Preprocessing Rules for Target Set Selection in Complex Networks

Renato Silva de Melo, André Luís Vignatti

[1]Department of Informatics – Federal University of Paraná (UFPR)
Curitiba – PR – Brazil

{rsmelo,vignatti}@inf.ufpr.br

Abstract. *In the Target Set Selection (TSS) problem, we want to find the minimum set of individuals in a network to spread information across the entire network. This problem is NP-hard, so find good strategies to deal with it, even for a particular case, is something of interest. We introduce preprocessing rules that allow reducing the size of the input without losing the optimality of the solution when the input graph is a complex network. Such type of network has a set of topological properties that commonly occurs in graphs that model real systems. We present computational experiments with real-world complex networks and synthetic power law graphs. Our strategies do particularly well on graphs with power law degree distribution, such as several real-world complex networks. Such rules provide a notable reduction in the size of the problem and, consequently, gains in scalability.*

1. Introduction

The dynamics of information diffusion in complex networks has attracted interest, not only in social sciences but also in mathematics and computer science. Applications in viral marketing have motivated researches on an algorithmic perspective. An important problem investigated in this context consists in finding a small set of individuals of a social network to start a cascade of adoption throughout the entire system. Furthermore, social networks, when modeled with graph theory, usually present a particular degree distribution of the individual's relationships. Such distribution is referred to as power law (or scale-free). Roughly speaking, this means that a small number of individuals have a large number of connections, while the majority have a very small amount of connections.

The problem investigated in this paper has roots in viral marketing applications [Domingos and Richardson 2001], where the objective is to market a new product and induces a "word-of-mouth" effect such that it would be gradually adopted by a large number of individuals in the network. It was first approached as a discrete optimization problem by [Kempe et al. 2003] with a maximization objective function, in a probabilistic environment, called Influence Maximization Problem. They showed that it is NP-hard to solve and also NP-hard to approximate within a factor $n^{1-\epsilon}$, for any constant $\epsilon > 0$. Subsequent to [Kempe et al. 2003], several efficient heuristic solutions have emerged for this problem taking into account different diffusion models, for example [Goyal et al. 2011a, Goyal et al. 2011b, Tang et al. 2014, Chen et al. 2010a, Chen et al. 2010b]. The TSS problem was studied by [Chen 2009] in a deterministic setting with a minimization objective function. He also showed inapproximability results, even with additional assumptions such as the probabilistic thresholds. When the input graph is restricted to be a tree, [Chen 2009] presents an exact polynomial time algorithm. The work of [Ackerman et al. 2010] is among the first to propose an Integer Linear Programming formulation for TSS. Using this formulation they derived lower and upper bounds on the size of the minimum target set for fixed deterministic threshold functions named majority threshold and strict majority threshold.

There is a reasonable amount of studies on the spreading of information considering the power law degree distribution of social networks. [Zhang et al. 2012] concentrate on working with the POSITIVE INFLUENCE DOMINATING SET (PIDS) problem on power law graphs and prove that a greedy algorithm presented by [Wang et al. 2011] admits a constant approximation ratio in such networks. They show that the power law degree distribution can improve the performance of greedy algorithms for a class of problems known as *submodular cover* problems. Further, they also proved that the PIDS problem belongs to the class of submodular cover problems and has a constant approximation factor.

Regarding the INFLUENCE MAXIMIZATION problem, [Liu et al. 2014] presented a Monte Carlo based method for estimating the spread of influence in a social network that follows a power law degree distribution. The estimation method proposed by [Liu et al. 2014] aims to use a supervised sampling to predict the number of vertices needed to be sampled according to the power law exponent β of a given social network. This strategy avoids computing the exact value of influence function, and it can efficiently estimate the influence spreading at a small cost of precision. From another perspective, [Melo and Vignatti 2018] proposed a preprocessing strategy for the INFLUENCE MAXIMIZATION problem by employing a simple greedy heuristic. The algorithm exploits the degree distribution to select in advance the most promising vertices to be in the solution of the influence maximization problem. Thus, it is not necessary to explore every vertex of the graph during the search for the most influential ones. The experiments performed in [Melo and Vignatti 2018] indicate that a natural greedy algorithm that chooses enough high degree vertices gives a good preprocessing heuristic in graphs with power law degree distribution.

