influence of A, the number of nodes in $Out(A)$ activated by vertices in A.*

In this way, note that for all $v \in V$, each $w \in Out(v)$ becomes active with probability p so $\inf(v)$ is a random variable. Thus, at every execution of the activation process, $\inf(v)$ can assume a different value between zero and $|Out(v)|$. The same happens with $\inf(A)$. Therefore, we can use expectation on the following results.

In the Algorithm 2, notice that a vertex v will not become a candidate if all its out-neighbors are covered by the set C, at line 5. This can happen in two different ways: (i) either $Out(v) \subseteq C$, or (ii) the out-neighbors of v are covered but not all $w \in Out(v)$ belongs to C. Lemmas 2.2 and 2.3 address these cases respectively.

Lemma 2.2. *Let v_i be a vertex at the i-th iteration of the Algorithm 2. If $Out(v_i) \subseteq C$, then the additional influence that v_i could yield for the set C is either null or negative, that is,* $\mathrm{E}[inf(C \cup \{v_i\})] - \mathrm{E}[inf(C)] \leq 0$.

Proof. To validate the inequality of the lemma, we need to know the value of $\inf(C)$. Knowing that the set C has at least one edge to each $w \in Out(C)$, the reasoning is as follows. To find the number of vertices that can be directly activated by C, consider a random variable Y_w which is 1 if w was activated by a vertex in C, and 0 otherwise. Thus,

$$\inf(C) = \sum_{w \in Out(C)} Y_w.$$

By the linearity of expectation and the definition of Y_w as binary variable, the expected value of $\inf(C)$ is

$$\mathrm{E}[\inf(C)] = \sum_{w \in Out(C)} \mathrm{E}[Y_w] = \sum_{w \in Out(C)} \Pr(Y_w = 1). \quad (1)$$

Suppose now that we have added v_i to C in order to increase $\inf(C)$. To determine the effect of this change in the activation probabilities, we need to consider whether v_i was in the neighborhood of C before becoming a candidate. In the negative case, that is, if $v_i \notin Out(C)$, it is simple to visualize that including v_i in C does not increase the value of $\inf(C)$, once v_i has no neighbor outside of C, no edge will be added to the sum of the probabilities on the Eq. 1, that is, $\mathrm{E}[\inf(C \cup \{v_i\})] = \mathrm{E}[\inf(C)]$. In an activation process in which the set C is active, all the out-neighbors of v_i would already be activated, and then v_i would not activate another vertex. However, if $v_i \in Out(C)$, things can be different because adding v_i to C reduces one element of $Out(C)$, then the value of $\inf(C)$ cannot be larger than $|Out(C)| - 1$. Hence, at least one edge is removed from the sum of the Eq. 1. Then we have

$$\mathrm{E}[\inf(C \cup \{v_i\})] = \sum_{w \in Out(C) \setminus \{v_i\}} \Pr(Y_w = 1) < \mathrm{E}[\inf(C)].$$

Given these two possibilities, we finally have that $\mathrm{E}[\inf(C \cup \{v_i\})] \leq \mathrm{E}[\inf(C)]$. □

We saw that the vertices in which all its out-neighbors are candidates do not improve the direct influence of C. Now we can think about the vertices that have out-neighbors covered, that is, it share all the out-neighbors with the set of candidates, but such neighbors can be both in C and $Out(C)$. In this case, the Lemma 2.3 gives us an upper bound to the additional direct influence that this type of vertex can provide to the set of candidates.

Lemma 2.3. *Let v_i be a vertex at the i-th iteration of the Algorithm 2. Denoting as $\mathcal{G}(C, v_i)$ the additional influence provided by adding v_i in C. If $Out(v_i) \subseteq D$, but $Out(v_i)$ is not fully in C, then $\mathcal{G}(C, v_i) \leq \frac{1}{4}|Out(v_i) \setminus C|$.*

Proof. Whereas the set $Out(v_i)$ is not fully contained in C but belongs to D, v_i should stay in $V \setminus C$. However, if we add v_i to C in order to increase inf(C), as in Lemma 2.2, we have to consider two possible situations: (i) $v_i \in Out(C)$, and (ii) $v_i \notin Out(C)$. Unlike the Lemma 2.2, now in both cases (i) and (ii) the probabilities will change and increase the direct influence of C. This happens because the number of edges incident to $Out(C)$ increases, and so the probability of such vertices becoming active increases. We want to find an upper bound for this probability growth.

