Limiting the Neighborhood: De-Small-World Network for Outbreak Prevention

Ruoming Jin[†] Yelong Sheng[†] Lin Liu[†] Xue-Wen Chen[‡]
[†] Department of Computer Science
Kent State University
{jin,ysheng,lliu}@cs.kent.edu
[‡] Department of EECS
The University of Kansas
xwchen@ku.edu

ABSTRACT

In this work, we study a basic and practically important strategy to help prevent and/or delay an outbreak in the context of network: limiting the contact between individuals. In this paper, we introduce the *average neighborhood size* as a new measure for the degree of being small-world and utilize it to formally define the *de-small-world* network problem. We also prove the NP-hardness of the general reachable pair cut problem and propose a greedy edge betweenness based approach as the benchmark in selecting the candidate edges for solving our problem. Furthermore, we transform the de-small-world network problem as an OR-AND Boolean function maximization problem, which is also an NP-hardness problem. In addition, we develop a numerical relaxation approach to solve the Boolean function maximization and the de-small-world problem. Also, we introduce the *short-betweenness*, which measures the edge importance in terms of all short paths with distance no greater than a certain threshold, and utilize it to speed up our numerical relaxation approach. The experimental evaluation demonstrates the effectiveness and efficiency of our approaches.

1. INTRODUCTION

The interconnected network structure has been recognized to play a pivotal role in many complex systems, ranging from natural (cellular system), to man-made (Internet), to the social and economical systems. Many of these networks exhibit the "small-world" phenomenon, i.e., any two vertices in the network is often connected by a small number of intermediate vertices (the shortest-path distance is small). The small-world phenomenon in the real populations was first discovered by Milgram [13]. In his study, the average distance between two Americans is around 6. Several recent studies [11, 14, 8] offer significant evidence to support similar observations in the online social networks and Internet itself. In addition, the power-law degree distribution (or scale-free property) which many of these networks also directly lead to the small average distance [1]. Clearly, the small-world property can help facilitate the communication and speed up the diffusion process and information spreading in a large network.

However, the small-world effect can be a dangerous *double-edged sword*. When a system is benefited from the efficient communication and fast information diffusion, it also makes itself more vulnerable to various attacks: diseases, (computer) virus, spams, and misinformation, etc. For instance, it has been shown that a small-world graph can have much faster disease propagation than a regular lattice or a random graph [15]. Indeed, the six degrees of separation may suggest that a highly infectious disease could spread to all six billion people living in the earth about size incubation periods of the diseases [15]. The small-word property of Internet and WWW not only enables the computer virus and spams to be much easier to spread, but also makes them hard to stop. More recently, the misinformation problem in the social networks has made several public outcry [3]. These small-world online social network potentially facilitate the spread of misinformation to reach a large number of audience in short time, which may cause public panic and have other disruptive effects.

To prevent an outbreak, the most basic strategy is to remove the affected individuals (or computers) from the network system, like quarantine. However, in many situations, the explicit quarantine may be hard to achieve: the contagious individuals are either unknown or hard to detect; or it is often impossible to detect and remove each infected individual; or there are many already being affected and it become too costly to remove all of them in a timely fashion. Thus, it is important to consider alternative strategies to help prevent and even delay the spreading where the latter can be essential in discovering and/or deploying new methods for dealing with the outbreaks.

Recently, there have been a lot of interests in understanding the network factors (such as the small-word and scale-free properties) in the epidemics and information diffusion process, and utilizing the network structures in detecting/preventing the outbreaks. Several studies have focused on modeling the disease epidemics on the small-world and/or scale-free networks [17, 15], [16]; in [12], Leskovec *et al.* study how to deploy sensors cost-effectively in a network system (sensors are assigned to vertices) to detect an outbreak; in [3], Budak *et al.* consider how to limit the misinformation by identifying a set of individuals that are needed to adopt the "good" information (being immune in epidemics) in order to minimize those being affected by the "bad" information (being infected in epidemics). In addition, we note that from a different angle (viral marketing), there have been a list of studies on the *influence maximization* problem [18, 9], which aim to discover a set of most influential seeds to maximize the information spreading in the network. From the disease epidemics perspective, those seeds (assuming being selected using contagious model) may need particular protection to prevent an outbreak.

In this work, we study another basic and practically important strategy to help prevent and/or delay an outbreak in the context of network: limiting the contact between individuals. Different from the pure quarantine approach, here individuals can still perform in the network system, though some contact relationships are forbidden. In other words, instead of removing vertices (individuals) form a network as in the quarantine approach, this strategy focuses on removing edges so that the (potential) outbreaks can be slowed down. Intuitively, if an individual contacts less number of other individuals, the chance for him or her to spread or being infected from the disease (misinformation) becomes less. From the network viewpoint, the edge-removal strategy essentially make the underlying (social) network less small-world, or simply "de-small-world", i.e., the distances between individuals increase to delay the spreading process. In many situations, such a strategy is often easily and even voluntarily adopted. For instance, during the SARS epidemic in Beijing, 2004, there are much less people appearing in the public places. In addition, this approach can also be deployed in complement to the quarantine approach.

1.1 Our Contribution

Even though the edge-removal or *de-small-world* approach seems to be conceptually easy to understand, its mathematical foundation is still lack of study. Clearly, different edges (interactions) in the network are not being equivalent in terms of slowing down any potential outbreak: for a given individual, a link to an individual of high degree connection can be more dangerous than a link to another one with low degree connection. The edge importance (in terms of distance) especially coincides with Kleighnberg's theoretical model [10] which utilizes the *long-range edges* on top of an underlying grid for explaining the small-world phenomenon. In this model, the long-range edges are the main factors which help connect the otherwise long-distance pairs with a smaller number of edges. However, there are no direct studies in fitting such a model to the real world graph to discover those long-range edges. In the mean time, additional constraint, such as the number of edges can be removed from the network, may exist because removing an edge can associate with certain cost. These factors and requirements give arise to the following fundamental research problem: *how can we maximally de-small-world a graph (making a graph to be less small-world) by removing a fixed number of edges?*

To tackle the *de-small-world* network problem, we make the following contributions in this work:

1. We introduce the *average neighborhood size* as a new measure for the degree of being small-world and utilizes it to formally define the de-small-world network problem. Note that the typical average distance for measuring the small-world effects cannot uniformly treat the connected and disconnected networks; neither does it fit well with the spreading process. We also reformulate the de-small-world as the *local-reachable pair cut* problem.

2. We prove the NP-hardness of the general reachable pair cut problem and propose a greedy edge betweenness based approach as the benchmark in selecting the candidate edges for solving the de-small-world network. We transform the de-small-world network problem and express it as a OR-AND Boolean function maximization problem, which is also an NP-hard problem.

3. We develop a numerical relaxation approach to solve the de-small-world problem using its OR-AND boolean format. Our approach can find a local minimum based on the iterative gradient optimization procedure. In addition, we further generalize the betweenness measure and introduces the *short betweenness*, which measures the edge importance in terms of all the paths with distance no greater than a certain threshold. Using this measure, we can speed up the numerical relaxation approach by selecting a small set of candidate edges for removal.