Although combining the spread of influence with complex networks is a natural way of studying the problem, we are not aware of other works dealing with the TSS problem in complex networks. In this work, we present preprocessing rules to obtain a partial optimal solution to the TARGET SET SELECTION problem. In the partial optimal solution, we efficiently (in linear time) identify some vertices that certainly belong to an optimal solution. Having a fraction of the graph solved optimally, obtaining a complete solution depends only on solving the remaining part, i.e., the part of the graph where the partial optimal solution was not obtained. The preprocessing has linear time complexity, indicating that it is worth performing the preprocessing before any strategy chosen later. In particular, due to the nature of the problem, it is of interest to evaluate the behavior of these preprocessing rules in real-world graphs. It is well known that many real-world graphs follow a power law degree distribution – such graphs are called power law graphs. In this way, through experiments on power law graphs, we show that our preprocessing rules obtain an optimal partial solution in a significant fraction of the graph, in some cases the preprocessing is sufficient to completely solve the problem. As the problem is NP-hard, an efficient way to solve it is not known. So the main advantage of our new approach is to accelerate the solving process in general graphs whose structure takes advantage of the use of the preprocessing rules, such as complex networks or, more specifically, power law graphs.

The rest of the paper is organized as follows. Section 2 is devoted to define the notation and the statement of the problem. In Section 3, we introduce the preprocessing rules to reduce the instance of the problem. In Section 4, we present an experimental evaluation using real-world complex networks and synthetic random graphs and discussion of our results. We conclude in Section 5.

2. Notation and definitions

For a directed graph G with vertices $V(G)$ and edges (or arcs) $E(G)$, consider the following notation. Denote by $N^{+}(v) = \{w : (v, w) \in E(G)\}$ the *out-neighborhood* of a vertex v. Similarly, $N^{-}(v) = \{u : (u, v) \in E(G)\}$ is the *in-neighborhood* of v. The *out-degree* of v is the number $\delta^{+}(v) = |N^{+}(v)|$, the *in-degree* of v is $\delta^{-}(v) = |N^{-}(v)|$, and $\delta(v) = \delta^{+}(v) + \delta^{-}(v)$ is the *degree* of v. We say that a vertex v with $\delta^{-}(v) = 0$ is a *source* and, conversely, when $\delta^{+}(v) = 0$ we say that v is a *sink*. Also, v is a *isolated* vertex when $\delta(v) = 0$.

A directed graph H is a *subgraph* of G, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. H is an *induced* subgraph of G, if every arc $(u, v) \in E(G)$ for which $u, v \in V(H)$, is also an arc of H. A directed graph G is called *acyclic*, or a DAG, if it has no directed cycles.

To represent how the influence spreads over a network, in this work, we consider the well-known diffusion model called the *threshold model*, introduced by [Granovetter 1978]. Unlike commonly used models like the linear threshold and independent cascade models [Kempe et al. 2003], this model is deterministic. We are given a directed graph G representing a social network, with vertices $V(G)$ modeling the individuals and arcs $E(G)$ representing the social relations between these individuals. Let $t : V(G) \to \mathbb{N}$ be a threshold function that models the "resistance" of an individual to become influenced. Each vertex is in one of two states, *active* or *inactive*. Informally, a vertex is active if it has been persuaded to adopt a new behavior (for instance, convinced to buy a new product), and inactive otherwise. Next, we formally explain the activation process. The process proceeds progressively, i.e., a vertex can change from inactive to active but not from active to inactive. At the beginning, a subset $S \subseteq V(G)$, the *target set*, is active and all the vertices in $V(G) \setminus S$ are inactive. At each time step, an inactive vertex can be activated by its neighbors. We assume that each vertex exerts the same influence over each neighbor. So, a vertex v gets active if at least $t(v)$ of its in-neighbors are active at the previous step. More precisely, let A_t be the set of active vertices at the time step t with $S = A_0$. We have $v \in A_t$ if $|N^{-}(v) \cap A_{t-1}| \geq t(v)$ or $v \in S$. The process runs until there are no more vertices to be activated. When a vertex v becomes active at time t because of its neighbors that are in A_{t-1}, we say that u exerts influence over v, for every $u \in N^{-}(v) \cap A_{t-1}$.

A *propagation graph* G^{*} is the subgraph of an input graph G with $V(G^{*}) = V(G)$, induced by a subset $E(G^{*}) \subseteq E(G)$ of arcs in which the influence is exerted, that is, for every $(u, v) \in E(G^{*})$ where u exerts influence over v, at the end of the activation process.