Given a vertex $w \in Out(v_i) \setminus C$, this vertex can be directly activated by the vertices in C, and we can obtain the probability of C activate w directly as follows. Let A be the event in which w is activated by C. Remembering that p is the activation probability in the independent cascade model, we have $\Pr(A) \geq p$, that is, the set C has at least one edge to w. Including v_i in C, w could be activated by C and by v_i. Thus, let B be the event in which w is activated by v_i, then $\Pr(B) = p$ and the probability of the set $C \cup \{v_i\}$ activate w can be obtained with the equation

$$\Pr(A \cup B) = \Pr(A) + \Pr(B) - \Pr(A) \cdot \Pr(B) = \Pr(A) + p - \Pr(A) \cdot p.$$

Here we apply the principle of inclusion and exclusion in the sum of the activation probability. The first equality occurs due to the independence between the events A and B.

The increase supplied by v_i on the probability of C activate w is the difference between $\Pr(A \cup B)$ and $\Pr(A)$. Note that $\Pr(A)$ is equivalent to $\Pr(Y_w = 1)$ in the Eq. 1. Thus, let $\mathrm{E}[\inf_w(C)]$ be the expected direct influence of C on w. By the Eq. 1, we have $\mathrm{E}[\inf_w(C)] = \Pr(Y_w = 1) = \Pr(A)$, similarly, $\Pr(A \cup B) = \mathrm{E}[\inf_w(C \cup \{v_i\})]$. Let $\mathcal{G}_w(C, v_i)$ be the additional influence provided by v_i on the probability of C activate w. The value of $\mathcal{G}_w(C, v_i)$ is given by the following equation

$$\begin{aligned}
\mathcal{G}_w(C, v_i) &= \mathrm{E}[\inf_w(C \cup \{v_i\})] - \mathrm{E}[\inf_w(C)] \\
&= \Pr(A \cup B) - \Pr(A) \\
&= \Pr(A) + p - \Pr(A) \cdot p - \Pr(A) \\
&= p - \Pr(A) \cdot p \\
&= p(1 - \Pr(A)) \\
&\leq p(1 - p),
\end{aligned}$$

where the inequality holds because $\Pr(A) \geq p$. This holds for any $w \in Out(v_i) \setminus C$. Consequently, we need to apply the same difference for all vertices in $Out(v_i) \setminus C$. The increase of direct influence led us to a quadratic function that represents a parabola upside-down, such that the max value is $\frac{1}{4}$, when $p = \frac{1}{2}$. In consequence,

$$\begin{aligned}
\mathcal{G}(C, v_i) &= \prod_{w \in Out(v_i) \setminus C} \mathcal{G}_w(C, v_i) \\
&\leq \prod_{w \in Out(v_i) \setminus C} p(1 - p) \\
&\leq \prod_{w \in Out(v_i) \setminus C} \frac{1}{4} \\
&= \frac{1}{4} \cdot |Out(v_i) \setminus C|.
\end{aligned}$$

As the final point, the value of $\mathcal{G}_w(C, v_i)$ is mutually independent for each $w \in Out(v_i) \setminus C$, then we can take the product over all results. □

As shown above, selecting as candidate a vertex v_i such that $Out(v_i) \subseteq D$, provides a negligible additional influence. Additionally, the Theorem 2.4 brings up that at the end of preselection any vertex ruled out has less influence on its neighbors than the set C.