4. We perform a detailed experimental evaluation, which demonstrates the effectiveness and efficiency of proposed approaches.

2. PROBLEM DEFINITION AND PRELIMINARY

In this section, we first formally define the *de-small-world* network problem and prove its NP-hardness in (Subsection 2.1); then we introduce the basic greedy approaches based on edge betweenness which will serve as the basic benchmark (Subsection 2.2); and finally we show the de-small-world network problem can be transformed and expressed as a OR-AND Boolean function maximization problem (Subsection 2.3).

2.1 Problem Formulation

In order to model the edge-removal process and formally define the *de-small-world* network problem, a criterion is needed to precisely capture the *degree* of being small-world. Note that here the goal is to help prevent and/or delay the potential outbreak and epidemic process. The typical measure of small-world network is based on the average distance (the average length of the shortest path between any pair of vertices in the entire network). However, this measure is not able to provide unified treatment of the connected and cut network. Specifically, assuming a connected network is broken into several cut network and the average distance on the cut network is not easy to express. On the other hand, we note that the de-small-world network graph problem is different from the network decomposition (clustering) problem which tries to break the entire network into several components (connected subgraphs). From the outbreak prevention and delaying perspective, the cost of network decomposition is not only too high, but also may not be effective. This is because each individual component itself may still be small-world; and the likelihood of completely separating the contagious/infected group from the rest of populations (the other components) is often impossible.

Given this, we introduce the *average neighborhood size* as a new measure for the degree of being small-world and utilize it to formally define the de-small-world network problem. Especially, the new measure can not only uniformly treat both connected and cut networks and aims to directly help model the spreading/diffusion process. Simply speaking, for each vertex v in a network $G = (V, E)$ where V is the vertex set and E is the edge set, we define the neighborhood of v as the number of vertices with distance no greater than k to v, denoted as $N^k(v)$. Here k is the user-specified *spreading* (or delaying) parameter which aims to measure the outbreak speed, i.e., in a specified time unit, the maximum distance between individual u (source) to another one v (destination) who can be infected if u is infected. Thus, the average neighborhood size of G, $\sum_{v \in V} N^k(v)$, can be used to measure the robustness of the network with respect to a potential outbreak in a certain time framework. Clearly, a potential problem of the small-world network is that even for a small k, the average neighborhood size can be still rather large, indicating a large (expected) number of individuals can be quickly affected (within time framework k) during an outbreak process.

Formally, the *de-small-world* network problem is defined as follows:

DEFINITION 1. (**De-Small-World Network Problem**) *Given the edge-removal budget $L > 0$ and the spreading parameter $k > 1$ we seek a subset of edges $E_r \subset E$, where $|E_r| = L$, such that the average neighborhood size is minimized:*

$$\min_{|E_r|=L} \frac{\sum_{v \in V} N^k(v|G \backslash E_r)}{|V|}, \qquad (1)$$

where $N^k(v|G\backslash E_r)$ is the neighborhood size of v in the graph G after removing all edges in E_r from the edge set E.

Note that in the above definition, we assume each vertex has the equal probability to be the source of infection. In the general setting, we may consider to associate each vertex v with a probability to indicate its likelihood to be (initially) infected. Furthermore, we may assign each edge with a weight to indicate the cost to removing such an edge. For simplicity, we do not study those extensions in this work; though our approaches can be in general extended to handle those additional parameters. In addition, we note that in our problem, we require the spreading parameter $k > 1$. This is because for $k = 1$, this problem is trivial: the average neighborhood size is equivalent to the average vertex degree; and removing any edge has the same effect. In other words, when $k = 1$, the neighborhood criterion does not capture the spreading or cascading effects of the small-world graph. Therefore, we focus on $k > 1$, though in general k is still relatively small (for instance, no higher than 3 or 4 in general).

Reachable Pair Cut Formulation: We note the de-small-world network problem can be defined in terms of the *reachable pair cut* formulation. Let a pair of two vertices whose distance is no greater than k is referred to as a *local-reachable pair* or simply *reachable pair*. Let $\mathcal{R}_G$ record the set of all local reachable pairs in G.

DEFINITION 2. *(***Reachable Pair Cut Problem***) For a given local (u, v), if $d(u, v|G) \leq k$ in G, but $d(u, v|G \backslash E_s) > k$, where E_s is an edge set in G, then we say (u, v) is being* **local cut** *(or simply cut) by E_s. Given the edge-removal budget $L > 0$ and the spreading parameter $k > 1$, the reachable pair cut problem aims to find the edge set $E_r \subseteq E$, such that the maximum number of pairs in $\mathcal{R}_G$ is cut by E_r.*

Note that here the (local) cut for a pair of vertices simply refers to increase their distance; not necessarily completely disconnect them in the graph $(G \backslash E_s)$. Also, since $\mathcal{R}_{G \backslash E_r} \subseteq \mathcal{R}_G$, i.e., every local-reachable pair in the remaining network $G \setminus E_r$ is also the local-reachable in the original graph G, the problem is equivalently to maximize $|\mathcal{R}_G| - |\mathcal{R}_{G \backslash E_r}|$ and minimize the number of local reachable pairs $|\mathcal{R}_{G \backslash E_r}|$. Finally, the correctness of such a reformulation (de-small-world problem=reachable pair cut problem) follows this simple observation: $\sum_{v \in V} N^k(v|G) = 2|\mathcal{R}_G|$ (and $\sum_{v \in V} N^k(v|G \backslash E_r) = 2|\mathcal{R}_{G \backslash E_r}|$). Basically, every reachable pair is counted twice in the neighborhood size criterion.

In the following, we study the hardness of the general reachable pair cut problem.

THEOREM 1. *Given a set RS of local reachable pairs in $G = (V, E)$ with respect to k, the problem of finding L edges $E_r \subseteq E$ $(|E_r| = L)$ in G such that the maximal number of pairs in RS being cut by E_r is NP-Hard.*

Note that in the general problem, RS can be any subset of $\mathcal{R}_G$. The NP-hardness of the general reachable pair cut problem a strong indicator that the de-small-world network problem is also hard. The proof of Theorem 1 is in Appendix. In addition, we note that the submodularity property plays an important role in solving vertex-centered maximal influence [9], outbreak detection [12], and limiting misinformation spreading [3] problems. However, such property does not hold for the edge-centered de-small-world problem.