The problem definition is as follows.

Problem 1 (Target Set Selection (TSS))**.** *Given a directed graph $G = (V, E)$ with thresholds $t(v)$ on each $v \in V$, find a set $S \subseteq V$ of minimum size, such that the propagation graph G^* is acyclic with $\delta^-(v) \geq t(v)$ for every $v \in V(G^*) \setminus S$.*

3. Preprocessing rules

In this section, we present preprocessing rules to find an optimal partial solution to the TSS problem. As we intend to solve the problem on complex networks, the rules are designed with the properties of these networks in mind. Observe that, in the TSS problem, every vertex with no incoming edges certainly belongs to the optimal solution because no other vertex can activate it. So, we can include these vertices in the partial optimal solution. By the same reasoning, we can exclude from the optimal target set those vertices with no outgoing edges. It is worth noting that in graphs with power law degree distribution, these two types of vertices are the majority. Thus, we can exploit this graph topology by processing the low degree vertices first.

Consider a directed graph G as an instance of TSS problem. Let $S \subseteq V(G)$ be the optimal target set and let $A(S)$ be the set of active vertices at the end of the activation process. We can set the partial optimal solution according to the following propositions.

Proposition 1. *If $\delta^-(v) = 0$, then $v \in S$ for all $v \in V(G)$.*

Proof. Suppose, by contradiction, that $v \notin S$. By the activation process in the threshold model, no vertex in $A(S)$ activates v because $N^-(v) = \emptyset$. Therefore, $A(S) = V(G) \setminus \{v\}$ and S is not a solution of the problem, contradicting the hypothesis. □

Similarly to the reasoning of Proposition 1, vertices with $t(v) > \delta^-(v)$ must be in the target set S. So for Proposition 2, we consider that $t(v) \leq \delta^-(v)$.

Proposition 2. *If $\delta^+(v) = 0$, then $v \notin S$ for all $v \in V(G)$.*

Proof. Suppose, by contradiction, that $v \in S$ and let $S' = S \setminus \{v\}$. By the problem definition, $A(S) = V(G)$. Since $\delta^+(v) = 0$, v cannot activate any other vertex of G, then the set of vertices activated by S is not greater than the set of vertices activated by S', i.e., $A(S) \subseteq A(S')$. Furthermore, at the end of the activation process, every in-neighbor of v is active, i.e., $N^-(v) \subseteq A(S')$. Thus, v has to be activated unanimously by $N^-(v)$. Consequently, $v \in A(S')$ and $A(S') = V(G)$. This is a contradiction because S is a minimum target set. □

For disconnected graphs, isolated vertex must be selected as target set elements in advance and each connected component can be treated as a separate graph in the formulation.

From Propositions 1 e 2, we derive Algorithm 1. We start by creating a copying graph G' of the input graph G. The set $S \subseteq V(G')$ starts empty and will contain the optimal partial target set (all vertices satisfying Proposition 1). The set $D \subseteq V(G')$ stores the vertices to be removed from G' during the algorithm execution. The propagation graph G^* starts with no arcs but containing all the vertices of G. In this algorithm, we want to remove as many vertices as possible from G', aiming to decrease the size of the returned problem instance. The first part of the algorithm (lines 4-18) removes every source, sink, and isolated vertices. The condition in lines 5-7 set the isolated vertices to be in the partial solution S, and add them to the set D of vertices to be removed from G'. Based on Proposition 1, lines 8-13 adds every v with no incoming arcs to the partial optimal solution S, and further inserts the outgoing arcs of v to the propagation graph G^*. This can be done because such arcs do not belong to any directed cycle since v is a source vertex. As a consequence, we can decrease by one the threshold of each v's out-neighbor w in G' meaning that v exerts influence over w. Lines 14-17 deals with the case of Proposition 2, indicating that v is not in S if it has no outgoing arc. By the same reasoning, the incoming arcs are set to be in partial optimal solution G^* without risk of generating cycles. In line 18, the vertices in D will be removed from G' as well as the incident arcs to each $v \in D$. More precisely, the graph obtained by deleting D from G' is the subgraph induced by $V(G') \setminus D$, denoted as $G[V(G') \setminus D]$.

