Refining Social Graph Connectivity via Shortcut Edge Addition

MANOS PAPAGELIS, University of Toronto

Small changes on the structure of a graph can have a dramatic effect on its connectivity. While in the traditional graph theory, the focus is on well-defined properties of graph connectivity, such as biconnectivity, in the context of a *social graph*, connectivity is typically manifested by its ability to carry on *social processes*. In this paper, we consider the problem of adding a small set of nonexisting edges (*shortcuts*) in a social graph with the main objective of minimizing its *characteristic path length*. This property determines the average distance between pairs of vertices and essentially controls how broadly information can propagate through a network. We formally define the problem of interest, characterize its hardness and propose a novel method, *path screening*, which quickly identifies important shortcuts to guide the augmentation of the graph. We devise a sampling-based variant of our method that can scale up the computation in larger graphs. The claims of our methods are formally validated. Through experiments on real and synthetic data, we demonstrate that our methods are *a multitude of times* faster than standard approaches, their accuracy outperforms sensible baselines and they can ease the spread of information in a network, for a varying range of conditions.

Categories and Subject Descriptors: H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Information networks*; H.2.8 [**Database Management**]: Database Applications—*Data mining*; J.4 [**Computer Applications**]: Social and Behavioral Science

General Terms: Algorithms, Performance, Human Factors

Additional Key Words and Phrases: Graph augmentation, social networks, network engineering, shortcuts, propagation, mixing time, conductance

ACM Reference Format:
Manos Papagelis. 2015. Refining social graph connectivity via shortcut edge addition. ACM Trans. Knowl. Discov. Data 10, 2, Article 12 (October 2015), 35 pages.
DOI: http://dx.doi.org/10.1145/2757281

1. INTRODUCTION

Group formation and segregation, the rise and fall of fashion fads, the adoption or rejection of an innovation are all manifestations of fundamental *social processes*, such as homophily, influence, and contagion, that now take place in online social networks and can be monitored and examined through traces of online social interactions. The ability of a social network to carry on such social processes is a characteristic that depends, to a great extent, on the *topology of the network*. Consider the process of information propagation; as individuals become aware of new or interesting pieces of information they have the chance to pass them forward to their friends, and so on and so forth. The number of nodes in the network that can be reached during the propagation depends on *contextual properties* of the propagation itself (e.g., the content of the information being transmitted, how susceptible nodes are in receiving

Author's address: M. Papagelis, Computer Science Department, University of Toronto, 40 St. George Street, Toronto ON M5S 2E4, Canada; email: papaggel@cs.toronto.edu.

DOI: http://dx.doi.org/10.1145/2757281

the information, how willing they are to further transmit it, and so on), but there is also a genuine connection between the propagation process and the topology of the network. It becomes obvious that slight modifications in the network topology, might have a dramatic effect on its connectivity and thus to its capacity to carry on such social processes.

In this paper, we approach network modification as a *graph augmentation problem*, in which we ask to find a small set of nonexisting edges (from now on we refer to them as *shortcut edges* or simply *shortcuts*) to add to the network that will optimize a connectivity property. While in the traditional graph theory, network connectivity typically refers to a well-defined property, such as biconnectivity, in the context of a social graph, connectivity is usually manifested by the ability of a network to bring users closer to each other and drive up user engagement. Having plenty of choices for measuring the level of connectivity in the social graph, we focus on a structural feature and try to minimize the *characteristic path length* of the network. This property determines the average distance between pairs of vertices and essentially controls the evolution of social processes in the network; for example, it controls how efficiently information can propagate through a network. In the augmented network, information propagates easier and faster, as it has to travel from a node to another by traversing fewer edges of a more connected network. It is also interesting to note that while in traditional graph augmentation problems the addition of edges can occur anywhere in the network, in the context of a social graph, augmentation might need to respect constraints and *norms of social activity*. For example, it might make sense to connect two social peers (nodes) if only they share a contact (an adjacent node); we further elaborate on this issue when we formally define the problem.

There are many possible application scenarios of this work. For example, suggesting friends in a social network with the global objective of enabling efficient information dissemination or performing faster network simulations by reducing the convergence time of random walk processes or boosting collaboration in scientific networks by connecting the right peers. In the next section, we provide more details about a few of these applications to further motivate our research.

1.1. Motivating Applications

We describe two applications, where a slightly modified network topology is preferred, to motivate the problem of interest.

1.1.1. Boosting Information Propagation Processes. A longstanding interesting topic of research in network analysis has been to identify community-like structure in complex networks (such as social, biological, or information networks). Community structure (typically referred to as a module or cluster) refers to the occurrence of groups of nodes in a network that are more densely connected internally than with the rest of the network. Identifying communities is interesting because communities are often interpreted as organizational units in social networks. However, there are cases where evidence of tightly knit communities in a network topology might constrain the evolution of processes that take place or unfold in it. This is the case of the process by which information flows in a social network. A *critical problem* of this social process' evolution is that while information spreads effectively among members of the same community (intracluster), it can eventually stall when it tries to break into other communities (intercluster). In fact, as Easley and Kleinberg [2010] state:

> Clusters and cascades (the outcome of information spread) can be seen as natural opposites, in the sense that clusters block the spread of cascades, and whenever a cascade comes to a stop, there is a cluster that can be used to explain why.

In this research, we are interested in exploring how small changes in a network's topology, manifested by the addition of a few shortcuts, might alleviate its community structure, giving rise to more efficient information propagation processes.

1.1.2. Boosting Random Walks on Graphs. Graph sampling has been a common strategy to efficiently compute interesting metrics in a large graph [Leskovec and Faloutsos 2006] or a large social network [Papagelis et al. 2013]. The objective of sampling is to randomly select elements (nodes or edges) according to a predetermined probability distribution, and then to generate aggregate estimations based on the retrieved samples. Many of the graph sampling techniques are built upon the idea of performing random walks over the graph. One of the most popular schemes is the simple random walk (SRW) [Lovász 1993], in which a walk starts from an arbitrary node and in each step picks the next node to visit uniformly at random from the adjacent nodes. The last node to be visited by the walk forms a node of the sample. If the random walk is sufficiently long, then the probability of each node to be in the sample tends to reach a stationary (probability) distribution proportional to the degree of each node (the number of connections it has to other nodes). Based on the retrieved samples and knowledge of such a stationary distribution, one can generate unbiased estimations of aggregates over all nodes in the network.

A *critical problem* of that sampling method is that a random walk might require to take a very large number of steps in order to retrieve a single sample. The minimal length of a random walk in order to reach the stationary distribution usually appears in the literature as the *mixing time* or *mixing rate* of a random walk or Markov chain (i.e., how fast the random walk converges to its stationary distribution). Studies on the graph theory suggest that the mixing time is tightly related to the connectivity of the graph. In particular, strongly connected graphs are known to have small mixing time (fast convergence), while weakly connected graphs are known to have large mixing time (slow convergence). How "well-knit" a graph is, is usually determined by *the conductance of a graph*. It is known that graphs with large conductance have a small mixing time. Therefore, the smaller the conductance of a graph is, the longer the random walk of the sampling method will be.

In this research, we are interested in exploring how small changes in a network's topology, manifested by the addition of a few shortcuts, might increase the conductance of the graph and dramatically improve the performance of graph sampling-based simulations, driven by faster random walks.

1.2. Contributions

The paper makes the following contributions.

—We formalize the problem of finding a set of shortcuts to add in a social graph that will *shrink* the network as much as possible and formally characterize its hardness.
—We propose a novel method (*path screening*) for efficiently evaluating the utility of adding shortcuts in a network.
—We present a *sampling-based variant* of the proposed method that improves its performance and extents its applicability to larger graphs.
—We formally validate the claims of our methods.
—We evaluate the accuracy and efficiency of our methods using real and synthetic data and show that they (i) outperform sensible baselines, (ii) ease the spread of information in networks, and (iii) speed up the convergence of random walk-based simulations in graphs, by *a multitude of times* and for *a varying range of conditions*.

1.3. Paper Organization

The rest of the paper is organized as follows. Section 2 introduces notation and presents background on the graph theory that is required to understand the paper. Section 3 formally defines the problem of interest and characterizes its hardness. Some theoretical results of the problem are presented in Section 4. In particular, we show that in a graph, the sum of the lengths of all-pairs shortest paths is equal to the sum of the edge betweenness of all its edges. We also show that the function of adding shortcuts in a graph in order to minimize its characteristic path length, while being monotonic, it is not submodular; therefore efficient techniques that rely on the submodularity assumption cannot be employed to solve the optimization problem. Section 5 presents efficient heuristic methods for fast assessment of the importance of a shortcut in a network. We experimentally evaluate our methods in Section 6. In Section 7, we review related work and conclude in Section 8.