LEMMA 1. *Let set function $f : 2^E \rightarrow Z^+$ records the number of local reachable pairs in $\mathcal{R}_G$ is cut by an edge set E_s in graph G. Function f is neither submodular (diminishing return) nor supermodular.*

Proof Sketch: For a supermodular function: we have $f(A \cup B) + f(A \cap B) \geq f(A) + f(B)$; for a submodular function: we need show $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. Here we use two counter examples (suppose $k = 2$). For G_1 in Figure 1, we can see that edge sets $\{e_1\}$ and $\{e_2\}$ cut (cut) respectively two reachable pairs $\{(ab, ac\}$ and $\{ac, bc\}$. Then we have $f(\{e_1\}) + f(\{e_2\}) = 4$; however, $f(\{e_1, e_2\}) + f(\emptyset) = 3$. Therefore, supermodularity does not hold. For G_2 in Figure 1, we can see that edge set $\{e_1\}$ and $\{e_3\}$ each can only cut one reachable pair. However, $\{e_1, e_3\}$ could cut four pairs. That means, $f(\{e_1, e_3\}) + f(\emptyset) > f(\{e_1\}) + |f(\{e_3\})$. Therefore, submodularity can not hold. $\Box$

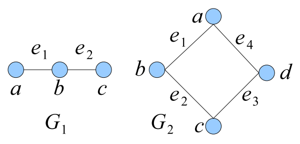


Figure 1: Counter Examples

2.2 Greedy Betweenness-Based Approach

Finding the optimal solution for the de-small-world problem is likely to be NP-hard. Clearly, it is computationally prohibitive to enumerate all the possible removal edge set E_r and to measure how many reachable pairs could be cut or how much the average neighborhood size is reduced. In the following, we describe a greedy approach to heuristically discover a solution edge-set. This approach also serves as the benchmark for the de-small-world problem.

The basic approach is based on the edge-betweenness, which is a useful criterion to measure the edge importance in a network. Intuitively, the edge-betweenness measures the edge important with respect to the shortest paths in the network. The high betweenness suggests that the edge is involved into many shortest paths; and thus removing them will likely increase the distance of those pairs linked by these shortest paths. Here, we consider two variants of edge-betweenness: the (global) edge-betweenness [5] and the local edge-betweenness [6]. The global edge-betweenness is the original one [5] and is defined as follows:

$$B(e) = \sum_{s \neq t \in V} \frac{\delta_{st}(e)}{\delta_{st}},$$

where δ_{st} is the total number of shortest paths between vertex s and t, and $\delta_{st}(e)$ the total number of shortest paths between u and v containing edge e.

The local edge-betweenness considers only those vertex pairs whose shortest paths are no greater than k, and is defined as

$$LB(e) = \sum_{s \neq t \in V, d(s,t) \leq k} \frac{\delta_{st}(e)}{\delta_{st}},$$

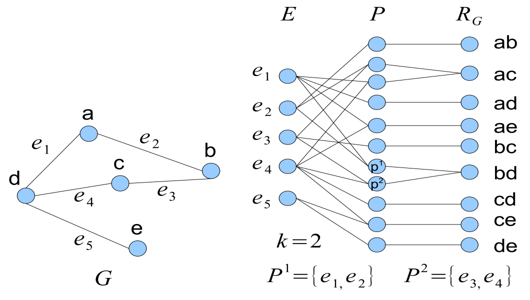
The reason to use the local edge-betweenness measure is because in the de-small-world (and reachable pair cut) problem, we focus on those local reachable pairs (distance no greater than k). Thus, the contribution to the (global) betweenness from those pairs with distance greater than k can be omitted. The exact edge-betweenness can be computed in $O(nm)$ worst case time complexity [2] where $n = |V|$ (the number of vertices) and $m = |E$ (the number of edges) in a given graph, though in practical the local one can be computed much faster.

Using the edge-betweenness measure, we may consider the following *generic procedure* to select the L edges for E_r:
1) Select the top $r < L$ edges into E_r, and remove those edges from the input graph G;
2) Recompute the betweenness for all remaining edges in the updated graph G;
3) Repeat the above procedure $\lceil L/r \rceil$ times until all L edges are selected.

Note that the special case $r = 1$, where we select each edge in each iteration, the procedure is very similar to the Girvan-Newman algorithm [5] in which they utilize the edge-betweenness for community discovery. Gregory [6] generalizes it to use the local-edge betweenness. Here, we only consider to pickup L edges and allow users to select the frequency to recompute the edge-betweenness (mainly for efficiency consideration). The overall time complexity of the betweenness based approach is $O(\lceil L/r \rceil nm)$ (assuming the exact betweenness computation is adopted).

2.3 OR-AND Boolean Function and its Maximization Problem

In the following, we transform the de-small-world network problem and express it as a OR-AND Boolean function maximization problem, which forms the basis for our optimization problem in next section. First, we will utilize the OR-AND graph to help represent the de-small-world (reachable pair cut) problem. Let us denote P the set of all the short paths in G that have length at most k.



(a) Example Graph (b) OR-AND Graph

Figure 2: *OR-AND* graph

OR-AND Graph: Given a graph $G = (V, E)$, the vertex set of an *OR-AND* graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is comprised of three kinds of nodes $\mathcal{V}_E$, $\mathcal{V}_P$ and $\mathcal{V}_{\mathcal{R}_G}$, where each node in $\mathcal{V}_E$ corresponds to a unique edge in E, each node in $\mathcal{V}_P$ corresponds to a short path in P, and each node in $\mathcal{V}_{\mathcal{R}_G}$ corresponds to a unique reachable pair in G (with respect to k). Figure 2(b) shows those nodes for graph G in Figure 2(a). The edge set consists of two types of edges: 1) Each short path node in $\mathcal{V}_P$ is linked with the vertices in $\mathcal{V}_E$ corresponding to those edges in in the path. For instance path node p^1 in $\mathcal{V}_P$ links to edge node e_1 and e_2 in $\mathcal{V}_E$ in Figure 2(b). Each reachable pair node in $\mathcal{V}_{\mathcal{R}_G}$ links to those path nodes which connects the reachable pair . For instance, the reachable pair *bd* is connected with path node p^1 and p^2 in Figure 2(b).

Intuitively, in the OR-AND graph, we can see that in order to cut a reachable pair, we have to cut *all* the short paths between them (AND). To cut one short path, we need to remove only one edge in that path (OR). Let $P(u, v)$ consists all the (simple) short paths between u and v whose length are no more than k. For each short path p in $P(u, v)$, let e corresponds to a Boolean variable for edge $e \in p$: if $e_i = T$, then the edge e_i is not cut; if $e_i = F$, then the edge is cut ($e_i \in E_r$). Thus, for each reachable pair $(u, v) \in \mathcal{R}_G$, we can utilize the a Boolean OR-AND expression to describe it:

$$I(u, v) = \bigvee_{p \in P(u,v)} \bigwedge_{e \in p} e \qquad (2)$$

For instance, in the graph G (Figure 2(b)),

$$I(b, d) = (e_1 \wedge e_2) \vee (e_3 \wedge e_4)$$

Here, $I(b, d) = T$ indicating the pair is being cut only if for both p^1 and p^2 are cut. For instance, if $e_1 = F$ and $e_3 = F$, then $I(b, d) = F$; and $e_1 = F$, but $e_3 = T$ and $e_4 = T$, $I(b, d) = T$. Given this, the de-small-world problem (and the reachable pair cut problem) can be expressed as the following Boolean function maximization problem.

DEFINITION 3. ***(Boolean Function Maximization Problem)*** Given a list of Boolean functions (such as $I(u, v)$, where $(u, v) \in \mathcal{R}_G$), we seek a Boolean variable assignment where exactly L variables are assigned false ($e = F$ iff $e \in E_r$, and $|E_r| = L$), such that the maximal number of Boolean functions being false ($I(u, v) = F$ corresponding to (u, v) is cut by E_r).

Unfortunately, the Boolean function maximization problem is also NP-hard since it can directly express the general reachable pair cut problem. In the next section, we will introduce a numerical relation approach to solve this problem.