Theorem 2.4. *Let $inf_{Out(v)}(C)$ be the direct influence of C on $Out(v)$, where $v \in V$. At the end of the Algorithm 2, we have $\mathrm{E}[inf(v)] \leq \mathrm{E}[inf_{Out(v)}(C)]$, for all $v \in V \setminus C$.*

Proof. By the Algorithm 2, if $v \in V \setminus C$ then all $w \in Out(v)$ should be in D. We want to compare the v's influence with C's influence on $Out(v)$. To this end, we need to consider just the vertices in $Out(v) \setminus C$. Thus, when $w \in Out(C)$, besides receiving an edge of v, w also receives at least one edge from C. Let $X_{v,w}$ be a binary random variable, which is 1 if v activates w and 0 otherwise. We can get the value of inf(v) by the equation

$$\inf(v) = \sum_{w \in Out(v) \setminus C} X_{v,w}.$$

Using the linearity of expectation, we have

$$\begin{aligned}
\mathrm{E}[\inf(v)] &= \sum_{w \in Out(v) \setminus C} \mathrm{E}[X_{v,w}] \\
&= \sum_{w \in Out(v) \setminus C} \Pr(X_{v,w} = 1) \\
&= \sum_{w \in Out(v) \setminus C} p. \qquad (2)
\end{aligned}$$

Since some vertices in C have edges to $Out(v)$, each $w \in Out(v)$ have at least one edge from C. Let $B_w = \{u \in C$ such that $(u, w) \in E\}$ be the set of vertices in C with edges to w, for all $w \in Out(v) \setminus C$. Note that, $|B_w| \geq 1$, otherwise w would not be covered by C. Now consider the events $E_1, E_2, ..., E_{|B_w|}$ such that E_i is the event in which w is activated by $u_i \in B_w$. Once we have defined that all edges have activation probability p, then $\Pr(E_1) = \Pr(E_2) = ... = \Pr(E_{|B_w|}) = p$. As a result, for each $w \in Out(v) \setminus C$, we have

$$\begin{aligned}
\Pr(X_{v,w} = 1) = p &\leq \Pr(E_1 \cup E_2 \cup ... \cup E_{|B_w|}) \qquad (3) \\
&= \Pr(B_w \text{ activate } w).
\end{aligned}$$

To know the number of nodes in $Out(v)$ activated by C, consider a random variable Y_w which is 1 if w is activated by a vertex in C and 0 otherwise. Thus,

$$\inf_{Out(v)}(C) = \sum_{w \in Out(v) \setminus C} Y_w.$$

Again, by the linearity of expectation,

$$\begin{aligned}
\mathrm{E}[\inf_{Out(v)}(C)] &= \sum_{w \in Out(v) \setminus C} \mathrm{E}[Y_w] \\
&= \sum_{w \in Out(v) \setminus C} \Pr(Y_w = 1) \\
&= \sum_{w \in Out(v) \setminus C} \Pr(B_w \text{ activate } w) \\
&\geq \sum_{w \in Out(v) \setminus C} p \\
&= \mathrm{E}[\inf(v)],
\end{aligned}$$

where the inequality follows from Eq. 3 and the last equality came from Eq. 2. □

Such results show that the vertices in C have more probability of becoming early adopters than most of the vertices in $V \setminus C$. Due to the criteria to choose C, all the vertices that have no out-neighbors are in $V \setminus C$ while many of the higher degree vertices belong to C. Hence, we suppose that the marginal gain of the vertices in $V \setminus C$ is always low and would be discarded in any way. In view of power law graphs, where most of the vertices have low degree and a small number have high degree, there are two general aspects of the excluded vertices. First, a large number of nodes has degree equal to one, many of which has no out-neighbors. Furthermore, some vertices which degrees are greater than one also have no out-neighbors. Consequently, these nodes would not activate other vertices. Second, for all $v \in V \setminus C$ in which the out degree is greater than one, v is subject to Lemmas 2.2, 2.3 and Theorem 2.4, that is, the nodes in C are enough to achieve the $Out(v)$.

It is worthwhile noticing that, based on the number of nodes without out edges, it is possible to quantify the number of vertices that cannot activate anyone. But it requires a very thorough analysis on random power law graphs, and this is not the scope of this work. Given this, we suppose that we can select the k early adopters of set S within the set C without losing quality of spread, then we do not need to consider every vertex of the graph using the greedy algorithm. Therefore, although the analysis was simplified, there are strong indications that the results would be positive in more complex models, with distinct propagation probabilities.