The iteration loop in lines 19-34 of Algorithm 1 proceeds by removing more vertices of G' according to the updated threshold $t(v)$ of each $v \in V(G')$. The first conditional considers that if $t(v) \leq 0$ (due steps 12 and 27), then the incoming arcs that were removed (and inserted in the propagation graph) in the early steps (or early iterations) are enough to activate v, so we can exclude the remaining incoming arcs of v of the reduced instance G'. This helps us to avoid cycles. Further, we also force the outgoing arcs of v to be in the solution and decrease the threshold of out-neighbors of v. Again, in lines 29-32, we repeat steps 14-17. We repeatedly remove vertices from G' that enter the conditionals until there are no more vertices to be removed.

Figure 1 presents an example of the algorithm execution in a small graph. The labels in each vertex denote the name and the threshold. For instance, the vertex in the upper left corner has the name a and threshold $t(a) = 1$. In the leftmost graph, the dashed vertices and arcs are the sources and sinks to be removed in the first part of the algorithm, lines 4-17. In the second graph, the sources and sinks were removed. Note that, due the line 12, vertex f has threshold 0. So f is inserted in set D to be removed later. Also, the arc (c, f) is excluded from the solution. The arc (c, f) can be excluded because the threshold of f is 0 at this moment, meaning that the arc (g, f), which is already in the solution, is enough to achieve the threshold of f and activate it. Furthermore, by removing the arc (c, f), we are eliminating the cycle (c, f, d) of the solution. Vertex h will be removed because of the condition in lines 29-32. That is, g can activate f, and as a consequence, h can be activated by f, mimicking the diffusion process of the threshold model. In the rightmost graph, the search stops because there is no vertex to be removed. So, the reduced graph G' is the remaining 2-cycle composed by vertices c and d.

Theorem 3 bounds the time complexity of Algorithm 1.

Theorem 3. *For a directed graph G with $|V(G)| = n$ and $|E(G)| = m$, Algorithm 1 takes $O(m)$ time in the worst case.*

Proof. At line 2, the time complexity for cloning the graph is $O(m)$. Next, creating G^* demands $O(n)$ to copy the vertices of G. The loop in lines 4-17 visits every vertex once, and for each vertex, it iterates through its set of neighbors, overall, this takes $O(m)$. At line 18, given a set of vertices marked for removal, each vertex must be removed along with its incident edges. In the worst case, it takes $O(m)$ time. Next, we argue that

Algorithm 1: PREPROCESSING

```
Input: Graph $G$, threshold function $t : V(G) \to \mathbb{N}$
Output: Reduced graph $G'$, partial propagation graph $G^*$, and partial target
        set $S \subseteq V(G)$
begin
    $G' \leftarrow G$; $S \leftarrow \emptyset$; $D \leftarrow \emptyset$
    Create the graph $G^*$ such that $V(G^*) = V(G)$ and $E(G^*) = \emptyset$
    foreach $v \in V(G')$ do
        if $\delta(v) = 0$ then // isolated vertex
            $S \leftarrow S \cup \{v\}$
            $D \leftarrow D \cup \{v\}$
        else if $\delta^-(v) = 0$ or $t(v) > \delta^-(v)$ then // source vertex
            $S \leftarrow S \cup \{v\}$
            foreach $w \in N^+(v)$ do
                add arc $(v, w)$ to propagation graph $G^*$
                $t(w) \leftarrow t(w) - 1$
            $D \leftarrow D \cup \{v\}$
        else if $\delta^+(v) = 0$ then // sink vertex
            foreach $u \in N^-(v)$ do
                add arc $(v, w)$ to propagation graph $G^*$
            $D \leftarrow D \cup \{v\}$
    $G' \leftarrow G'[V(G') \setminus D]$ ;            // remove vertices from $G'$
    repeat
        $D \leftarrow \emptyset$
        foreach $v \in V(G')$ do
            if $t(v) \leq 0$ then
                foreach $u \in N^-(v)$ do
                    remove arc $(u, v)$ from $G'$
                foreach $w \in N^+(v)$ do
                    add arc $(v, w)$ to propagation graph $G^*$
                    $t(w) \leftarrow t(w) - 1$
                $D \leftarrow D \cup \{v\}$
            else if $\delta^+(v) = 0$ then // sink vertex
                foreach $u \in N^-(v)$ do
                    add arc $(v, w)$ to propagation graph $G^*$
                $D \leftarrow D \cup \{v\}$
        $G' \leftarrow G'[V(G') \setminus D]$
    until $D = \emptyset$;// there are no vertices to be removed
```