3. PATH ALGEBRA AND OPTIMIZATION ALGORITHM

In this section, we introduce a numerical relaxation approach to solve the Boolean function maximization problem (and thus the de-small-world problem). Here, the basic idea is that since the direct solution for the Boolean function maximization problem is hard, instead of working on the Boolean (binary) edge variable, we relax to it to be a numerical value. However, the challenge is that we need to define the numerical function optimization problem such that it meet the following two criteria: 1) it is rather accurately match the Boolean function maximization; and 2) it can enable numerical solvers to be applied to optimize the numerical function. In Subsection 3.1, we introduce the numerical optimization problem based on the path algebra. In Subsection 3.2, we discuss the optimization approach for solving this problem.

3.1 Path-Algebra and Numerical Optimization Problem

To construct a numerical optimization problem for the Boolean function maximization format of the de-small-world problem, we introduce the following path-algebra to describe all the short paths between any reachable pair in $\mathcal{R}_G$. For each edge e in the graph $G = (V, E)$, we associate it with a variable x_e. Then, for any reachable pair $(u, v) \in \mathcal{R}_G$, we define its corresponding path-algebra expression $\mathcal{P}(u, v)$ as follows:

$$\mathcal{P}(u, v) = \sum_{p \in P(u,v)} \prod_{e \in p} x_e \qquad (3)$$

Taking the path-algebra for (b, d) in Figure 2 and Figure 3 as example, we have

$$\mathcal{P}(b, d) = x_2 x_1 + x_3 x_4$$

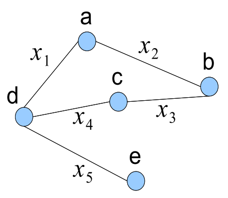


Figure 3: Algebra Variable

Basically, the path-algebra expression $\mathcal{P}(u,v)$ directly corresponds to the Boolean expression $I(u,v)$ by replacing $AND(\wedge)$ with product $(\times)$, $OR(\vee)$ with sum $(+)$, and Boolean variable e with algebraic variable x_e. Intuitively, $\mathcal{P}(u,v)$ records the weighted sum of each path in $P(u,v)$, where the weight is the product based on the edge variable x_e. Note that when $x_e = 1$ for every edge e, when $\mathcal{P}(u,v)$ simply records the number of different short paths (with length no more than k) between u and v, i.e., $\mathcal{P}(u,v) = |P(u,v)|$. Furthermore, if assuming $x_e \geq 0$, then $\mathcal{P}(u,v) = 0$ is equivalent to in each path $p \in P(u,v)$, there is at least one edge variable is equivalent to 0. In other words, assuming if variable $x_e = 0$ iff $e = T$, then $\mathcal{P}(u,v) = 0$ iff $I(u,v) = F$ and $\mathcal{P}(u,v) > 0$ iff $I(u,v) = T$.

Given this, we may be tempted to optimize the follow objective function based on the path-algebra expression to represent the Boolean function maximization problem:
$\sum_{(u,v)\in\mathcal{R}_G} \mathcal{P}(u,v)$. However, this does not accurately reflect our goal, as to minimize $\sum_{(u,v)\in\mathcal{R}_G} \mathcal{P}(u,v)$, we may not need any $\mathcal{P}(u,v) = 0$ (which shall be our main goal). This is because $\mathcal{P}(u,v)$ corresponds to the weighted sum of path products. Can we use the path-algebra to address the importance of $\mathcal{P}(u,v) = 0$ in the objective function?

We provide a positive answer to this problem by utilizing an exponential function transformation. Specifically, we introduce the following *numerical maximization* problem based on the path expression:

$$\sum_{(u,v)\in\mathcal{R}_G} e^{-\lambda\mathcal{P}(u,v)}, \text{ where}, 0 \leq x_e \leq 1, \sum x_e \geq X - L \quad (4)$$

Note that $0 \leq e^{-\lambda\mathcal{P}(u,v)} \leq 1$ (each $x_e \geq 0$), and only when $\mathcal{P}(u,v) = 0$, $e^{-\lambda\mathcal{P}(u,v)} = 1$ (the largest value for each term). When $\mathcal{P}(u,v) \approx 1$, the term $e^{-\lambda\mathcal{P}(u,v)}$ can be rather small (approach 0). The parameter λ is the adjusting parameter to help control the exponential curve and smooth the objective function. Furthermore, the summation constraint $\sum x_e \geq X - L)$ is to express the budget condition that there shall have L variables with $x_i \approx 0$. Here X is the total number of variables in the objective function ($X = |E|$ if we consider every single edge variable x_e).

3.2 Gradient Optimization

Clearly, it is very hard to find the exact (or closed form) solution for maximizing function in Equation 4 under these linear constraints. In this section, we utilize the standard *gradient* (ascent) approach together with the *active set* method [7] to discover a local maximum. The gradient ascent takes steps proportional to the positive of the gradient iteratively to approach a local minimum. The active set approach is a standard approach in optimization which deals with the *feasible regions* (represented as constraints). Here we utilize it to handle the constraint in Equation 4.

Gradient Computation: To perform gradient ascent optimization, we need compute the gradient $g(x_e)$ for each variable x_e. Fortunately, we can derive a closed form of $g(x_e)$ in $\sum_{(u,v)\in\mathcal{R}_G} e^{-\lambda\mathcal{P}(u,v)}$ as follows:

$$g(x_e) = \frac{\partial \sum_{(u,v)\in\mathcal{R}_G} e^{-\lambda\mathcal{P}(u,v)}}{\partial x_e} = \sum_{(u,v)\in\mathcal{R}_G} -\lambda\mathcal{P}(u,v,e)e^{-\lambda\mathcal{P}(u,v)},$$

where $\mathcal{P}(u,v,e)$ is the sum of the path-product on all the paths going through e and we treat $x_e = 1$ in the path-product. More precisely, let $P(u,v,e)$ be the set of all short paths (with length no more than k) between u and v going through edge e, and then,

$$\mathcal{P}(u,v,e) = \sum_{p\in P(u,v,e)} \prod_{e'\in p\setminus\{e\}} x_{e'} \quad (5)$$

Using the example in Figure 2 and Figure 3, we have

$$\mathcal{P}(b,d,e_1) = x_2$$

Note that once we have all the gradients for each edge variable x_e, then we update them accordingly,

$$x_e = x_e + \beta g(x_e),$$

where β is the step size (a very small positive real value) to control the rate of convergence.

$\mathcal{P}(u,v)$ and $\mathcal{P}(u,v,e)$ Computation To compute the gradient, we need compute all $\mathcal{P}(u,v)$ and $\mathcal{P}(u,v,e)$ for $(u,v) \in \mathcal{R}_G$. Especially, the difficulty is that even compute the total number of simple short paths (with length no more than k) between u and v, denoted as $|P(u,v)|$ is known to be expensive. In the following, we describe an efficient procedure to compute $\mathcal{P}(u,v)$ and $\mathcal{P}(u,v,e)$ efficiently. The basic idea is that we perform a DFS from each vertex u with traversal depth no more than k. During the traversal form vertex u, we maintain the partial sum of both $\mathcal{P}(u,v)$ and $\mathcal{P}(u,v,e)$ for each v and e where u can reach within k steps. After each traversal, we can then compute the exact value of $\mathcal{P}(u,*)$ and $\mathcal{P}(u,*,*)$.