3 THE PREVALENTSEED ALGORITHM

We now present an algorithm that chooses the early adopters in power law graphs, called PrevalentSeed. We combine the PreSelector with the Celf's "*Lazy Forward*" update scheme. The idea is to show how the preselection can be used in a seed set selection algorithm. Algorithm 3 shows the pseudo code. Initially, we divide the vertex set into two disjoint subsets, C and $V \setminus C$, such that C is the set of candidates (line 3). Here, we chose to use the Celf optimization as a sub routine instead of the greedy algorithm of Kempe *et al.* [12], since it is faster. Therefore, the code snippet between lines 4-15 is a modification of Celf, in which the difference is the loop at lines 4-7, where we inserted in the list Q only the vertices of set C. From this moment on, the marginal gain of the vertices in $V \setminus C$ is not estimated anymore. Next, the marginal gain of each $v \in C$ is estimated and v is added to Q in a non increasing order of marginal gain. The search follows the Celf's idea, at lines 8-15, to make a greedy search and select the k vertices of higher marginal gain from the vertices belonging to the set C.

As well as in the Celf algorithm, the element of Q corresponding to v stores a table of the form $\langle \delta_v, v.it \rangle$, where $\delta_v = \sigma(S \cup \{v\}) - \sigma(S)$ is the marginal gain of v compared to S, and $v.it$ marks at which iteration the value of δ_v was last updated. In each of the k iterations of 'while' loop, v is removed from the queue and checked if the marginal gain already was computed at the current iteration, using the *it* attribute. If yes, v is the vertex of the greatest marginal gain at the current iteration, so it will be selected as a seed (lines 10-11). Otherwise, the lines 12-15 recompute the v's marginal gain and insert it again in Q such that the order is maintained.

Algorithm 3: PrevalentSeed

```
   Input: G, k, σ
   Output: Seed set S
1  begin
2      S ← ∅, Q ← ∅
3      C ← PreSelector(G)
4      foreach u ∈ C do
5          δ_u ← σ({u})
6          u.it ← 0
7          Add u to Q in a non increasing order by δ_u
8      while |S| ≤ k do
9          Dequeue u from Q
10         if u.it = |S| then
11             S ← S ∪ {u}
12         else
13             δ_u ← σ(S ∪ {u}) − σ(S)
14             u.it ← |S|
15             Enqueue u in Q and sort
```

Theorem 3.1 determines the running time of the PrevalentSeed and shows that it is asymptotically equal to Celf, even making the preselection.

Theorem 3.1. *Let G be a directed graph with n vertices and m edges. Algorithm 3 executes in $O(knrm)$ time.*

Proof. The PrevalentSeed's running time is given as follows. (i) By Theorem 2.1, the call to PreSelector at line 3 uses $O(n+m)$ steps to find the set C. (ii) In the loop of lines 4-7, $O(|C|rm)$ operations are made. This loop computes the value of $\sigma(v)$ for all $v \in C$. The $\sigma(v)$ is estimated with $r = 10.000$ simulations of spread process (Monte Carlo method). Every call to $\sigma(v)$ spends $O(rm)$ time. Moreover, each insertion in Q has time $O(1)$. Thus, this loop needs $O(|C|rm)$ operations. (iii) To choose k nodes $O(knrm)$ steps are needed at the 'while' loop. This loop is an adaptation of Celf optimization in with the difference that the set of vertices V is replaced by the set C of candidates. As explained by [14], this algorithm has time $O(knrm)$. Since $|C| \leq n$, then we have the same bound. Therefore, the total running time is the sum of items (i), (ii) and (iii). $O(n+m) + O(|C|rm) + O(knrm) = O(knrm)$. □

4 EXPERIMENTS

We conducted the experiments in two types of datasets, real social networks and synthetically generated graphs. The comparison was made between PrevalentSeed and the Celf algorithm by taking into account two metrics: size of set of vertices achieved by the spread of influence (ie, quality of seed set) and running time. In the experiments, the proposed algorithm got significant gains in performance compared to Celf besides preserving the expected spread in a competitive level. Although the Celf++ is faster than Celf in the experiments reported by Goyal et al. [10] we chose Celf as baseline because in some empirical evaluations the Celf remains more robust on different types of graph.