2. PRELIMINARIES

2.1. Network Model

Let $G(V, E)$ be a connected undirected simple graph with V the set of vertices and E the set of edges, where $|V| = n$ and $|E| = m$. Let $d(u, v)$ be the length of the shortest path between the vertices u and v. The sum of all-pairs shortest path lengths is $L = \sum_{(u,v)\in V\times V} d(u, v)$. Then, the *characteristic path length*, $\bar{L}$, is defined as the average shortest path length over all pairs of vertices in the graph ($\bar{L} = L/(\binom{n}{2})$). $\bar{L}$ is a measure of the efficiency of information transport on a network. It is also a measure of how close users are to each other, which encourages social interaction and drives up user engagement. The *range* of an edge $R(u, v)$ is the length of a shortest path between u and v in the absence of that edge. An edge with $R(u, v) \geq 2$ is called a *shortcut*.

2.2. SRW on a Graph

We recall a few definitions of random walks on graphs. For more detailed exposition, see [Sinclair 1992]. We define the degree of a node $v \in V$ as the number of nodes in V adjacent to v and denote it by $deg(v)$. We also define the neighborhood of a node v to be $N(v) = \{u : d(u, v) = 1\}$ (so $v \notin N(v)$). A SRW on a graph G is a Markov Chain Monte Carlo method that takes successive random steps according to a transition probability matrix $P = [p_{vu}]$ of size $n \times n$ where the $(v, u)^{th}$ entry in P is the probability of moving from node v to node u defined as

$$p_{vu} = \begin{cases} \frac{1}{deg(v)}, & \text{if } u \in N(v) \\ 0, & \text{otherwise.} \end{cases}$$

At any given iteration t of the random walk, let us denote with $\pi(t)$ the probability distribution of the random walk state at that iteration ($\pi(t)$ is a vector of n entries). The state distribution after t iterations is given by $\pi(t) = \pi(0) \cdot P^t$, where $\pi(0)$ is the initial state distribution. The probability of a t steps random walk starting from v and ending at u is given by P^t_{vu}. For irreducible and aperiodic graphs (which is the case of undirected connected social graphs), the corresponding Markov chain is said to be ergodic. In that case, the stationary distribution π of the Markov chain (the distribution after a random walk of length μ, as $\mu \to \infty$.) is a probability distribution that is invariant to the transition matrix P (i.e., satisfies $\pi = \pi \cdot P$). For an undirected unweighted connected graph G, the stationary distribution of the Markov chain over G is the probability vector $\pi = [\pi_v]$, where $\pi_v = deg(v)/2|E|$.

2.2.1. Mixing Time. The mixing time of a Markov process is a measure of the minimal length of the random walk in order to reach the stationary distribution. Let $\Delta(t)$ be

the relative point-wise distance between the current sampling distribution and the stationary distribution, after t steps of SRW:

$$\Delta(t) = \max_{v,u \in V, u \in N(v)} \left\{ \frac{|P^t_{vu} - \pi_u|}{\pi_u} \right\}$$

where P^t_{vu} is the element of P^t with indices v and u.

Definition 2.1 (*Mixing Time of a Graph*). The *mixing time* of a graph is the minimum number of steps t required by a random walk to converge; that is, $\Delta(t) \leq \epsilon$, where ϵ is a threshold.

Note that social graphs have been shown to have relatively larger mixing time than what it was usually anticipated [Mohaisen et al. 2010].

2.3. Community Structure and Conductance

A common characteristic in the study of complex networks is *community structure*. The quality of a community is commonly defined using modularity or flow-based measures. We adopt a more intuitive community quality concept, *conductance*.

Definition 2.2 (*Conductance of a Set of Nodes*). Let $G = (V, E)$ be an undirected graph and $C \subset V$ be a set of nodes (C can be thought of as a cluster or community). Then, the conductance $\Phi(C, \bar{C})$ (or simply $\Phi(C)$) is given by:

$$\Phi(C) = \Phi(C, \bar{C}) = \frac{|E_{C,\bar{C}}|}{Vol(C)}$$

where

$$\bar{C} = V - C$$
$$E_{C,\bar{C}} = \{(v, u) \in E | v \in C, u \in \bar{C}\}$$
$$Vol(C) = \sum_{v \in C} \deg(v).$$

The notion of conductance $\Phi(C, \bar{C})$ of a set of nodes C may be thought of as the ratio between the number of connections pointing outside the community C (to nodes in $\bar{C}$) and the number of connections inside C. It is widely used to capture quantitatively the notion of a good network community as a set of nodes that has better internal connectivity than external connectivity [Leskovec et al. 2008]. Note that more community-like sets of nodes have lower conductance (Figure 1).

Using conductance as a measure of network community quality, one can define the *conductance of a graph*.

Definition 2.3 (*Conductance of a Graph*). Let $G = (V, E)$ be an undirected graph. The conductance Φ_G of G is given by:

$$\Phi_G = \min_{|C| \leq |V|/2} \Phi(C, \bar{C}).$$

2.3.1. Graph Partitioning. The conductance Φ_G of a graph is intractable to compute exactly (*NP-hard*), since we need to search over all possible cuts of a graph (of any community size $|C|$) and find the one with minimum conductance. Therefore, we have to resort to efficient approximation methods for computing partitions of graphs with minimum conductance over all the possible cuts. This is the *graph partitioning* problem, in which the goal is to find groups of nodes (clusters) in a graph G, with roughly equal size, such that the number of edges between the groups is minimized and a merit

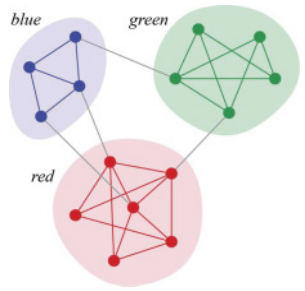


Fig. 1. Community structure in complex networks. For example, $\Phi(blue) = 3/13$, $\Phi(green) = 2/18$, $\Phi(red) = 3/25$. The set of *green* nodes (that has the lower conductance) forms a more community-like set of nodes than the set of *blue* or *red* nodes. The conductance of the graph is $\Phi_G = 2/18$, which represents the minimum conductance of any set of nodes C, where $|C| \leq |V|/2$.

function is optimized. In our analysis, the popular graph partitioning package Metis [Karypis and Kumar 1998] has been used to obtain sets of nodes that have very good conductance scores. Metis is a multilevel graph partitioning algorithm that can give both fast execution times and high quality results.

2.4. Relation between Conductance and Mixing Time

Conductance is a measure of the tendency of a random walk to move out of a subset of nodes. The relationship between the graph conductance and the mixing time of a random walk is explained by the following inequalities [Alon 1986; Jerrum and Sinclair 1989]:

$$(1 - 2\Phi(G))^t \leq \Delta(t) \leq \frac{2|E|}{\min_{v \in V} \cdot \deg(v)} \left(1 - \frac{\Phi(G)^2}{2}\right)^t.$$

The aforementioned inequalities effectively provide a characterization of a SRW's mixing time in terms of the graph conductance $\Phi(G)$. One can see that the larger the graph conductance $\Phi(G)$ is, the smaller the mixing time t will be. Intuitively, high conductance would imply that the graph has a relatively low mixing time (see [King 2003] for details and discussion). Note, as well, that because of the logscale relationship between $\Phi(G)$ and the mixing time t, a small change on $\Phi(G)$ will lead to a significant change of t. Now, if ϵ is a threshold, let

$$\frac{2|E|}{\min_{v \in V} \cdot \deg(v)}(1 - \Phi(G)^2/2)^t \leq \epsilon.$$

By applying the natural logarithm to both sides, we get

$$t \geq \frac{1}{\log(1 - \Phi(G)^2/2)} log\left(\frac{\epsilon}{\frac{2|E|}{\min_{v \in V} \cdot \deg(v)}}\right). \quad (1)$$

Based on the minimum value of t in (1), we can define a theoretical mixing time of the fastest random walk in a graph. We employ this measure in the experimental evaluation.

3. THE PROBLEM

Let $G(V, E)$ be a connected undirected simple graph. Now, assume that we are allowed to augment G with a small number of edges, with the objective of optimizing a connectivity property. We denote G_{aug} an augmentation of G (see Figure 2). While having

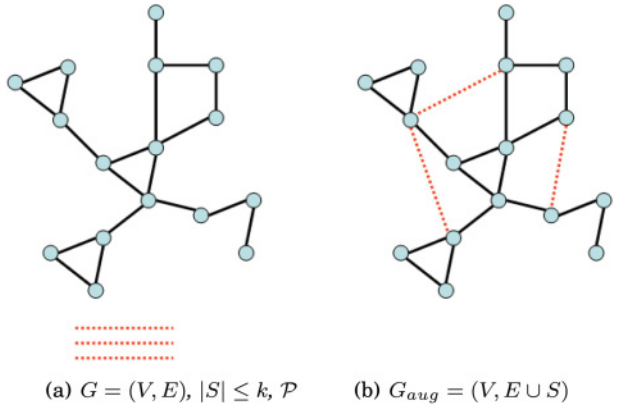


Fig. 2. Graph augmentation of G to G_{aug}. (a) The input consists of G, an integer k and an augmentation policy $\mathcal{P}$, (b) the augmented graph G_{aug} is a possible output. We ask to determine the set of shortcuts S that when used to augment G, $u(S)$ is maximized.