Algorithm 1 ComputePUVE(G,u,k,p,w)

```
1: Input: $G = (V,E)$ and starting vertex $u$;
2: Input: spreading parameter $k$, path $p$, and partial product $w$;
3: Output: $P(u,*)$, $P(u,*,*)$;
4: if $|p| = k$ {traversal depth no more than $k$} then
5:    return
6: end if
7: $z = s.top()$ {the last visited vertex in the traveral}
8: for each $v \in Neighbor(z)$ and $v \notin p$ {simple path} do
9:    $p.push(v)$ {the current path};
10:   $w \leftarrow w \times x_{(v,z)}$ {corresponding path product};
11:   $\mathcal{P}(u,v) \leftarrow \mathcal{P}(u,v) + w$;
12:   for each $e \in p$ {every edge in the current path} do
13:      $\mathcal{P}(u,v,e) \leftarrow \mathcal{P}(u,v,e) + \frac{w}{x_e}$;
14:   end for
15:   ComputePUVE$(G,u,k,p,w)$;
16:   $p.pop()$; $w \leftarrow \frac{w}{x_{(v,z)}}$;
17: end for
```

The DFS procedure starting from u to compute all $\mathcal{P}(u,*)$ and $\mathcal{P}(u,*,*)$ is illustrated in Algorithm 1. In Algorithm 1, we maintain the current path (based on the DFS traversal procedure) in p and its corresponding product $\sum_{e\in p} x_e$ is maintained in variable w (Line 9 and 10). Then, we incrementally update $\mathcal{P}(u,v)$ assuming v is the end of the path p (Line 11). In addition, we go over each edge in the current path, and incrementally update $\mathcal{P}(u,v)$ ($w/x_e = \prod_{e'\in p\setminus\{e\}} x_{e'}$, Line 13.) Note that we need invoke this procedure for every vertex u to compute all $\mathcal{P}(u,v)$ and $\mathcal{P}(u,v,e)$. Thus, the overall time complexity can be written as $O(|V|\overline{d}^k)$ for a random graph where $\overline{d}$ is the average vertex degree.

Overall Gradient Algorithm

The overall gradient optimization algorithm is depicted in Algorithm 2. Here, we use $\mathcal{C}$ to describe all the edges which need be processed for optimization. At this point, we consider all the edges and thus $\mathcal{C} = E$. Later, we will consider to first select some candidate edges. The entire algorithm performs iteratively and each iteration has three major steps:

Step 1 (Lines $6-8$): it calculates the gradient $g(x_e)$ of for every edge variable x_e and an average gradient $\overline{g}$;

Algorithm 2 OptimizationAlg(G, L)

```
1: Input: $G = (V, E)$, and edge removal budget $L$;
2: Output: edge set $E_r$;
3: $\forall e \in \mathcal{C}$ ($\mathcal{C} = E$), $x_e \leftarrow 1$; {initialization}
4: $\mathcal{A} \leftarrow \emptyset$; {active set}
5: while NOT every $x_e$ converges do
6:    $\forall x_e$, calculate $\mathcal{P}(u, v)$ and $\mathcal{P}(u, v, e)$ using Algorithm 1;
7:    $\forall x_e$, $g(x_e) \leftarrow -\lambda \sum_{T(x_e)} e^{-\lambda \mathcal{P}(u,v)} \mathcal{P}(u, v, e)$;
8:    $\overline{g} \leftarrow \frac{\sum_{x_e \in \mathcal{C} \setminus \mathcal{A}} g(x_e)}{|\mathcal{C} \setminus \mathcal{A}|}$ {average gradient};
9:    for each $e \in \mathcal{C} \setminus \mathcal{A}$ do
10:       if bound reached ($\sum_{e \in \mathcal{C}} x_e < |\mathcal{C}| - L$) {using $x_e$ from
          last iteration} then
11:          $x_e \leftarrow max(x_e + \beta(g(x_e) - \overline{g}), 0)$;
12:       else
13:          $x_e \leftarrow max(x_e + \beta g(x_e), 0)$;
14:       end if
15:       $x_e \leftarrow min(x_e, 1)$;
16:    end for
17:    for each $e \in \mathcal{C} \setminus \mathcal{A}$ and ($x_e = 1$ or $x_e = 0$) do
18:       $\mathcal{A} \leftarrow \mathcal{A} \cup \{e\}$; {add to active set}
19:    end for
20:    for each $e \in \mathcal{A}$ and (($x_e = 0 \wedge g(x_e) <= \overline{g}$) $\vee$ ($x_e =$
       $1 \wedge g(e) >= \overline{g}$)) do
21:       $\mathcal{A} \leftarrow \mathcal{A} \backslash \{e\}$; {remove from active set}
22:    end for
23: end while
24: sort all $\{x_e\}$ in increasing order, and add top $L$ edges to $E_r$;
```

Step 2 (Lines $9 - 16$): only those variables are not in the active set $\mathcal{A}$ will be updated. Specifically, if the condition ($\sum_{e \in E} x_e \geq |E| - L$) is not met, we try to adjust x_e back to the feasible region. Note that by using $g(x_e) - \overline{g}$ (Line 11) instead of $g(x_e)$ (Line 13), we are able to increase the value of those x_e whose gradient is below average. However, such adjustment can still guarantee the overall objective function is not decreased (thus will converge). Also, we make sure x_e will be between 0 and 1.
Step 3 (Lines $17 - 22$): the active set is updated. When an edge variable reaches 0 or 1, we put them in the active set so that we will not need to update them in Step 2. However, for those edges variables in the active set, if their gradients are less (higher) than the average gradient for $x_e = 0$ ($x_e = 1$), we will release them from the active set and let them to be further updated.

Note that the gradient ascent with the active set method guarantees the convergence of the algorithm (mainly because the overall objective function is not decreased). However, we note that in Algorithm 2, the bounded condition ($\sum_{e \in E} x_e \geq |E| - L$) may not be necessarily satisfied even with the update in Line 11. Though this can be achieved through additional adjustment, we do not consider them mainly due to the goal here is not to find the exact optimization, but mainly on identifying the smallest L edges based on x_e. Finally, the overall time complexity of the optimization algorithm is $O(t(|V| * \overline{d}^k + |E|))$, given t is the maximum number of iterations before convergence.

4. SHORT BETWEENNESS AND SPEEDUP TECHNIQUES

In Section 3, we reformulate our problem into a numerical optimization problem. We further develop an iterative *gradient* algorithm to select the top L edges in to E_r. However, the basic algorithm can not scale well to very large graphs due to the large number ($|E|$) of variables involved. In this section, we introduce a new variant of the edge-betweenness and use it to quickly reduce the variables needed in the optimization algorithm (Algorithm 2). In addition, we can further speedup the DFS procedure to compute $\mathcal{P}(u, v)$ and $\mathcal{P}(u, v, e)$ in Algorithm 1.