The algorithm was implemented in Java using the JGraphT library (jgrapht.org) and all experiments were performed on a machine with GNU/Linux (Linux Mint 17) which hardware configurations were: (i) Processor: Intel(R) Core(TM) i5-3210M CPU, 2.50GHz, x86_64 architecture, and 4 CPU's. (ii) Cache memory: 128KiB L1 cache; 512KiB L2 cache; 3MiB L3 cache. (iii) RAM: 6GiB SODIMM DDR3 Synchronous 1600 MHz (0,6 ns).

4.1 Real World Power Law Graphs

We seeked networks that exhibited structural features of large scale social networks and power law degree distribution. Six graphs were used to exemplify the results. Table 1 summarizes some data about these graphs.

Table 1: Statistics information of the social networks. The β values are from Liu *et al.* [15] and Tang *et al.* [16] results.

Social Network	Vertices	Edges	Exponent (β)
NetHEPT	15,233	32,213	2.651
NetPHY	37,154	180,826	2.843
Enron	36,692	367,662	2.357
Epinions	75,879	508,837	2.383
Amazon	262,111	1,234,877	2.432
DBLP	654,628	1,990,259	3.361

All the tests were carried out in the independent cascade model. In order to evaluate graphs relatively large, we splitted the experiments into two categories. The smallest graphs and the largest ones. Due to the long time required to make Monte Carlo simulations, we set the propagation probabilities to $p = 0.025$ on smaller graphs and $p = 0.0025$ on larger graphs. It was necessary because higher probabilities would make the experiments infeasible, as far as the running time is concerned. We simulated the propagation process by 10,000 times for each selected set, as in the literature [10, 12, 14].

4.1.1 Results and Discussion. The quality of the early adopters selected by the algorithms was evaluated based on the number of activated vertices. The higher the spread, the better the quality. In the graphics of Figure 1 the algorithms have similar results on the influence propagation, that is, the number of activated vertices are almost the same. Note that despite the difference of probabilities, the results are similar and both algorithms have produced good seed sets. Concerning the running time, Table 2 summarizes the effectiveness of the PrevalentSeed compared to Celf's time when $k = 50$. For simplicity purposes, we kept only two decimal places of precision. The last column shows how much PrevalentSeed was faster than Celf. In this case, we achieved a reduction up to 57%, but unfortunately we also got negative results. It is important to note that the result of the two last graphs on Table 2 was negative mainly due to a very important fact, which is the size of set C.

In the graphs which the performance of PrevalentSeed was worse than Celf (Enron and Epinions), the density is higher than in the other graphs. This feature implies that a smaller number of vertices are needed to cover all the graph, that is, the size of C decreases when the number of edges increases. With few candidates, the Celf's priority queue needs to be reorganized more times. To reorganize the priority queue, it is necessary to estimate the marginal gain again, making new calls to the σ function. Since, in this step, the set S is not empty, such computation should be more time consuming because the spread tends to be larger when S grows Thus, each new call to σ can negatively affect the algorithm's run time. That is why PrevalentSeed can be worse than Celf.

Table 2: Difference between the running time of Celf and PrevalentSeed, for $k = 50$.

Network	PrevalentSeed	Celf	Gain
NetHEP	220.84	315.61	30.02%
NetPHY	3,195.05	5,366.88	40.46%
Amazon	2,438.71	5,679.99	57,06%
DBLP	5,541.06	10,876.31	49,06%
Enron	212,964.45	192,744.20	-10.49%
Epinions	123,392.18	112,237.96	-9,93%

Table 3: Calls to the σ function in all tested graphs. Columns 4 and 5 shows the total of reorganizations of Q needed to PrevalentSeed and Celf, respectively.