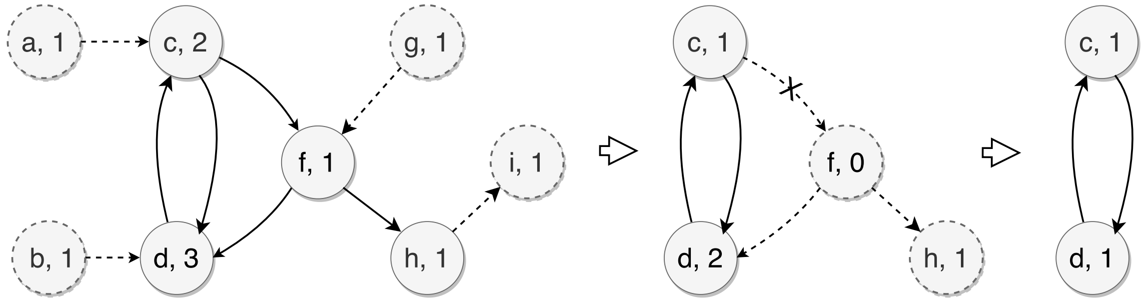


Figure 1. Example of the algorithm execution.

the block in lines 19-34 takes $O(m)$ time. Each vertex in $V(G')$ meets the conditionals of lines 22 or 29 at most once because if it meets any of the conditionals, then it is added to D, and immediately afterward it is removed from $V(G')$. For each vertex that meets the conditionals, it iterates through its set of neighbors. Thus, overall, the time spent is proportional to the sum of the degrees, i.e., $O(m)$. However, the loop at line 21 must be carefully implemented, as if in every iteration of line 21, we go through all the vertices, then we could end up with a quadratic time algorithm. We can overcome this issue with a careful implementation, where we keep track of only those vertices that meet the conditionals so that we do not iterate over unnecessary vertices. Finally, the analysis of line 33 is similar to line 18, where the accumulated time over all iterations in the worst case is $O(m)$. □

It is noteworthy to mention the fact stated in Theorem 4.

Theorem 4. *If $V(G') = \emptyset$ when the Algorithm 1 ends, then the propagation graph defined by the arcs chosen in the algorithm is an optimal solution for the TSS problem.*

Proof. As argued before, the vertices in S belong to an optimal target set. Besides, the removed vertices are related to the activated vertices in the diffusion process, so removing all the vertices of G' means that the whole graph is activated. Therefore, we have an optimal partial solution that activates all the vertices in the graph, i.e., the partial solution is optimal, and S is an optimal target set. □

3.1. Expected size of the solution on power law graphs

The success of Algorithm 1 depends on the topology of the input graph. For example, it has no effect on strongly connected graphs and in graphs with no sources or sinks. On the other hand, the algorithm can perform well in power law graphs, because in this case, the majority of the vertices has low degree, with many having only one incoming arc or one outgoing arc. Moreover, given a power law graph model, for example, $P(\alpha, \beta)$ presented by [Aiello et al. 2001], it is easy to quantify the expected number of vertices belonging to this category of low degree vertices. In real-world networks, the power law exponent β is typically in the range $2 < \beta < 3$ [Choromanski et al. 2013]. Thus, we can quantify the expected decrease in the instance size after the preprocessing. On the $P(\alpha, \beta)$ model, the number of vertices with degree 1 is e^{α}, and $|V|$ is (asymptotically) $e^{\alpha}\zeta(\beta)$, where $\zeta(\beta)$ is the Riemann zeta function. Therefore, the vertices of degree 1 correspond to $\frac{1}{\zeta(\beta)}$ of the total number of vertices. Assuming $2 < \beta < 3$, from 60.7% to

83.1% of the vertices of the graph have degree 1. A vertex of degree 1 has either in-degree or out-degree equal to zero, therefore, regardless of the case, it is always removed from the original graph by Algorithm 1. In addition to the number of vertices with degree 1, there are also vertices with degree greater than 1 which has no in-edges (or out-edges). Therefore Algorithm 1 sets a huge fraction of vertices as belonging or not to the optimal partial solution, significantly reducing the size of the instance. These properties have also been empirically confirmed, as we show in Section 4.

4. Computational experiments

In this section, we proceed with the experimental evaluation. The experiments were launched in an Intel(R) Core(TM) i5-3210M CPU @ 2.5GHz and 4 GB RAM. The algorithms were implemented in Java 11 language. For graph manipulations, we use the JGraphT 1.4 library [Michail et al. 2019].