plenty of choices for measuring the level of connectivity in G_{aug}, we focus on a structural feature, namely the *characteristic path length*, $\bar{L}$. Note that, adding edges in a network results in altering some of the st-shortest paths in the graph, where $s, t \in V$. It is worth noting immediately that while adding long-haul edges (shortcuts with large range R) may bring a large gain in terms of connectivity for some (s, t) pairs (the new length of a shortest path connecting them will be much shorter), long-haul edges are likely to affect the shortest path length of only a few (s, t) pairs. On the other hand, suggesting short-haul, intracommunity edges (shortcuts with small range R) is likely to affect the shortest path length of many (s, t) pairs, but the gain in connectivity for each of them might be limited (i.e., the new length of a shortest path connecting them will not be much shorter). Therefore, in trying to suggest edges to be added to the graph we are facing an interesting tradeoff. Let CS represent the set of all candidate shortcuts for addition. This set represents all shortcut edges (i, j) between any nonadjacent nodes $i, j \in G$ (i.e., $R(i, j) \geq 2$). Note that in a sparse network of n nodes (most real networks are sparse), there exist (almost) $n(n-1)/2$ candidate shortcuts. Now, assume the existence of a *utility set function* $u : 2^{CS} \rightarrow \mathbb{R}$ that maps a given subset of shortcut edges $S \subseteq CS$ to a nonnegative real number that represents the difference of the sum of all-pairs shortest path length ($L(.)$) in $G = (V, E)$ and $G_{\text{aug}} = (V, E \cup S)$. Formally,

$$u(S) = L(G) - L(G_{\text{aug}}). \tag{2}$$

Another critical issue is related to the semantics of the set of shortcut edges S. From a network optimization perspective, one might be interested to add shortcuts anywhere in the network. However, in some settings (e.g., suggesting friends in a social network), it might be required to suggest a few shortcuts to each node in the network, in a balanced way. In both scenarios, we are interested in a more efficient network, but we are required to respect an *augmentation policy* $\mathcal{P}$ of adding shortcuts in the network. We consider two particular cases that we see to be of more interest in practice and define the following *augmentation policies*.

—*Network-Centric Policy*: Graph augmentations can be applied anywhere in the network. In this case, an integer k that is provided as input represents the total number of shortcuts to be added, not necessarily equally distributed over nodes. Therefore, $|S| \leq k$.

—*Node-Centric Policy*: Graph augmentations should be applied equally over all nodes in the network. In this case, an integer k that is provided as input represents the total number of shortcuts to be attached to *each node*. Therefore, $|S| \leq k|N|$.

The two *augmentation policies* control the level of flexibility in suggesting specific shortcuts to add in a graph. For example, following the *network-centric policy*, it is allowed to select k shortcuts and attach them to a single node, in an attempt to resemble the topology of a star network. The intuition behind this is that stars are network topologies with very small average shortest path distance [Meyerson and Tagiku 2009]. On the other hand, the *node-centric policy* restricts the number of shortcuts that can be attached to a single node and, in addition, requires that the same number of shortcuts is attached to any node in the network (e.g, one shortcut to each node). It becomes clear that the *augmentation policy* can have important implications on the way that algorithms that try to solve the problem operate. In one case, we seek to find the best k shortcuts globally in the network, therefore early stopping conditions can be considered, as soon as, the required number of shortcuts has been reached. In the other case, we seek to find the best k shortcuts for *each node*, therefore we might need to evaluate a possible larger set of candidate shortcuts before meeting the requirement. We further elaborate on this issue in Section 6.

We are now in position to formally define the problem of interest in this paper.

PROBLEM 1. *Let $G = (V, E)$ be a connected, undirected, simple graph, and let $CS = \{(i, j) : R(i, j) \geq 2\}$ be a set of candidate shortcuts for augmentation. If $G_{aug} = (V, E \cup S)$ is the augmented graph of G, then the objective is, given G, an integer k and an augmentation policy $\mathcal{P}$, to find a set of shortcut edges $S \subseteq CS$ that maximizes $u(S)$.*

It is easy to see that maximizing $u(S)$ is equivalent to maximizing $u'(S) = \bar{L}(G) - \bar{L}(G_{\text{aug}})$, since we only add new edges in G and do not change its number of nodes.

3.1. Problem Hardness

We establish the hardness of our problem by exploiting its equivalence with the weighted version of the *Maximum Coverage Problem* (WMCP). Not surprisingly, it turns out that our problem is NP-hard.

THEOREM 3.1. *Problem 1 is NP-Hard.*

PROOF. We prove the claim by reducing our problem to the WMCP. In WMCP the input is a universe $\mathcal{U}$ of n elements $\mathcal{U} = \{e_1, e_2, \ldots, e_n\}$, a collection $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of m subsets of $\mathcal{U}$, such that $\bigcup_i C_i = \mathcal{U}$, a weight function $w : \mathcal{C} \rightarrow \mathbb{R}$ that assigns non-negative weights to subsets and an integer $k \leq m$. The goal is to select a subset $\mathcal{S} \subseteq \mathcal{C}$, such that $|\mathcal{S}| \leq k$ and the combined weight in the union of the subsets to be maximized. It is easy to see that our problem can be reduced to WMCP as follows: let a universe $\mathcal{U}$ of n elements, where each element represents an (s, t) pair of the graph G. Let, as well, a collection of candidate shortcuts $\mathcal{CS} = \{CS_1, CS_2, \ldots, CS_m\}$, where each candidate shortcut CS_i represents a set of (s, t) pairs; this is the set of (s, t) pairs whose shortest path lengths are affected (shortened) by the addition of the corresponding shortcut in G. Moreover, every candidate shortcut is assigned a weight w that represents the utility u of adding the corresponding shortcut in G. Then, given an integer k, a collection of candidate shortcuts and their utility scores, the goal is to select a subset of shortcuts $\mathcal{S} \subseteq \mathcal{CS}$, such that $|\mathcal{S}| \leq k$ that when added in G the combined utility is maximized. Problem 1 is NP-hard by reduction to the WMCP, one of the classical NP-hard combinatorial optimization problems. This concludes our proof. □

3.2. Negative Results

The maximum coverage problem is NP-hard, and cannot be approximated within $1 - \frac{1}{e} + o(1) \approx 0.632$ under standard assumptions. This result matches the approximation ratio achieved by a generic greedy algorithm used for maximization of submodular functions with a cardinality constraint [Nemhauser et al. 1978]. The generic greedy algorithm for maximum coverage chooses sets according to one rule: at each stage, choose a set which maximizes the number of elements covered by the new set but not by any of the preceding ones. Moreover, due to Feige, we know that the greedy algorithm is essentially the best-possible polynomial time approximation algorithm for maximum coverage [Feige 1998]. That inapproximability result applies to the weighted version of the maximum coverage problem, as well, since it holds the maximum coverage problem as a special case.

These results, suggest that the greedy heuristic that adds the current best shortcut to a graph G, until k shortcuts are added would be the best possible heuristic. However, we show that the function $u(.)$ of adding shortcuts in a graph in order to minimize the average shortest path length, while being strictly monotonic (see Theorem 4.5), it is not submodular (see Theorem 4.6). Submodular functions have some nice parametric and postoptimality properties and related optimization problems can be solved efficiently if the submodularity property is valid [Filmus and Ward 2012; Kapralov et al. 2013; Nemhauser et al. 1978; Vondrak 2008]. Since $u(.)$ does not possess the submodularity property, techniques that utilize it for approximation cannot be used. Thus, strictly speaking, we cannot expect to have an efficient algorithm with a provable approximation guarantee for our problem. Nevertheless, given the hardness of the problem, a greedy algorithm of successively adding shortcuts by repeatedly picking the best shortcut could still be a good heuristic (see Section 5). But, since the search space is massive, the greedy algorithm would be extremely slow. So, we focus our attention on designing an *efficient heuristic algorithm*, by avoiding unnecessary coverage evaluations. We explain the details of our algorithm in Section 5.

4. SOME THEORETICAL RESULTS

In this Section, some theoretical results are discussed.

4.1. Relationship between Characteristic Path Length and Edge Betweenness in *G*

We have structured the problem statement around minimizing the characteristic path length of a graph, $\bar{L}$. We discuss here the relationship between $\bar{L}$ and another popular network structural property, namely *edge betweenness*.

Definition 4.1 (*Edge Betweenness* [*Girvan and Newman 2002*]). The betweenness centrality of an edge or simply edge betweenness is the number of shortest paths between pairs of nodes that run along it. If there is more than one shortest path between a pair of nodes, each path is assigned equal weight such that the total weight of all of the paths is equal to unity. Formally, let $\sigma_{st} = \sigma_{ts}$ denote the number of shortest paths from $s \in V$ to $t \in V$ and let $\sigma_{st}(e)$ denote the number of shortest paths from s to t that pass through edge e. Then, the edge betweenness of e is given by

$$b_e = \sum_{s,t \in V, s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}.$$

Definition 4.2 (*Total Betweenness*). Let B be the sum of the edge betweenness of all edges in G, given by

$$B(G) = \sum_{e \in E} b_e.$$

Interestingly, we show that in a graph G, the sum of all-pairs shortest paths, $L(G)$, is equal to the sum of the edge betweenness of all edges, $B(G)$.