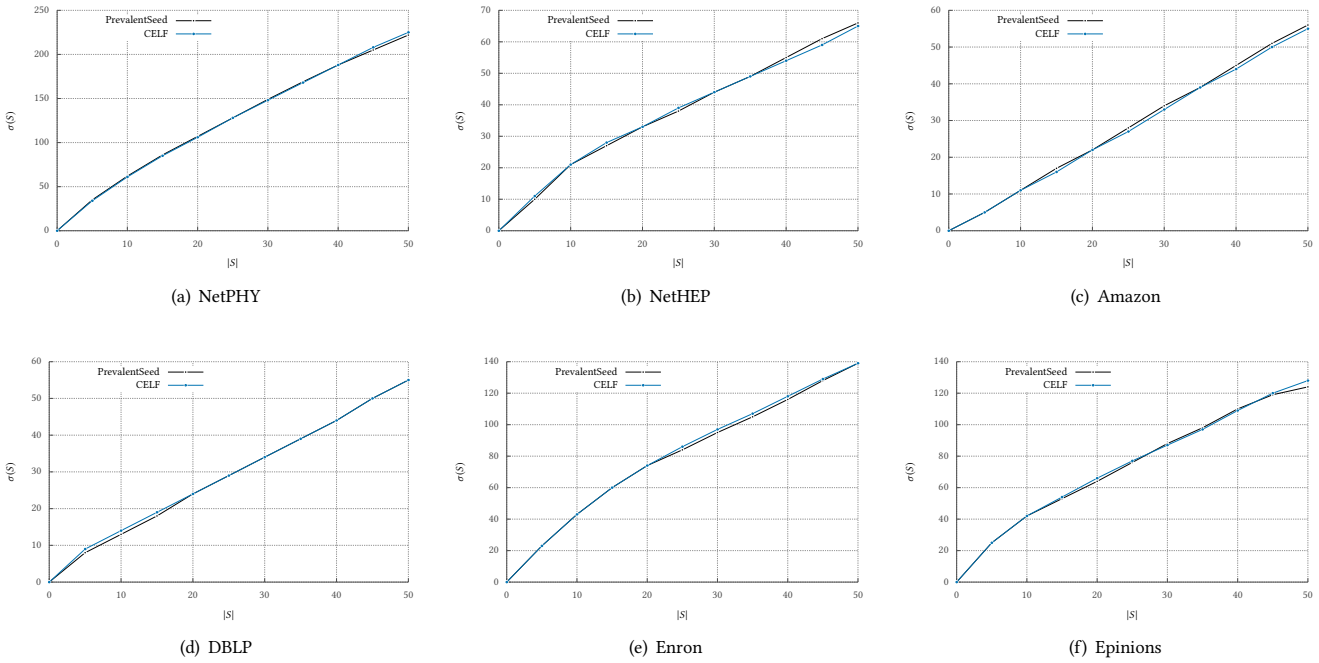
Graph	Calls to σ		Reordering of Q	
	PrevalentSeed	Celf	PrevalentSeed	Celf
NetHEP	4389	15370	125	137
NetPHY	8713	37495	213	341
Amazon	107058	262205	93	94
DBLP	117537	654726	96	98
Enron	3617	36861	194	169
Epinions	13367	76055	195	181

Table 3 presents how many reorganizations were needed to each graph in both algorithms. It is easy to note that PrevalentSeed behaves badly only in the cases where the number of reordering was greater than in Celf. Even with the notable difference between the total number of calls of the two algorithms, what really impacts the running time are the calls required to reorder the queue. Note that the total of calls is proportional to the number of vertices placed into the queue Q. As the goal of the preselection optimization is to reduce this number of vertex, the PrevalentSeed makes less calls to σ than Celf at the proportionality of $|C|$.

Fortunately, even with a worst time in some cases, the quality of seed set remained competitive (see on Figure 1). Since our preselection heuristic aims to solve the problem in power law graphs, we believe that this is not a prohibitive trouble. Based on these findings, we recommend that is enough to pay attention to scale coefficient β. The experiments show that the gain in time reduction is better when $\beta \geq 2.4$. Thus, when the density of the graph is higher, it is more appropriate to use only Celf optimization without Preselector.