4.1. Instance size reduction

To demonstrate the behavior of Algorithm 1 in reducing the size of the input graph, we consider some real-world complex networks obtained from the Stanford Large Network Dataset Collection [Leskovec and Krevl 2014]. Below, we give a brief description of each considered network.

- Bitcoin-alpha: Who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin Alpha.
- Bitcoin-OTC: Who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin OTC.
- Wiki-vote: Contains all the Wikipedia voting data from the inception of Wikipedia until January 2008.
- DBLP: Citation network of DBLP, a database of scientific publications such as papers and books.
- Reddit: Network of subreddit-to-subreddit hyperlinks extracted from hyperlinks in the body of the post.
- Epinions: A who-trust-whom online social network of a general consumer review site Epinions.com.
- Slashdot09: Slashdot social network from February 2009.
- Email-EuAll: Email network of a large European Research Institution (between October 2003 and March 2005).

To show the results in different scenarios, we consider three distinct threshold functions, in line with the threshold scenarios considered in [Chen et al. 2009, Ackerman et al. 2010]. First, the majority threshold is defined as $t(v) = \left\lceil \frac{\delta^-(v)}{2} \right\rceil$ for every $v \in V$. Second, the low threshold is $t(v) = \left\lceil \frac{\delta^-(v)}{4} \right\rceil$ for every $v \in V$. Lastly, the high thresholds are defined as $t(v) = \left\lceil \frac{3}{4}\delta^-(v) \right\rceil$ for every $v \in V$.

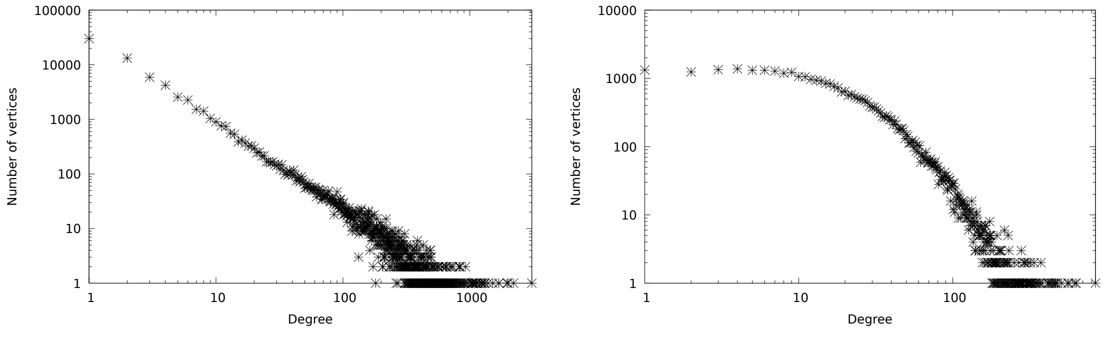
Table 1 shows the reduction of the instance size obtained after the preprocessing in graphs representing real-world networks. For each graph, we present the number of vertices ($|V|$) and arcs ($|E|$) in the original graph and the reduction after the preprocessing on each considered scenario. Column “majority threshold” shows the absolute size (number of vertices and edges) of the remaining graph together with the percentage of the

Table 1. Reduction in size of real social networks after preprocessing.

Network		Original	Reduction		
			Majority Threshold	Low Threshold	High Threshold
Bitcoin-alpha	$\|V\|$	3,783	3,251 (14.1%)	-	-
	$\|E\|$	24,186	23,333 (3.5%)	-	-
Bitcoin-OTC	$\|V\|$	5,881	4,763 (19.0%)	-	-
	$\|E\|$	35,592	33,640 (5.5%)	-	-
Wiki-vote	$\|V\|$	7,115	0 (100%)	-	81.9%
	$\|E\|$	103,689	0 (100%)	-	62.1%
DBLP	$\|V\|$	12,591	2 (99.9%)	-	99.5%
	$\|E\|$	49,728	2 (99.9%)	-	99.7%
Reddit	$\|V\|$	35,776	9,854 (72.5%)	99.7%	67.0%
	$\|E\|$	137,821	88,343 (35.9%)	99.9%	28.5%
Epinions	$\|V\|$	75,888	32,088 (57.7%)	98.4%	54.3%
	$\|E\|$	508,837	438,163 (13.9%)	99.7%	11.7%
Slashdot09	$\|V\|$	82,168	71,862 (12.5%)	-	12.5%
	$\|E\|$	870,161	842,217 (3.2%)	-	3.2%
Email-EuAll	$\|V\|$	265,214	56 (99.9%)	-	88.5%
	$\|E\|$	418,956	104 (99.9%)	-	69.7%