THEOREM 4.3. *Let $G(V, E)$ be a connected undirected simple graph with V the set of vertices and E the set of edges. If L is the length of all-pairs shortest paths and B is the sum of the edge betweenness of all edges in G, then $B(G) = L(G)$.*

PROOF. Without loss of generality, we fix a pair of nodes $s, t \in V$. Assume that there are p^{st} such shortest paths between s, t, all of which have the same length $d(s, t)$. Note that each path $p_i \in p^{st}$ consists of $|d(s, t)|$ edges and contributes to the edge betweenness of each edge along the path by a factor of $\frac{1}{p^{st}}$. So, each single shortest path p_i contributes to the total edge betweenness by a factor of $\frac{d(s,t)}{p^{st}}$, and all p^{st} shortest paths of the pair of nodes $s, t \in V$ contribute by $d(s, t)$ $(= p^{st} \cdot \frac{d(s,t)}{p^{st}})$. Now, considering and summing up the total edge betweennesses of all s, t pairs in G we get $B = \sum_{s \in V} \sum_{t \in V} d(s, t)$ or $B = L$. □

Due to Theorem 4.3, minimizing $u(S)$ in equation (2) is equivalent to minimizing:

$$u''(S) = B(G) - B(G_{\text{aug}}). \tag{3}$$

4.2. Properties of the Utility Function $u(.)$

We show some properties of the utility function $u(.)$.

LEMMA 4.4. *Let $G(V, E)$ be an undirected graph and let $G_{aug} = (V, E \cup \{e\})$ be an augmentation of G after the addition of an edge $e \in CS$. Then, $L(G_{aug}) < L(G)$.*

PROOF. Without loss of generality, we fix a pair of nodes $s, t \in V$ and let $d(s, t)$, $d'(s, t)$ be the length of a shortest path connecting them in G and in G_{aug}, respectively. For any shortest path between s and t in G_{aug}, there are two options:

—it does not traverse e: then $d'(s, t) = d(s, t)$;
—it traverses e: then $d'(s, t) < d(s, t)$.

Therefore, for any pair of nodes $s, t \in V$ it holds that $d'(s, t) \leq d(s, t)$. Summing up the shortest path lenghts of all s, t pairs in G_{aug} and G we get $L(G_{\text{aug}}) \leq L(G)$ (*monotonically decreasing*). Now, let the new edge e be incident to two nodes, say x and y. It is easy to see that the (x, y)-shortest path in G_{aug} is always shorter than the (x, y)-shortest path in G. Therefore, after the addition of the edge e in G, there is at least one pair of vertices, where $d'(x, y) < d(x, y)$. It follows that $L(G_{\text{aug}}) < L(G)$. □

THEOREM 4.5. *$u(.)$ is strictly monotonic.*

PROOF. From Lemma 4.4, it follows that adding an edge $e \in CS$ to a set $S \subset CS$ will always cause $u(.)$ to increase, so $u(S \cup \{e\}) > u(S)$ for any edge e and set S (strictly increasing). □

THEOREM 4.6. *$u(.)$ is not submodular.*

PROOF. Let Z be a finite set. A function $f : 2^Z \rightarrow R$ is submodular if for any $X \subset Y \subset Z$ and $e \in Z \setminus Y$, it holds that:

$$f(X \cup \{e\}) - f(X) \geq f(Y \cup \{e\}) - f(Y).$$

Intuitively, a submodular function over the subsets demonstrates "diminishing returns." We provide a counterexample that proves that the utility function $u(.)$ does not possess the submodularity property.

Counterexample: Let $G(V, E)$ be an undirected graph and let the sets $X = \{\}$, $Y = \{e_1\}$ and $Z = \{e_1, e_2\}$ (see Figure 3). It is easy to see that $X \subset Y \subset Z$. Note that the edges

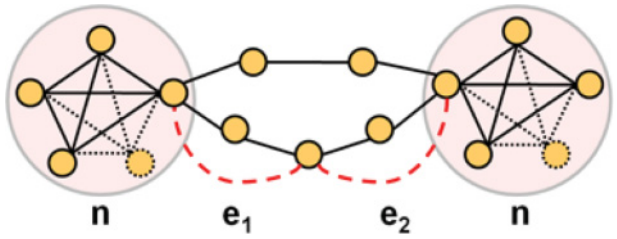


Fig. 3. Submodularity counterexample. Assume $G(V, E)$ and let the sets $X = \{\}$, $Y = \{e_1\}$ and $Z = \{e_1, e_2\}$.

in G form two very large fully connected subgraphs, each of size n (i.e., the number of nodes in each subgraph). Then, for $e = e_2 \in Z \setminus Y$ holds that:

$$\begin{aligned} u(X \cup \{e\}) - u(X) &= (L(G) - L(G(V, E \cup X \cup \{e\}))) \\ &\quad - \big(L(G) - L(G(V, E \cup X))\big) \\ &= L(G(V, E \cup X)) - L(G(V, E \cup X \cup \{e\})) \\ &\simeq 5\frac{n(n-1)}{2} - 5\frac{n(n-1)}{2} = 0 \end{aligned}$$

and

$$\begin{aligned} u(Y \cup \{e\}) - u(Y) &= (L(G) - L(G(V, E \cup Y \cup \{e\}))) \\ &\quad - (L(G) - L(G(V, E \cup Y))) \\ &= L(G(V, E \cup Y)) - L(G(V, E \cup Y \cup \{e\})) \\ &\simeq 5\frac{n(n-1)}{2} - 4\frac{n(n-1)}{2} = \frac{n(n-1)}{2}. \end{aligned}$$

The aforementioned coefficients 4 and 5 represent the typical shortest path lengths for (almost) all pairs of nodes in the network instances.

Asymptotically it holds that:

$$u(X \cup \{e\}) - u(X) < u(Y \cup \{e\}) - u(Y).$$

Therefore, we obtain a contradiction and we can conclude that the submodularity property does not hold. □

5. SHORTCUT UTILITY COMPUTATION

Let u_{xy} represent a measure of the utility of adding a single shortcut edge $(x, y) \in CS$ in G. According to (2), it is:

$$u_{xy} = L(G) - L(G_{\text{aug}})$$

To evaluate u_{xy}, we need to recompute the shortest paths between all pairs of vertices $s, t \in V$ in G_{aug}. This is because a previously found shortest path connecting s, t may now need to traverse the newly added edge (x, y). Let $d'(s, t)$ represent the length of the shortest path between s and t in G_{aug}. Then:

$$u_{xy} = \sum_{(s,t) \in V \times V \setminus E} (d(s, t) - d'(s, t)).$$

In the following sections, we describe methods for computing the utility of all candidate shortcuts for augmentation. We first describe a greedy method. Then, we explain why this approach is not scalable and thus not suitable for larger graphs, and discuss methods that can scale up its performance.

ALGORITHM 1: Greedy Method

Input: A Social Graph $G(V, E)$ and a set of Candidate Shortcuts $CS = \{(x, y) : R(x, y) \geq 2\}$
Output: A shortcut edge $e_{\max}$ with maximum utility

```
L(G) = computeL(G);
u_max = 0;
forall the (x, y) ∈ CS do
    G_aug ← G(V, E ∪ {(x, y)});                    // add (x, y) in G
    L(G_aug) = computeL(G_aug);
    u_xy = L(G) − L(G_aug);
    if u_xy > u_max then
        u_max = u_xy;
        e_max ← (x, y);
    end
end
return e_max;
```

5.1. Greedy Method

We describe a general greedy method (see Algorithm 1), where all candidate shortcuts are considered and a shortcut is selected in each iteration that lowers the average shortest path length of the graph as much as possible (locally optimal choice). The algorithm operates in k rounds. In each round i, the algorithm determines the shortcut with the highest utility and adds it to the set of the top-k edges to be suggested for augmentation. Equivalently, this means that the shortcut selected in round i is the one that minimizes the average shortest path length in this round and for this graph. In more detail, the method (see Algorithm 1) iterates over all candidate shortcuts CS and for each shortcut $(x, y) \in CS$ estimates its associated utility u_{xy} by adding it to the graph and computing the difference between the sums of all-pairs shortest path lengths in the original graph G and the augmented graph G_{aug}. Then, the candidate shortcut (x, y) is removed from the graph and the process is repeated for the rest ones. In order to determine the best shortcut for augmentation, the method needs to compute the utility of all candidate shortcuts of a graph.

The worst-case time complexity of the greedy method is $O(k \cdot n^2 \cdot n(n\log n + m))$: the utility computation of a single candidate shortcut has time complexity $O(n(n\log n + m))$ (equivalent to running the Johnshon's algorithm for all-pairs shortest paths one time [Johnson 1977], using Fibonacci heaps [Fredman and Tarjan 1987] in the implementation of Dijkstra's algorithm), there are $|CS| = n(n-1)/2 - m = \sim n^2$ candidate shortcuts that need to be evaluated in order to pick the best one, and to find the best k, one would need to repeat the procedure k times.