4.2 Synthetic Graphs

In order to artificially represent realistic social networks, the random graph model used in this study is based on the Aiello et al. [1]



(a) NetPHY (b) NetHEP (c) Amazon

(d) DBLP (e) Enron (f) Epinions

Figure 1: The simulations in NetHEP, NetPHY and Amazon have propagation probability $p = 0.025$, and the simulations in DBLP, Enron and Epinions have $p = 0.0025$.

model, denoted as $P(\alpha, \beta)$. The generated networks combine the topology introduced by $P(\alpha, \beta)$ model together with the *Generalized Random Graph* (GRG) model [19]. The purpose of this methodology is to generate power law random graphs with an adjustable scale exponent β. Roughly speaking, in the GRG model, a graph starts with a set of n vertices with no edges between them. Each vertex has a weight which determines the probability of having edges. The algorithm adds edges between pairs of vertices according to its weights. Hence, the graph topology depends on the chosen weights, and it can be handled such that the resulting graph has an expected degree according to a desired distribution. So, the $P(\alpha, \beta)$ model provides a well-defined sequence of weights, used as the input of the GRG model, that follows a power law distribution.

In our experiments, we generated 60 synthetic graphs. The network generation parameters are size and density, in which we vary the number of vertices such that $n = \{2, 4, 8, 16, 32, 64\}$ and the scale exponent was fixed in $\beta = 2.5$. For each n, we perform the experiments on 10 networks, and report the average results to both expected propagation and running time. In all the experiments, we preprocessed the graphs by eliminating the isolated nodes and small components in order to only use the connected graph of the giant component. The propagation probabilities on the edges are draw uniformly at random from the interval $[0, \frac{1}{4}]$. Such settings allow us to perform experiments in feasible time on the IC model with random probabilities.

4.2.1 Results and Discussion. We plot the performance of PrevalentSeed and Celf in both metrics, expected propagation and running time. Figure 2 presents the average expected propagation of the networks with 64 thousand vertices, where the spread of PrevalentSeed match almost perfectly with the Celf's spread. This confirms that our heuristic is able to reach the same quality. This matches the intuition from the end of the analysis of the preselection process, which says that without losing the quality of spread, we can select the early adopters within the set C. For the running time, we see that our algorithm does better than Celf. Figure 3(a) shows the amount of time required to find a seed set of 50 vertices on the random graphs. The gain in running time for each of the sizes 2k, 4k, 8k, 16k, 32k and 64k of the graphs are 17.2%, 22.42%, 36.4%, 26.61%, 24.4% and 29.67%, respectively.

Figure 3(b) presents the average number of calls to σ function. In these settings, the difference between the algorithms is directly related to the size of the set of candidates. Thus, the preselection decreases the quantity of influence estimation along the greedy search. Also, Figure 3(b) shows that even with a noteworthy decrease in the number of calls to σ function, the running time (Figure 3(a)) does not decreases proportionally. Again, the comparison between the running time and the number of calls to σ reinforce the idea that the more expensive calls to σ are those performed to reorder the priority queue, as already reported in the real-word network evaluations in Section 4.1.

We are aware of new algorithms and heuristics that outperforms our baseline and the PrevalentSeed into this field, for instance IMM [17] and TIM+ [18], and by the time the paper is published, some details of our comparison method will be outdated. Nonetheless, our main contribution remains valid because the focus of our