reduction after applying the preprocessing rules. For example, in the Wiki-vote network, we get a reduction of 100% in the size of both vertex and edge set and thus 0 remaining vertices and edges. This means that we have the best possible case, i.e., in this example, the problem is completely solved by our preprocessing strategy. The last two columns contain the results for the low and high threshold scenarios, and we show the percentage of the reduction. Dashed cells mean that the reduction is exactly the same as the first scenario (majority threshold), this similarity happens because in most cases only the sources and sinks were eliminated by the preprocessing rules. Note that, with low thresholds, the gains are more expressive and, on the contrary, when the thresholds are high the reduction is less noticeable.

Figure 2 shows the degree distribution, in logarithmic scale, of two differ-



(a) Straight shape.

(b) Curved shape.

Figure 2. Degree distribution in logarithmic scale of real world networks.

ent real-world networks, Epinions, in Figure 2a, and HEP-PH (citation graph of High Energy Physics Phenomenology [Leskovec and Krevl 2014]), in Figure 2b. The straight-line shape in Figure 2a indicates that the network is a power law graph [Mitzenmacher and Upfal 2005, Clauset et al. 2009], while a curved shape says that it is not a power law. In our experiments of Table 1 we only use networks that are power law. However, we note that many real networks present a "curved" degree distribution as in Figure 2b. For such cases, we also experimentally observed a substantial reduction in the size of the graph. We believe that this good behavior is because there are even more vertices of a low degree than in the case with a straight degree distribution.

4.2. Scalability

Motivated by the large sizes of real-world networks, it is necessary to understand the execution time behavior of the algorithm as the size of the network increases, i.e., the scalability of the algorithm. To experiment with networks of varying sizes, we use synthetic graph models for power law graphs. More specifically, among the various models of random power law graphs, we choose the Bollobás model [Bollobás et al. 2003], as it is tailor-made to generate power law directed graphs. In this model, the graph grows one edge per step, according to the probabilities a, b and c, where $a + b + c = 1$. Here, we set these values as $a = b = 0.33$, and $c = 0.34$. Were generated ten graphs of each size and considered the average running time.

Figure 3 illustrates how is the growth curve of the running time of Algorithm 1. The size of power law graphs ranges from 1,000 up to 50,000 vertices. The running time is the average between 10 generated graphs for each size. The line in the plot is not smoothed or straight in shape due to the precision of milliseconds, but it has a pattern that suggests linear growth. So Figure 3 empirically confirms the result stated theoretically in Theorem 3, that is, the increasing of the running time of PREPROCESSING is linear in the size of the input graph.

Taking advantage of the fact that we did experiments on synthetic graphs, it is worth mentioning the performance of our algorithm concerning the reduction in the size of the instance. In such a case, for 500 generated graphs, more than 99% were solved by the PREPROCESSING. Only 0.02% resulted in a not empty reduced graph G', even in this case, the resulting graph G' is a cycle of size 2 or 3, which is trivial to solve. This happens mainly because our strategy naturally behaves better in sparse graphs, and the random graphs generated by the model of [Bollobás et al. 2003] are usually very sparse. However, it is worth analyzing whether other models of power law graphs maintain this behavior.

5. Conclusion

Based on graph properties that commonly appear in complex networks, we presented a strategy that provides a notable reduction of the TSS problem search space. Despite the simplicity, the experiments indicate that our strategy provides tractability to the problem in power law graphs. In addition, this idea can be combined with sophisticated techniques and algorithms, in order to gain scalability. With the objective of refining this technique to achieve better results, there are some future directions. First, combine our preprocessing rules with other techniques with similar proposes, aiming to preprocess