4.1 Short Betweenness

In this subsection, we consider the following question: *What edge importance measure can directly correlate with x_e in the objective function in Eq. 4 so that we can use it to help quickly identify a candidate edge set for the numerical optimization described in Algorithm 2*? In this work, we propose a new edge-betweenness measure, referred to as the *short betweenness* to address the this question. It is intuitively simple and has an interestingly relationship with respect to the gradient $g(x_e)$ for each edge variable. It can even be directly applied for selecting E_r using the generic procedure in Section 2 and is much more effective compared with the global and local edge-betweenness which measure the edge importance in terms of the shortest path (See comparison in Section 5).

Here we formally define $\nabla(e_i)$ as *short betweenness*.

DEFINITION 4. **(Short Betweenness:)** *The short betweenness $SB(e)$ for edge e is as follows, $SB(e) = \sum_{(u,v) \in \mathcal{R}_G} \frac{|P(u,v,e)|}{|P(u,v)|}$.*

Recall that $(u, v) \in \mathcal{R}_G$ means $d(u, v) \leq k$; $|P(u, v)|$ is the number of short paths between u and v; and $|P(u, v, e)|$ is the number of short paths between u and v which must go through edge e. The following lemma highlights the relationship between the short betweenness and the gradient of edge variable x_e:

LEMMA 2. *Assuming for all edge variables $x_e = 1$, then $g(x_e) \geq -SB(e)$.*

Proof Sketch:

$$\begin{aligned} g(e) &= \sum_{(u,v) \in \mathcal{G}_R} -\lambda \mathcal{P}(u, v, e) e^{-\lambda \mathcal{P}(u,v)} \\ &= \sum_{(u,v) \in \mathcal{G}_R} -\lambda |P(u, v, e)| e^{-\lambda |P(u,v)|} (\forall e, x_e = 1) \\ &\geq \sum_{(u,v) \in \mathcal{G}_R} \frac{-\lambda |P(u, v, e)|}{\lambda |P(u, v)|} (e^{-x} < 1/x, x > 0) \\ &= -SB(e) \end{aligned}$$

□

Basically, when $x_e = 1$ for every edge variable x_e (this is also the initialization of Algorithm 2), the (negative) short betweenness serves a lower bound of the gradient $g(e)$. Especially, since the gradient is negative, the higher the gradient of $|g(e)|$ is, the more likely it can maximize the objective function (cut more reachable pairs in $\mathcal{R}_G$. Here, the short betweenness $SB(e)$ thus provide an upper bound (or approximation) on $|g(e)|$ (assuming all other edges are presented in the graph); and measures the the edge potential in removing those local reachable pairs. Finally, we note that Algorithm 1 can be utilized to compute $|P(u, v)|$ and $|P(u, v, e)|$, and thus the short betweenness (just assuming $x_e = 1$ for all edge variables).
Scaling Optimization using Short Betweenness: First, we can directly utilize the short betweenness to help us pickup a candidate set of edge variables, and then Algorithm 2 only need to work on these edge variables (considering other edge variables are set as 1). Basically, we can choose a subset of edges E_s which has the highest short betweenness in the entire graph. The size of E_s has to be larger than L; in general, we can assume $|E_s| = \alpha L$, where

$\alpha > 1$. In the experimental evaluation (Section 5), we found when $\alpha = 5$, the performance of using candidate set is almost as good as the original algorithm which uses the entire edge variables. Once the candidate set edge set is selected, we make the following simple observation:

LEMMA 3. *Given a candidate edge set $E_s \subseteq E$, if any reachable pair $(u, v) \in \mathcal{R}_G$ can be cut by E_r where $E_r \subseteq E_s$ and $|E_r| = L$, then, each path in $P(u, v)$ must contains at least one edge in E_s.*

Clearly, if there is one path in $P(u, v)$ does not contain an edge in E_s, it will always linked no matter how we select E_r and thus cannot cut by $E_r \subseteq E_s$. In other words, (u, v) has to be cut by E_s if it can be cut by E_r. Given this, we introduce $\mathcal{R}_s = \mathcal{R}_G \subseteq \mathcal{R}_{G \setminus E_s}$. Note that $\mathcal{R}_s$ can be easily computed by the DFS traversal procedure similar to Algorithm 1. Thus, we can focus on optimizing

$$\sum_{(u,v)\in\mathcal{R}_s} e^{-\lambda \mathcal{P}(u,v)}, \text{ where}, 0 \le x_e \le 1, \sum x_e \ge X - L \qquad (6)$$

Furthermore, let $E_P = \bigcup_{(u,v)\in\mathcal{R}_s} \bigcup_{p \in P(u,v)} p$, which records those edges appearing in certain path linking a reachable pair cut by E_P. Clearly, for those edges in $E \setminus E_P$, we can simply prune them from the original graph G without affecting the final results. To sum, the short betweenness measure can help speed up the numerical optimization process by reducing the number of edge variables and pruning non-essential edges from the original graph.

5. EXPERIMENTAL STUDY

In this section, we report the results of the empirical study of our methods. Specifically, we are interested in the performance (in terms of reachable pair cut) and the efficiency (running time).

Performance: Given a set of edges E_r with budget L, the total number of reachable pairs being cut by E_r is $|\mathcal{R}_G| - |\mathcal{R}_{G\setminus E_r}|$ or simply $\Delta|\mathcal{R}_G|$. We use the average pair being cut by an edge, i.e., $\delta = \frac{\Delta|\mathcal{R}_G|}{L}$ as the performance measure.

Efficiency: The running time of different algorithms.

Methods: Here we compare the following methods:

1) *Betweenness based method*, which is defined in terms of the shortest paths between any two vertices in the whole graph G; hereafter, we use BT to denote the method based on this criterion.

2) *Local Betweenness based method* , which, compared with betweenness method(BT), takes only the vertex pair within certain distance into consideration; hereafter, we use LB to stand for the method based on local betweenness.

3) *Short Betweenness based method*, the new betweenness introduced in this paper, which considers all short paths whose length is no more than certain threshold. Here we denote the method based on short betweenness as SB.

4) *Numerical Optimization method*, which solves the de-small-world problem iteratively by calculating gradients and updating the edge variables x_e. Based on whether the method use the candidate set or not, we have two versions of optimization methods: OMW (Optimization Method With candidate set) and OMO (Optimization Method withOut candidate set). Note that we normally choose the top $5L$ edges as our candidate set.

As mentioned before in Section 2, we have a generic procedure to select L edges depending on parameter r (batch size). We found for different methods BT, LB and SB, the effects of r seem to be rather small (as illustrated in Figure **??**). Thus, in the reminder of the experiments, we choose $r = L$, i.e., we select the top L edges using the betweenness calculated for the entire (original graph).

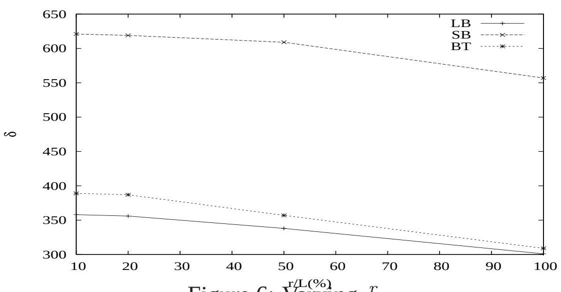


Figure 6: Varying $\frac{r}{L}$

All the algorithms are implemented using C++ and the Standard Template Library (STL), and the experiments are conducted on a 2.0GHz Dual Core AMD Opteron CPU with 4.0GB RAM running on Linux.