5.2. Relaxation of the Greedy Method

It is possible to drop k if instead of computing the best candidate shortcut, one at a time, we use the already computed shortcut utility scores from the first iteration and determine the best k shortcuts all at once (*batch processing*). While this variant is expected to be faster (time complexity is $O(n^2 \cdot n(n\log n + m))$), it might affect the overall performance of the shortcuts added, as the addition of a shortcut might lower the utility of a subsequent one. We discuss this tradeoff in the experimental evaluation section.

5.3. Path Screening Method

We describe a novel method that can speed up the process of computing the utility of all candidate shortcuts for augmentation. The method works by decomposing the

computation of the sum of the shortest path lengths into subproblems. It is easy to see, that in the general case the addition of a shortcut $(x, y) \in CS$ to G, alters the shortest path lengths of many (s, t) pairs with $s \in V$ and $t \in V$. We say that these (s, t) pairs *depend* on shortcut (x, y). However, many st-shortest paths do not depend on (x, y), and as such the computation of the shortcut utility does not need to consider them. Let $D_{st}(x, y)$ be the set of (s, t) pairs that depend on the shortcut (x, y). For each (s, t) pair, it holds that:

$$\begin{cases} (s,t) \in D_{st}(x,y) & \text{if } d'(s,t) < d(s,t) \\ (s,t) \notin D_{st}(x,y) & \text{if } d'(s,t) = d(s,t). \end{cases}$$

Note that for the majority of shortcuts (x, y), the number of (s, t) pairs that actually depend on a shortcut is much smaller than the total number of node pairs in G (i.e., $|D_{st}(x, y)| \ll |V \times V \setminus E|$). This suggests large savings in the computation of the utility u_{xy}, since in fact we only need to sum up the differences in the shortest path length of a much smaller set of pairs $(s, t) \in D_{st}(x, y)$:

$$u_{xy} = \sum_{(s,t) \in D_{st}(x,y)} (d(s,t) - d'(s,t)). \tag{4}$$

Next, we present a theorem that states that given a shortcut edge of a fixed range δ in G, it is more beneficial to attach this shortcut on nodes of an existing *st*-shortest path. We use this theorem to further decompose u_{xy} in (4).

THEOREM 5.1. *Let $G(V, E)$ be an undirected graph and let p_s be a shortest path and p_a be an alternative (i.e., nonshortest) path between nodes $s \in V$, $t \in V$. Assume an addition of a new shortcut $\{(x, y) : R(x, y) = \delta, \delta \leq d(s, t)\}$ in G. Then, it holds that $d'^{p_s}(s, t) < d'^{p_a}(s, t)$, if both $x \in V$, $y \in V$ are nodes traversed by p_s in G, where $d'^{p_s}(s, t)$, $d'^{p_a}(s, t)$ are the lengths of the shortest paths p_s, p_a in G_{aug}, respectively.*

PROOF. Let p_s represent a shortest path in G and assume that one (or both) of $x \in V$, $y \in V$ is not traversed by p_s (or any of the shortest paths connecting s, t). Now, let p_a represent a nonshortest path in G that connects $s \in V$, $t \in V$ and traverses both x, y. Then, (since p_a is not a shortest path) it holds that:

$$d^{p_a}(s, t) > d^{p_s}(s, t) \tag{5}$$

where $d^{p_a}(s, t)$, $d^{p_s}(s, t)$ are the lengths of the paths p_s and p_a in G. Now assume addition of a shortcut $\{(x, y) : R(x, y) = \delta\}$ in G. Then, it holds that:

$$d^{p_a}(s, t) = d'^{p_a}(s, t) + (\delta - 1) \tag{6}$$

where $d'^{p_a}(s, t)$ is the length of the path connecting (s, t) in the augmented graph G_{aug}. This is due to the constraint that $R(x, y) = \delta$. By (5) and (6), it follows that:

$$d'^{p_a}(s, t) + (\delta - 1) > d^{p_s}(s, t). \tag{7}$$

On the contrary, it is easy to see that if both x, y were traversed by p_s, then after addition of an edge $\{(x, y) : R(x, y) = \delta\}$ in G, it would hold that:

$$d^{p_s}(s, t) = d'^{p_s}(s, t) + (\delta - 1). \tag{8}$$

By (7) and (8), it follows that:

$$d'^{p_a}(s, t) > d'^{p_s}(s, t). \tag{9}$$

This concludes our proof. □

Theorem 5.1 states that given a shortcut edge of a fixed range δ in G, it is more beneficial to attach this shortcut on nodes of an existing *st*-shortest path. As such, it

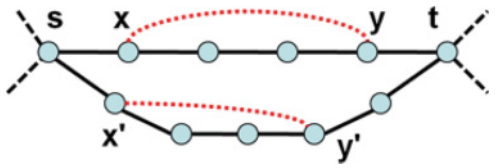


Fig. 4. The (s, t) pair belongs in both $D_{st}(x, y)$ and $D_{st}(x', y')$, but only to $D_{st}^{SP}(x, y)$. It doesn't belong to $D_{st}^{SP}(x', y')$, as x' or y' are not nodes of the st-shortest path.

constrains the set of (s, t) pairs that depend on a candidate shortcut (x, y) to be (s, t) pairs that their shortest paths in G traverse both nodes x and y. Let this set of (s, t) pairs that depend on a candidate shortcut (x, y) be $D_{st}^{SP}(x, y)$ (see Figure 4). Then, we can further decompose u_{xy} in (4) by:

$$\begin{aligned} u_{xy} &= \sum_{(s,t)\in D_{st}^{SP}(x,y)} ((d(s,x) + d(x,y) + d(y,t)) \\ &\quad - (d(s,x) + d'(x,y) + d(y,t))) \\ &= \sum_{(s,t)\in D_{st}^{SP}(x,y)} (d(x,y) - d'(x,y)). \end{aligned}$$

Note that the shortest path length $d(x, y)$ represents the range of (x, y) in G, $R(x, y)$. Moreover, the shortest path length $d'(x, y)$ represents the length of the newly added shortcut in G_{aug}, which is constant $(d'(x, y) = 1)$. Therefore,

$$u_{xy} = \sum_{(s,t)\in D_{st}^{SP}(x,y)} (R(x,y) - 1). \tag{10}$$

Eventually, the utility u_{xy} depends on the range of (x, y) in G and the size of $D_{st}^{SP}(x, y)$, in accordance with our initial intuition.

5.3.1. Fast Assessment of Shortcut Utility Scores. We look to efficiently compute the utility score u_{xy} of any shortcut $(x, y) \in CS$ in G according to (10). Algorithm 2 describes the steps to solve the problem. It uses a modification of Johnson's Algorithm [Johnson 1977] that does not only provide the lengths of the all-pairs shortest paths, but also reconstructs and stores them in P. Then, for each shortest path $p \in P$, it determines what are the (x, y) shortcuts that could shorten the current path p by sliding a variable-size window of size δ over the path's nodes (see Figure 5). Note that δ takes values that range from $\delta = 2$ (shortcuts between nonadjacent nodes) to $\delta = l$, where l is the length of the current path p. It is easy to see that the maximum δ to be considered will be equal to the *diameter* D of G, where $D = \max_{i,j} d(i, j)$ (the longest shortest path between any two nodes in G), so the range of values of δ is $2 \leq \delta \leq D$. As we slide the window over a path, the window's endpoints determine the x and y nodes of a candidate shortcut (x, y). Each time a shortcut (x, y) is encountered in a path p, an increment equal to $(\delta - 1)$ is contributed to its utility. The procedure continues over all paths $p \in P$ and at the end, for each (x, y) its utility u_{xy} is computed in accordance to (10). The time complexity of our path screening method is $O(n(n \log n + m)) + O(n^2 \bar{L}(G)\bar{L}(G))$: it requires $O(n(n \log n + m))$ time to find the shortest paths in G (equivalent to running the Johnson's algorithm ones). Then, for each path $(O(n^2))$ needs to vary the δ-size of the window $(O(\bar{L}(G)))$ and slide the window over the path p $(O(\bar{L}(G)))$.