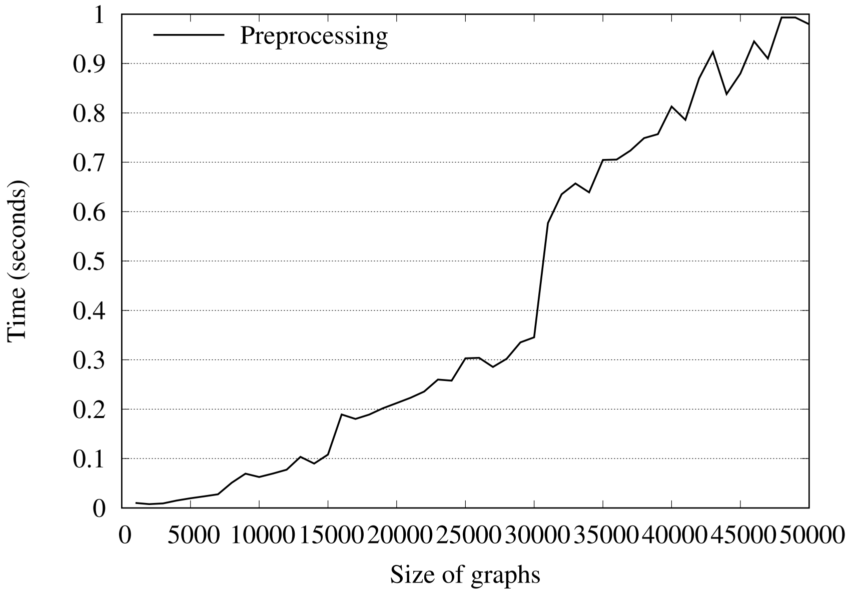


Figure 3. Order of growth of the running time of the PREPROCESSING algorithm.

more general graphs besides power law. Furthermore, incorporate these techniques into a linear programming based branch-and-bound framework together with lower and upper bounds, in order to accelerate the solving time in such techniques.

References

Ackerman, E., Ben-Zwi, O., and Wolfovitz, G. (2010). Combinatorial model and bounds for target set selection. *Theoretical Computer Science*, 411(44-46):4017–4022.

Aiello, W., Chung, F., and Lu, L. (2001). A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66.

Bollobás, B., Borgs, C., Chayes, J., and Riordan, O. (2003). Directed scale-free graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 132–139. Society for Industrial and Applied Mathematics.

Chen, N. (2009). On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400–1415.

Chen, W., Wang, C., and Wang, Y. (2010a). 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*, pages 1029–1038. ACM.

Chen, W., Wang, Y., and Yang, S. (2009). Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 199–208. ACM.

Chen, W., Yuan, Y., and Zhang, L. (2010b). Scalable influence maximization in social networks under the linear threshold model. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 88–97. IEEE.

Choromanski, K., Matuszak, M., and Miekisz, J. (2013). Scale-free graph with preferential attachment and evolving internal vertex structure. *J Stat Phys*, 151:1782–1789.

Clauset, A., Shalizi, C. R., and Newman, M. E. (2009). Power-law distributions in empirical data. *SIAM review*, 51(4):661–703.

Domingos, P. and Richardson, M. (2001). Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57–66. ACM.

Goyal, A., Bonchi, F., and Lakshmanan, L. V. (2011a). A data-based approach to social influence maximization. *Proceedings of the VLDB Endowment*, 5(1):73–84.

Goyal, A., Lu, W., and Lakshmanan, L. V. (2011b). Simpath: An efficient algorithm for influence maximization under the linear threshold model. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 211–220. IEEE.

Granovetter, M. (1978). Threshold models of collective behavior. *American journal of sociology*, pages 1420–1443.

Kempe, D., Kleinberg, J., and Tardos, É. (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*, pages 137–146. ACM.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Liu, X., Li, S., Liao, X., Peng, S., Wang, L., and Kong, Z. (2014). Know by a handful the whole sack: efficient sampling for top-k influential user identification in large graphs. *World Wide Web*, 17(4):627.

Melo, R. S. and Vignatti, A. L. (2018). A preselection algorithm for the influence maximization problem in power law graphs. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 1782–1789. ACM.

Michail, D., Kinable, J., Naveh, B., and Sichi, J. V. (2019). Jgrapht–a java library for graph data structures and algorithms. *arXiv preprint arXiv:1904.08355*.

Mitzenmacher, M. and Upfal, E. (2005). *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press.

Tang, Y., Xiao, X., and Shi, Y. (2014). Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 75–86. ACM.

Wang, F., Du, H., Camacho, E., Xu, K., Lee, W., Shi, Y., and Shan, S. (2011). On positive influence dominating sets in social networks. *Theoretical Computer Science*, 412(3):265–269.

Zhang, W., Wu, W., Wang, F., and Xu, K. (2012). Positive influence dominating sets in power-law graphs. *Social Network Analysis and Mining*, 2(1):31–37.