5.1 Result on Synthetic Datasets

In this subsection, we study the performance and efficiency of different methods on the synthetic datasets. Here, we generate various synthetic networks from two well-known small-world models: *Watts and Strogatz model* (WS model) [19] generating small-world graphs by interpolating between ER graph and a regular ring lattice; the small-world model proposed by Kleinberg [10] (KS model). Then, the networks generated from WS model, KS model are referred to as the WS network and KS network, respectively.

In the following, we conducted three groups of experiments.

Varying $|V|$: In this group of experiments, we generate networks respectively with the two models (WS, KS) using vertex size $1k$, $5k$ and $10k$. We also set the edge budget $L = 1000$ (edge removal budget) and $k = 3$ (spreading parameter). The results are summarized in Figure 10(a) and 10(e). From these two figures, we can see that LB method always produces the worst result (its δ is around 100, meaning each edge on average contribute to around 100 reachable pairs). Meantime, δ for BT method increases dramatically from 150 to 300 for both KS and WS graphs. Comparatively, OMW always reduces the biggest number of pairs compared with other methods. More specifically, its δ grows from 175 to more than 400. Meanwhile, SB method produces the similar result as OMW method. This suggests the power of short betweenness (which directly forms an upper bound for the absolute gradient $g(x_e)$).

Varying L: In this group of experiments, we study the reduction effect for different L and the result for KS model is reported in Figure 10(b). Generally, with the increase of L, δ decreases. This is reasonable because more reachable pairs is removed, each edge can remove the smaller number of reachable pairs. For the specific algorithms, similar to the situation in last group of experiment, LB and OMW methods produce the lowest δ and highest δ, respectively. Then the number of reduced reachable pairs by BT method is about three times that of LB, and is about $\frac{3}{4}$ of that reduced by SB and OMW. These cases also happen for the graphs generated by WS model as in Figure 10(f).

Varying k: Remember that we define the short path as the paths with length at most k. Given G, obviously k determines the size of reachable pairs. Given different k, the result of all algorithms are reported in Figure 10(c) for KS model graphs. We can see that generally, with the increase of k, the strength of each edge(δ) increases. This is understandable because with k increasing, each edge could effect more reachable pairs. For the specific algorithms, LB produces the lowest δ for all k. Then other three methods produce similar δ, which are normally about four times between than LB. The similar situation happens for WS graphs as in Figure

Figure 4: Network Statistics

Dataset	$\|V\|$	$\|E\|$	π
Gnutella04	10,876	39,994	9
Gnutella05	8,846	31,839	9
Gnutella06	8,717	31,525	9
Gnutella08	6,301	20,777	9
Gnutella09	8,114	26,013	9
Gnutella24	26,518	65,369	10
Gnutella25	22,687	54,705	11
Gnutella30	36,682	88,328	10
Gnutella31	62,586	147,892	11

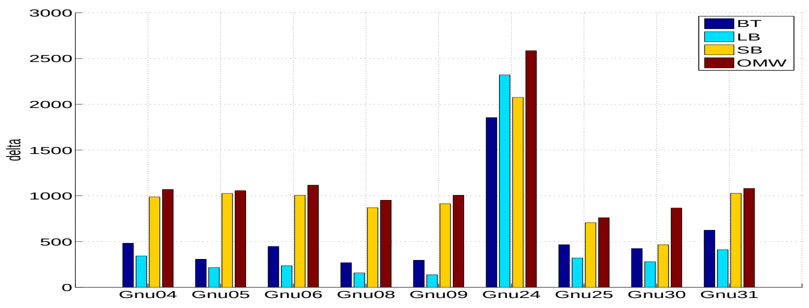


Figure 5: δ for all real datasets

10(g).

5.2 Result on Real Datasets

In this subsection, we study the performance of our algorithms on real datasets. The benchmarking datasets are listed in Figure 4. All networks contain certain properties commonly observed in social networks, such as small diameter. All datasets are downloadable from Stanford Large Network Dataset Collection [1].

In Figure 4, we present important characteristics of all real datasets, where π is graph diamter. All these nine networks are snapshots of the Gnutella peer to peer file sharing network starting from August 2002. Nodes stand for the hosts in the Gnutella network topology and the edges for the connections between the hosts.

Varying L: We perform this group of experiments on dataset $Gnu05$ and we fix $k = 3$. Here we run these methods on three different edge buget L: 500, 1000 and 2000. The result is reported in Table 8. The general trend is that with smaller L, δ becomes bigger. This is because the set of reachable pairs removed by different edges could have intersection; when one edge is removed, the set of reachable pairs for other edges is also reduced. For particular methods, BT and OMO methods produces the lowest and highest δ, and the different between OMW and OMO is very small.

Varying k: In this group of experiments, we fix $L = 1000$ and we choose $Gnu04$. Here we choose three values for k: 2, 3 and 4. The result is reported in Table 9. From the result, we can see that when k becomes bigger, δ become higher. This is also reasonable:when k becomes bigger, more reachable pairs are generated and meanwhile $|E|$ is constant; therefore, each edge is potentially able to remove more reachable pairs. From the above three groups of experiments, we can see that OMO does not produce significant results compared with OMW. Therefore, in the following experiment, we do not study OMO method again.

δ on all real datasets: In this groups of experiment, we study the performance of each method on these nine datasets, with L being proportional to $|E|$. Specifically, $L = |E| \times 1\%$. We report the result in Figure **??**. LB generally produces the lowest δ, around half that of BT; and also the best method, is the SB and OMW methods. Specifically, OMW is always slightly better than SB.

6. CONCLUSION

In this paper, we introduce the *de-small-world* network problem; to solve it, we first present a greedy edge betweenness based approach as the benchmark and then provide a numerical relaxation approach to slove our problem using OR-AND boolean format, which can find a local minimum. In addition, we introduce the *short-betweenness* to speed up our algorithm. The empirical study demonstrates the efficiency and effectiveness of our approaches. In the future, we plan to utilize MapReduce framework(e.g. Hadoop) to scale our methods to handle graphs with tens of millions of vertices.

[1]http://snap.stanford.edu/data/index.html

7. REFERENCES

[1] R. Andersen, F. Chung, and L. Lu. Modeling the small-world phenomenon with local network flow. *Internet Mathematics*, pages 359–385, 2005.

[2] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[3] C. Budak, D. Agrawal, and A. El Abbadi. Limiting the spread of misinformation in social networks. In *Proceedings of the 20th international conference on World wide web*, WWW '11, 2011.

[4] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29:2001, 1999.

[5] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002.

[6] S. Gregory. Local betweenness for finding communities in networks. Technical report, University of Bristol, February 2008.

[7] W. W. Hager and H. Zhang. A new active set algorithm for box constrained optimization. *SIAM J. on Optimization*, 17:526–557, August 2006.

[8] S. Jin and A. Bestavros. Small-world characteristics of internet topologies and implications on multicast scaling. *Comput. Netw.*, 50, April 2006.