5.4. Path Screening Sampling-Based Method

Algorithm 2 describes an efficient way to accurately compute the utility of each candidate shortcut u_{xy}. However, it is still not very practical for large graphs as it requires

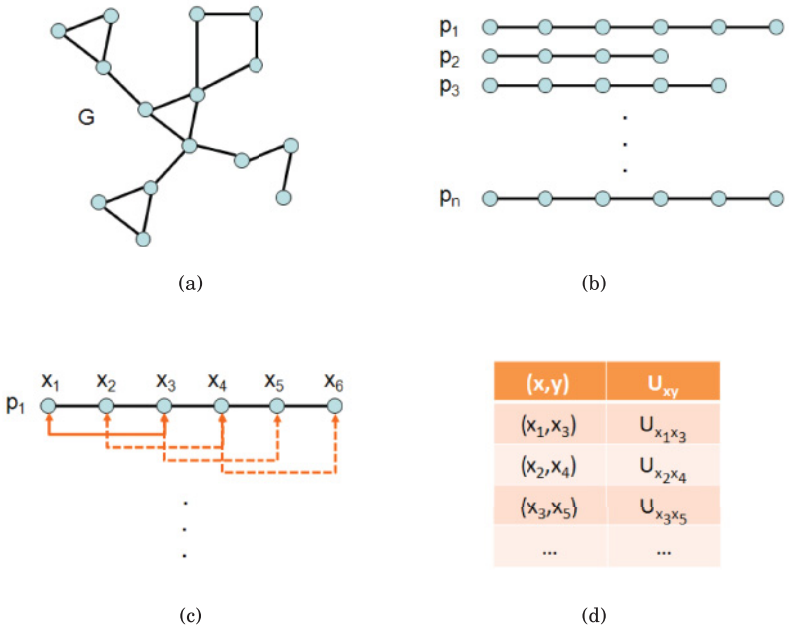


Fig. 5. Path screening method. (a) The input consists of a graph G. (b) All-pairs shortest paths are extracted from G. (c) A δ-size window (for varying δ) is slided over each shortest path. (d) Occurrences of any candidate shortcut (x, y) found contribute to its utility u_{xy}.

ALGORITHM 2: Path Screening Method

Input: A Social Graph $G(V, E)$
Output: A Map between Candidate Shortcuts and their utility scores

$U \leftarrow \emptyset$;
$P = getAllPairsShortestPaths(G)$;
forall the $p \in P$ **do**
  $l = length(p)$;
  // Variable size of window
  for $\delta = 2$ *to* l **do**
    // Slide a δ-size window over nodes of p
    for $i = 0$ *to* $l - \delta$ **do**
      $x = p[i]$; // ith node of path p
      $y = p[i + \delta]$; // $(i + \delta)$th node of path p
      if $(x, y) \notin U$ **then**
        $u_{xy} = (\delta - 1)$;
        add $< (x, y), u_{xy} >$ in U;
      else
        update $< (x, y), u_{xy} + (\delta - 1) >$ in U;
      end
    end
  end
end
return U

that all-pairs shortest paths can be efficiently computed ($O(n(n \log n + m))$) and then screened ($O(n^2 \bar{L}(G)\bar{L}(G))$). In this section, we describe a sampling-based variant of the path screening method for efficiently computing the utilities of the candidate shortcuts. Ideally, we would like to sample uniformly at random a few shortest paths from the set of all available ones. Then, we would feed them to our Algorithm 2 and compute estimates of the utility of each candidate shortcut based on that sample. However, uniform shortest path sampling is rarely feasible [Kandula and Mahajan 2009]. Instead, we sample uniformly at random a set of Q nodes $Q = \{q_1, \ldots, q_q\}$. Then, starting from a node q_i, $i = 1, \ldots, q$, where $q \ll n$, we execute the Dijkstra algorithm [Dijkstra 1959] and compute the single source shortest path tree from q_i to all nodes x in the graph. Now, instead of computing the utility of a candidate shortcut by operating over all-pairs shortest paths, we operate only on the set Q of shortest paths returned by the sample:

$$u_{xy}^{Q} = \sum_{s \in Q, t \in V} d(s,t) - d'(s,t).$$

The way we perform the sampling reduces the search space where candidates can be found (as not all shortest paths become available) and is likely to introduce bias. The main reason for the bias is that some paths are more likely to exist in the shortest path trees of sampled sources [Kandula and Mahajan 2009]. However, in our setting, we are interested in finding only a few candidate shortcuts that have very high utility. Note that edges with high betweenness centrality [Newman 2005] in G are more likely to be in the sample as by definition they are edges that are traversed by many st-shortest paths. And, since important candidate shortcuts are shortcuts (x, y) that their endpoints x, y are found in many st-shortest paths, it is expected that candidate shortcuts with high utility will be sufficiently represented in the single-source shortest path trees of the sampled nodes. Therefore, our path screening method will be able to detect them. We assess the effect of the bias in the experimental evaluation. The rest of the algorithm remains the same. The point is that the aforementioned computation is relatively efficient, since it does not need to compute all-pairs shortest paths. Instead, we compute q times the single-source shortest-path Dijkstra. Thus, the time complexity of our sampling-based path screening method is $O(q(n \log n + m)) + O(qn\bar{L}(G)\bar{L}(G))$, where $q \ll n$.

5.4.1. Sampling-Based Method Estimation. In this section, we formally describe the sampling-based estimation method. First, we employ a commonly used graph embedding technique, called *reference node embedding* [Goldberg and Harrelson 2005; Kleinberg et al. 2004; Qiao et al. 2012], to show that the distances between any two nodes in a graph can be estimated if a small set of reference nodes are selected [each serving as the root of a shortest path tree (SPT)] and then the distances between these nodes to all other nodes in a graph are computed.

Definition 5.2. Let T be a rooted tree with n nodes. The lowest common ancestor (lca) of two nodes u and v in T is the shared ancestor of u and v that is located farthest from the root.

THEOREM 5.3. *Given a set of reference nodes $Q = \{q_1, q_2, \ldots q_q\}$ $\forall s, t \in V$ it holds that:*

$$d(s,t) \geq \max_{q \in Q} \{|d(s,q) - d(q,t)|\} \tag{11}$$

$$d(s,t) \leq \min_{q \in Q} \{d(lca^q, s) + d(lca^q, t)\} \tag{12}$$

PROOF. Given a set of reference nodes $Q = \{q_1, q_2, \ldots q_q\}$, our sampling-based method considers the shortest paths for the node pairs $\{(q, v) : q \in Q, v \in V\}$ by computing the

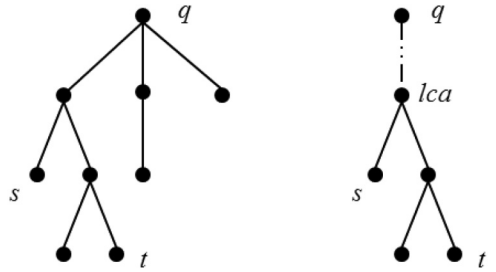


Fig. 6. Example shortest path trees (SPT). (a) SPT rooted at $q \in Q$. (b) Lowest common ancestor (lca) in a SPT.

single-source shortest path trees for every $q \in Q$. The shortest-path distance in graphs is a metric, and therefore it satisfies the triangle inequality. That is, given a pair of nodes (s, t), $\forall q \in Q$ the following inequalities hold (see Figure 6(a)):

$$d(s,t) \geq |d(s,q) - d(q,t)| \tag{13}$$

$$d(s,t) \leq d(s,q) + d(q,t) \tag{14}$$

By considering the lcas, tighter bounds are feasible [see Figure 6(b)]. $\forall q \in Q$ it holds that:

$$d(s,q) + d(q,t) = d(\text{lca},s) + d(\text{lca},t) + 2d(\text{lca},q).$$

As $d(\text{lca}, q) \geq 0$, we have:

$$d(s,\text{lca}) + d(\text{lca},t) \leq d(s,q) + d(q,t).$$

Consequently,

$$d(s,t) \leq d(s,\text{lca}) + d(\text{lca},t).$$

By considering all reference nodes $q \in Q$, we have tighter bounds:

$$d(s,t) \geq \max_{q \in Q}\{|d(s,q) - d(q,t)|\} \tag{15}$$

$$d(s,t) \leq \min_{q \in Q}\{d(\text{lca}^q,s) + d(\text{lca}^q,t)\} \tag{16}$$

where lca^q is the lca of s, t in an SPT rooted at $q \in Q$. This concludes the proof. □

Then, we show that despite the fact that our sampling-based method picks shortcuts to add based on a small sample of shortest paths, the addition of these shortcuts in G will eventually shrink the shortest path length of *any* (s, t) pair.

THEOREM 5.4. *Let $d^i(u, v)$ denote the distance between $u \in V$, $v \in V$ after i iterations, where in each iteration a single shortcut is added in G. Then, $\forall s, t \in V$ and sufficiently large n it holds that:*

$$d^n(s,t) < d(s,t) \tag{17}$$

PROOF. Assume addition of a shortcut in G. Then, $\forall u, v \in V$ in G' it holds that:

$$d'(u,v) \leq d(u,v) \tag{18}$$

since the new shortcut might shorten the distance between $u \in V$, $v \in V$ or not. Now, let $d^i(u, v)$ denote the distance between $u \in V$, $v \in V$ after i iterations, where in each

Table I. Comparison of the Worst-Case Time Complexity of the Various Methods

Method	Worst-Case Time Complexity
Greedy	$O(k \cdot n^2 \cdot n(n\log n + m))$
Relaxation of Greedy	$O(n^2 \cdot n(n\log n + m))$
Path Screening	$O(n(n \log n + m)) + O(n^2 \bar{L}(G)\bar{L}(G))$
Path Screening Sampling-Based	$O(q(n \log n + m)) + O(qn\bar{L}(G)\bar{L}(G))$, where $q \ll n$

iteration a single shortcut is added in G. Then, it holds that:

$$d^1(s,t) \leq d^1(s,q) + d^1(q,t) = \alpha_1$$
$$d^2(s,t) \leq d^2(s,q) + d^2(q,t) = \alpha_2$$
$$\cdots$$
$$d^n(s,t) \leq d^n(s,q) + d^n(q,t) = \alpha_n$$

and because of (18), it holds that:

$$\alpha_1 \geq \alpha_2 \geq \cdots > \cdots \geq \cdots > \cdots \geq \alpha_n.$$