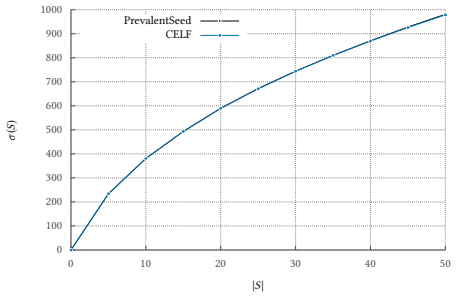
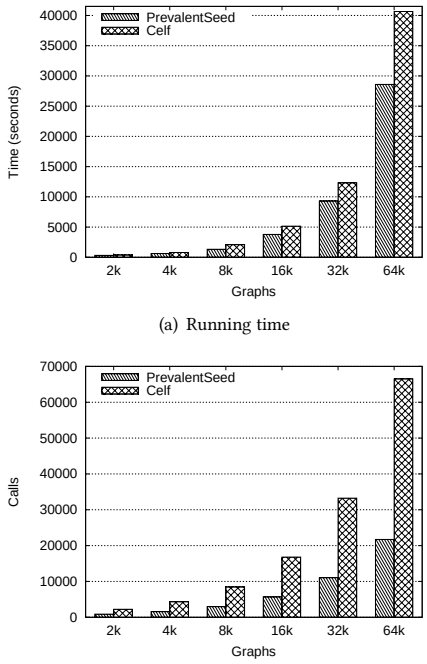


Figure 2: Simulations in synthetic graphs with 64 thousand vertices and propagation probability $p \in [0, \frac{1}{4}]$. We plot only one size of graph due to similar results in all tested sizes.



(a) Running time

(b) Calls to σ function

Figure 3: Average running time (a) and number of calls to the σ function (b) of both algorithms to find 50 seeds on synthetic graphs.

work is presenting a preselection methodology specialized in power law graphs, that can be applied or combined with any greedy algorithm. The purpose, therefore, is not to compete with new algorithms in selecting the seed set, but to offer a way to improve the performance in power law graphs. The PrevalentSeed algorithm is an example of how the preselection can be applied in a given seed selection algorithm. For this reason, we kept the original Celf as baseline, since our goal is to compare the results with and without the preselection.

5 CONCLUSIONS

Some interesting features of the preselection are that it explores the relationship between influence propagation and degree distribution of social networks to highlight the most promising vertices, preventing unnecessary processing by cutting out some elements of the search. Experimentally, the PrevalentSeed is reasonably faster than Celf in most of the evaluated graphs. This happens mainly due to the reduction of the number of estimation of the influence function. Moreover, the set of activated nodes chosen by PrevalentSeed are very competitive with those found by Celf in terms of quality. In addition, the theoretical analysis concerning the reach of spread produce results that goes according to the empirical analysis.

REFERENCES

[1] William Aiello, Fan Chung, and Linyuan Lu. 2001. A random graph model for power law graphs. *Experimental Mathematics* 10, 1 (2001), 53–66.

[2] Akhil Arora, Sainyam Galhotra, and Sayan Ranu. 2017. Debunking the myths of influence maximization: An in-depth benchmarking study. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 651–666.

[3] Wei Chen, Chi Wang, and Yajun Wang. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1029–1038.

[4] Wei Chen, Yajun Wang, and Siyu Yang. 2009. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 199–208.

[5] Wei Chen, Yifei Yuan, and Li Zhang. 2010. Scalable influence maximization in social networks under the linear threshold model. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 88–97.

[6] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703.

[7] Pedro Domingos and Matt Richardson. 2001. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 57–66.

[8] David Easley and Jon Kleinberg. 2010. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press.

[9] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. 2011. A data-based approach to social influence maximization. *Proceedings of the VLDB Endowment* 5, 1 (2011), 73–84.

[10] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. 2011. Celf++: optimizing the greedy algorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*. ACM, 47–48.

[11] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. 2011. Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE, 211–220.

[12] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 137–146.

[13] Jon Kleinberg. 2007. Cascading behavior in networks: Algorithmic and economic issues. *Algorithmic game theory* 24 (2007), 613–632.

[14] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 420–429.

[15] Xiaodong Liu, Shanshan Li, Xiangke Liao, Shaoliang Peng, Lei Wang, and Zhiyin Kong. 2014. Know by a handful the whole sack: efficient sampling for top-k influential user identification in large graphs. *World Wide Web* 17, 4 (2014), 627.

[16] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 990–998.

[17] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1539–1554.

[18] Youze Tang, Xiaokui Xiao, and Yanchen Shi. 2014. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 75–86.

[19] Remco Van Der Hofstad. 2016. Random graphs and complex networks. (2016).