[9] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 137–146, 2003.

[10] J. Kleinberg. The Small-World phenomenon: An algorithmic perspective. In *32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.

[11] J. Leskovec and E. Horvitz. Planetary-scale views on a large instant-messaging network. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, 2008.

[12] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 420–429, 2007.

[13] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.

[14] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, 2007.

[15] C. Moore and M. E. J. Newman. Epidemics and percolation in small-world networks. *Physical Review E*, pages 5678–5682, 2000.

[16] M. E. J. Newman. Spread of epidemic disease on networks. *Physical Review E*, 66:016128+, 2002.

[17] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Phys. Rev. Lett.*, pages 3200–3203, 2001.

Figure 7: Running Time (Seconds)

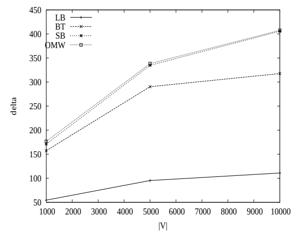
Time	BT	LB	SB	OMW
10,876	382.27	24.82	33.75	1021.66
8,846	21346.54	496.17	8.98	110.80
62586	392.54	25.31	34.60	1092.55

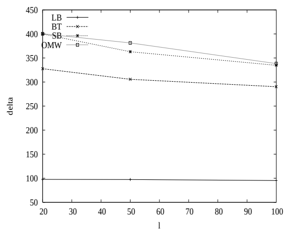
Figure 8: δ By Varying l

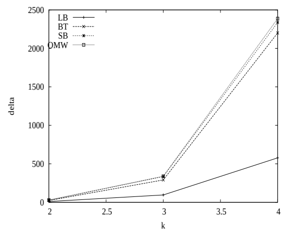
l	BT	LB	SB	OMW	OMO
500	240	415	912	996	973
1000	261	372	740	803	805
2000	301	329	572	620	622

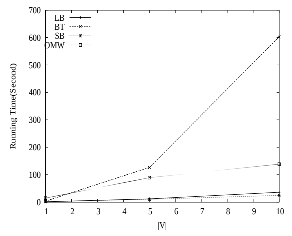
Figure 9: δ By Varying k

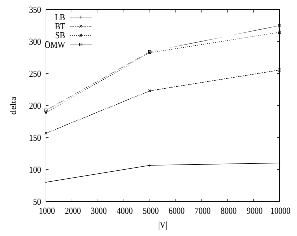
k	BT	LB	SB	OMW	OMO
2	25	32	55	58	58
3	261	372	740	803	805
4	761	976	2113	2389	-

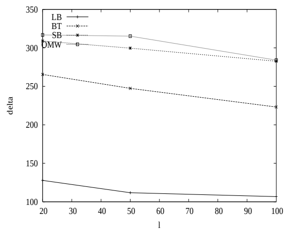

(a) Varying $|V|$ for KS

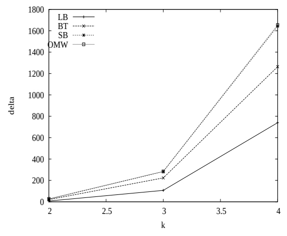

(b) Varying L for KS


(c) Varying k for KS


(d) Running Time for KS


(e) Varying $|V|$ for WS


(f) Varying L for WS


(g) Varying k for WS

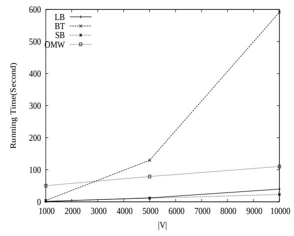

(h) Running Time for KS

Figure 10: Experiments on Synthetic Datasets

[18] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 61–70, 2002.

[19] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, pages 440–442, 1998.

APPENDIX

A. PROOF OF THEOREM 1

To prove this theorem, we first introduce the *dense κ-sub-hypergraph* problem. Let $H = (V_H, E_H)$ be a hypergraph, where V_H is the vertex set, and E_H is the hyperedge set, such that $e_h \in E_h$ and $e_h \subseteq V$. Each hyperedge in the hypergraph is a subset of vertices (not necessarily a pair as in the general graph). Furthermore, given a subset of vertices $V_s \subseteq V_H$, if an hypergraph $e_h \subseteq V_S$, then, we say this hyperedge is *covered* by the vertex subset V_s.

DEFINITION 5. **Dense κ-Sub-Hypergraph Problem:** *Given hypergraph $G = (V_H, E_H)$ and a parameter κ, we seek to find a subset of vertices $V_s \subseteq V_H$ and $|V_s| = \kappa$, such that the maximal number of hyperedges in E_H is covered by V_s.*

Dense κ-hyper-subgraph problem can be easily proven to be NP-Hard, because its special case, dense κ-subgraph problem (each edge $e \subset V \times V$) has been shown to be NP-Hard [4].

Proof Sketch: To reduce the Dense κ-Sub-Hypergraph problem to our problem, we construct the following graph from the given hypergraph $H = (V_H, E_H)$. Figure11 illustrates the transformation. For each hyperedge $e_i \in E_H$, which consists a set $p_i = \{v_{i_1}, v_{i_2}, \cdots, v_{i_k}\}$ of vertices in V_H, we represent it as a vertex pair p_i^s and p_i^t in the graph G, and each pair is connected by i_k different paths with length 3, where each middle edge in G corresponds to a unique vertex in V_H. For instance for hyperedge $p_1 = \{a, b, c\}$, the middle edges of the three-paths in G correspond to edges a, b, c. To facilitate our discussion, the middle edges of these length-3 paths linking the vertex pairs corresponding to each hypergraph in H are referred to MS, which has one-to-one correspondence to the vertex set V_H in the hypergraph.

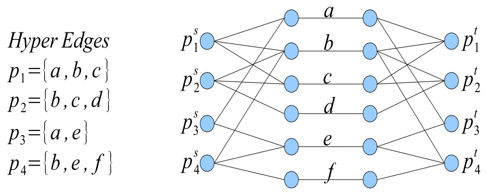


Figure 11: Graph for Reachable Pair cut Problem

Now we show for the dense κ sub-hypergraph problem, its optimal solution can be solved by an instance of the general reachable pair cut problem, where $L = \kappa$ and RS consists of all the (p_i^s, p_i^e) reachable pairs ($k = 3$). In other worlds, $|RS| = |E_H|$. Specifically, we need show that if a subset of edges S ($|S| = L$) in MS can maximally disconnect the reachable pairs in RS, then its corresponding vertex subset V_S can maximally cover the hyperedges. This is easy to observe due to the one-to-one correspondence relationship between MS and V_H and between RS and E_H. Given this, we need show that the optimal solution of the reachable pair cut problem can be always found using only edges in MS. Suppose the edge set ES' with size L is the optimal solution which contain some edge $e \in ES$, and $e \notin MS$. In this case, we can simply replace e with its adjacent middle edge e' from MS in the result set. This is because the replacement $ES' \backslash \{e\} \cup \{e'\}$ will still be able to disconnect all the reachable pairs in RS being cut by ES'. Note that the middle edge e' can cut more paths than e (form a superset of the paths cut by e) with respect to RS. □