Note that with a sufficient large n there will eventually be instances where the equality can be dropped as a new edge will shrink one of the distances $d(s,q)$ or $d(q,t)$. Finally, after n iterations, the distance between s and t will fall below a constant α:

$$d^n(s,t) \leq d^n(s,q) + d^n(q,t) = \alpha_n \leq \alpha$$

where $\alpha \ll d(s,t)$. Therefore,

$$d^n(s,t) < d(s,t).$$

This concludes the proof. □

Theorem (5.3) states that for any *st*-shortest path not included in the sample, alternative shortest paths exist that can be used to estimate its length and this estimation is bound. Theorem (5.4) states that by adding shortcuts that aim to shorten the length of specific shortest paths in the sample, eventually we manage to shorten many more *st*-shortest path lengths in the original graph, and thus successfully shrink the characteristic path length of the graph. The premise is that our sampling-based method is not only fast, but it also succeeds in approximating the solution found by applying the path screening method on all-pairs shortest paths. We experimentally validate our claims and empirically show the tradeoff between efficiency and quality degradation in the evaluation section.

5.5. Summary of Methods

Table I provides a summary of the worst-case time complexity of the various methods discussed for easy reference.

6. EXPERIMENTAL EVALUATION

In this Section, we evaluate methods that suggest shortcuts for addition in a graph; they take as input a graph G, an integer k and an augmentation policy $\mathcal{P}$ and return a set S of the best shortcuts for augmenting G. Following, we describe the methods that we employ in the experimental evaluation. These reflect the case of *network-centric policy*. In the case of the *node-centric policy*, the procedure is similar, but the method has to be repeated until the best k shortcuts for *each node* in G have been determined. The following methods are considered.

Table II. Networks

Name	#Nodes	#Edges	**#CS**	#FoFs	Density	$\bar{L}(G)$
dolphins	62	159	**1,732**	448	0.084	3.357
netsc	379	914	**70,717**	2,916	0.012	6.042
yeast	2,224	7,049	**2,464,927**	73,479	0.003	4.376
ba-s	50	49	**1,176**	99	0.040	4.641
ba-m	500	499	**124,251**	1,685	0.004	7.398
ba-l	1,000	999	**498,501**	3,501	0.002	8.132
comm-s	50	127	**1,098**	197	0.103	3.786
comm-m	500	1,469	**123,281**	6,589	0.012	6.241
comm-l	1,000	2,959	**496,541**	14,620	0.006	6.764

—*Greedy-S*: The greedy method described in Section 5.1.
—*Greedy-B*: The *batch variant* of *Greedy-S* described in Section 5.2.
—*PS*: Our path screening method described in Section 5.3.
—*PSS*: Our sampling-based variant of *PS* described in Section 5.4.
—*Random*: A baseline method that selects k shortcuts uniformly at random to add in G, inspired by Meyerson and Tagiku [2009].
—*Mutual Friends Intersection (MFI)*: This is another sensible baseline method that selects shortcuts based on the number of adjacent nodes that two nodes share. This method is inspired by the common practice of online social networking services to suggest new friendships (shortcuts) to users based on the number of their mutual friends. If $N(x)$ is the neighborhood of node x and $N(y)$ is the neighborhood of node y, then the utility score of a shortcut (x, y) can be expressed as the size of the intersection of the neighborhoods of x and y.

Network Topologies: For the needs of our experimental evaluation, we consider both *real* and *synthetic* network topologies. The real network topologies, represent connected,[1] undirected, unweighted networks (*dolphins* [Lusseau et al. 2003], *netsc* [Newman 2006], *yeast* [Bu et al. 2003]). The synthetic network topologies are of two types. Networks of the first type simulate a preferential attachment network topology based on the Barabási–Albert network model [Barabási and Albert 1999]. The number of nodes and edges in these networks have been selected so as to resemble trees; trees are typically sparse (less dense) and the shortest paths between pairs of nodes are unique. We experiment with three of them, a small (*ba-s*), a medium (*ba-m*), and a large (*ba-l*). Networks of the second type represent tightly knit communities (clusters). These networks are typically more dense and have the characteristic that for a large number of pairs of nodes there are many alternative shortest paths that connect them. We experiment with three of them, a small (*comm-s*), a medium (*comm-m*), and a large (*comm-l*). Table II provides a summary of the networks we consider.[2] For each network, we report the number of candidate shortcuts, the number of candidate shortcuts that represent Friends of Friends (FoFs), the network density ($Density = 2|E|\backslash(|V|(|V|-1))$) and its characteristic path length $\bar{L}(G)$.

Evaluation Metrics: We assess the performance of the various methods according to *accuracy* and *efficiency* measures. *Accuracy* concerns the degree of usefulness of adding a specific set of shortcuts, in terms of optimizing a network property. To assess

[1]In cases where a network is not connected, we extract its largest connected component and operate on it.
[2]Regarding the choice of the graph sizes considered, one should bear in mind that in this problem we are interested in the evaluation of the utility of missing edges (shortcuts) of a graph, which in our cases range from a few thousands and hundreds of thousands to millions, as is depicted in Table II.

the accuracy of the various algorithms we define a *gain* metric, which expresses the percentage change on the *characteristic path length*, $\bar{L}$, in the original graph G and in the augmented graph G_{aug}. Formally, the gain is given by:

$$Gain = \frac{|\bar{L}(G) - \bar{L}(G_{\text{aug}})|}{\bar{L}(G)} \times 100.$$

Efficiency concerns the time performance of a shortcut edge addition method. To assess the time performance of a method we measure the period of its execution time *in milliseconds*.

Computing Environment: All our experiments have been conducted on a Dell XPS 8700 desktop running 64-bit Windows 8.1. The desktop is equipped with an Intel Core i7-4770 CPU 3.40GHz with 24.0GB of RAM. The Intel Core i7-4770 processor has four cores and uses hyperthreading, so it appears to have eight logical CPUs. Our algorithms have been implemented using the Java programming language version 8.0 (build 1.8.0_31) and the experiments are executed on a Java HotSpot 64-Bit Server VM (build 25.31-b07, mixed mode).

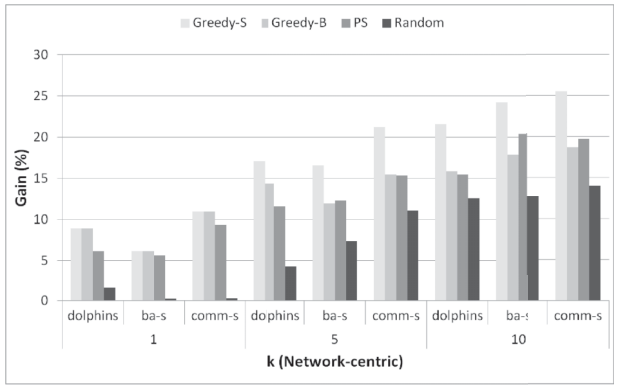
6.1. Evaluation of the Path Screening Method

Our experiments are grouped into three sets. For the methods that introduce randomness or are based on computing shortest path trees (i.e., all methods other than *Greedy-S*), single runs are susceptible to introduce bias due to randomness. We alleviate this bias by repeating the experiment many times (x10) and averaging together the results. Then, we report average values of accuracy and efficiency.

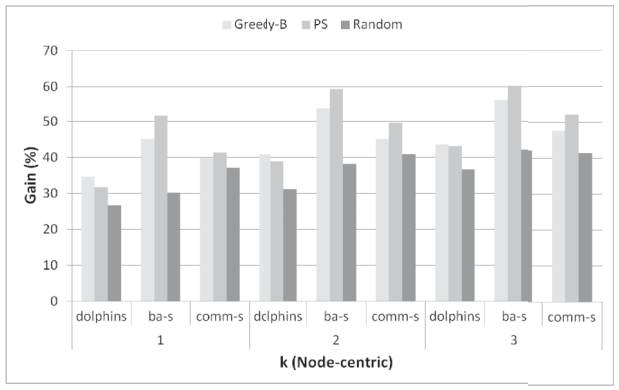
6.1.1. PS Performance/Network-centric Policy. We evaluate the *accuracy* and *efficiency* of our path screening method against the greedy methods, for a varying range of network topologies and number of edges k to be added. More specifically, we experiment with *dolphins*, *ba-s,* and *comm-s* and we add 1, 5, and 10 shortcuts. For this set of experiments, we had to limit our experiments on very small networks and on *network policy*, because *Greedy-S* is extremely slow (as we discuss below), so it is not appropriate for larger graphs or cases of *node-centric policy* (because the set of candidate shortcuts that need to be evaluated is very large).

Accuracy: Figure 7(a) presents the accuracy results for the various networks. In all instances, *Greedy-S* performs better than any other method. This is because *Greedy-S* evaluates any of the k edges one at a time. Even if this method is still suboptimal (due to making locally optimal choices), it ensures that at each iteration the correct utility for each of the subsequent candidate shortcuts is computed, taking into consideration the previously added ones. As such, it avoids adding shortcuts that have overlapping utility. On the other hand, *Random* performs poorly in all instances, as expected, and forms a baseline for the performance of the other methods. The accuracy performance of the rest two methods, *Greedy-B* and *PS* is comparable. They both perform only slightly worse than *Greedy-S*, but outperform *Random*. Overall, the performance of *PS* is always comparable and sometimes better than that of the *Greedy-B* method. On another note, the difference between the performance of *Greedy-S* and the other methods is becoming more evident as k is increasing. This is to be expected; as all the other methods add shortcuts on a batch way, the larger the number of added shortcuts is, the larger the likelihood that these shortcuts share overlapping *st*-shortest paths will be, rendering the overall gain to be smaller.

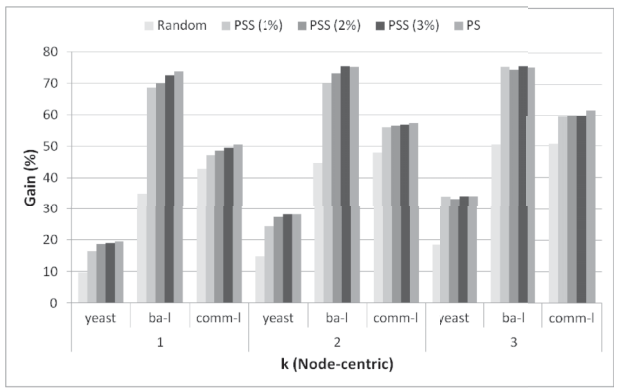
Efficiency: Table III presents the efficiency results for the various networks. Note that, for all methods other than the *Greedy-S*, it is sufficient to report only one value for each network, since these methods will evaluate the utility of all shortcuts only once. On the other hand, for the case of *Greedy-S*, we report values for all different



(a) PS Accuracy / Network-centric



(b) PS Accuracy / Node-centric



(c) PSS Accuracy

Fig. 7. Accuracy of our path screening method (PS) and its sampling-based variant (PSS).

instantiations of k (i.e., *Greedy-S-1*, *Greedy-S-5*, *Greedy-S-10*). It becomes obvious from Table III that *Greedy-S* is extremely inefficient, as it adds one shortcut at a time. In fact, *Greedy-S* will be (almost) k times slower than *Greedy-B*. More importantly, our path screening method, *PS*, outperforms *Greedy-B* by *a multitude of times* (i.e., x164,

Table III. PS Efficiency (ms) / Network Centric

Method	dolphins	ba-s	comm-s
Greedy-S-1	7,575	2,919	3,793
Greedy-S-5	33,495	10,830	14,050
Greedy-S-10	65,435	20,295	27,849
Greedy-B	7,729	3,004	3,849
PS	**47**	**39**	**41**
Random	1	1	1

Table IV. PS Efficiency (ms) / Node Centric

Method	dolphins	ba-s	comm-s
Greedy-B	7,718	3,033	3,829
PS	**47**	**31**	**32**
Random	1	1	1

x77, and x93 times faster, respectively). The gain in efficiency is due to the fact that, while *Greedy-B* computes the utility of all shortcuts, *PS* looks for shortcuts that connect nodes that lie on existing shortest paths. As such, our method suggests huge savings in the computation of shortcut utility.

6.1.2. PS Performance/Node-Centric Policy. We evaluate the accuracy and efficiency of our path screening method *PS* against *Greedy-B* and *Random*, for a varying range of network topologies. More specifically, we experiment with *dolphins*, *ba-s*, *comm-s*, and we ask to add 1, 2, and 3 shortcuts, following the *node-centric policy* (i.e., for a graph of n nodes, we ask to find n, $2n$, $3n$ shortcuts, respectively). We had to constrain our tests to smaller networks due to the inefficiency of the *Greedy-B* method.

Accuracy: Figure 7(b) presents the accuracy results for the various networks. In all instances, *PS* and *Greedy-B* perform much better than the *Random* method. It is also easy to see that as k increases the gain increases as well. This is to be expected; the larger the number of shortcuts added in the network, the lower is the average shortest path of the network and therefore, the larger the gain would be. Overall, the performance of *PS* is comparable to *Greedy-B* and always better than that of the *Random* algorithms.

Efficiency: Table IV presents the efficiency results for the various networks. Again, our path screening method, *PS*, outperforms *Greedy-B* by a *a multitude of times* (i.e., x164, x97 and x119 times faster, respectively).

6.1.3. PSS Performance. We have demonstrated that *PS* is extremely faster than *Greedy-S* and *Greedy-B*, while maintaining high levels of accuracy. However, the applicability of *PS* can be limited by the need to compute all-pairs shortest paths in the graph. In this set of experiments, we evaluate the accuracy and efficiency of *PSS*, a sampling-based variant of our path screening method.

Accuracy: We assess the performance of *PSS* against the *PS* method that serves as the ground truth and the *Random* that serves as baseline for our sampling method. We experiment in large networks (*yeast*, *ba-l*, *comm-l*) and for varying k, as well as, varying sample size. In particular, for each network, we experiment with sample sizes of 1%, 2%, and 3% of the total nodes in the network that define the methods *PSS (1%)*, *PSS (2%)*, and *PSS (3%)*, respectively, and at each run we consider adding 1, 2, and 3 shortcuts, following the *node-centric policy*. Figure 7(c) presents the results for the various instances. In all cases, the sampling accuracy increases with the sample size. This is expected, as more shortest paths are evaluated in larger samples. Moreover,

Table V. PSS Efficiency (ms) / Node Centric Policy

Method	yeast	ba-l	comm-l
Random	250	187	181
PSS (1%)	**490**	**157**	**172**
PSS (2%)	**848**	**344**	**282**
PSS (3%)	**1,179**	**437**	**422**
PS	16,475	4,365	4,764

Table VI. Large Networks

Name	#Nodes	#Edges	**#CS**	Density
facebook	4,039	88,234	**8.1E+06**	1.1E-02
astro-ph	14,845	119,652	**1.1E+08**	1.1E-03
enron	33,696	180,811	**5.6E+08**	3.2E-04
cond-mat	36,458	171,736	**6.6E+08**	2.5E-04
twitter	81,306	1,342,310	**3.3E+09**	4.1E-04
dblp	317,072	1,049,779	**5.0E+10**	2.1E-05
amazon	334,860	925,838	**5.6E+10**	1.6E-05
youtube	1,134,791	2,986,629	**6.4E+11**	4.6E-06

PSS has always a better accuracy than *Random* and is only slightly lower than that of *PS*.

Efficiency: Table V presents the efficiency results for the various methods in varying networks. The execution time of *PS* represents the time of the slowest method. For example, while *PS* requires around 4.76s for finding shortcuts to add in the *comm-l* network, *PSS (1%)* requires less than 0.2s (i.e., around x27 times faster). At the same time, the accuracy of *PSS (1%)* is comparable to that of *PS* for all k (as depicted in Figure 7(c)). Overall, PSS can be used to boost the performance of *PS*, without duly affecting its accuracy.

6.1.4. PSS Scalability. We have demonstrated that *PSS* is a multitude of times faster than standard approaches (*Greedy-S* and *Greedy-B*) and the nonsampling version of our algorithm (*PS*), while maintaining high levels of accuracy. In this set of experiments, we evaluate the scalability of *PSS* in much larger networks, following the *node-centric policy*, where $k = 1$ (i.e., we seek to add a new shortcut to each node). For this set of experiments, the rest of the algorithms we have discussed cannot be evaluated, as they are not designed to scale. In particular, we employ a few of the larger network datasets freely available online,[3] including an email communication network (*enron*), three coauthoring collaboration networks (*astro-ph*, *cond-mat*, *dblp*), four social networks (*facebook*, *twitter*, *youtube*) and a network of copurchased products (*amazon*). Details about these networks can be found in Table VI and in Leskovec and Krevl [2014]. To allow our *PSS* algorithm to scale, we control the *sample size*; for the smaller of these graphs, we considered a sample of 1% of the network nodes [*PSS (1%)*] and for the larger of these graphs we considered a sample of 0.01% of the network nodes [*PSS (0.01%)*]. Recall that, as we discussed in Section 5.4, each of the nodes of the sample is used as a *reference node* (or root) on which we execute the Dijkstra algorithm and compute the single source shortest path tree from that node to all other nodes in the graph. Then, these shortest path trees are processed in order to determine the best shortcuts to add in the graph. Table VI also lists the number of all candidate shortcuts in the graph and the density of each of these large networks. It is important to note

[3]snap.stanford.edu/data/.

Table VII. PSS Scalability (ms)

Method	facebook	astro-ph	enron	cond-mat
PSS (1%)	**2,563**	**26,002**	**114,360**	**175,620**

Table VIII. PSS Scalability (ms)

Method	twitter	dblp	amazon	youtube
PSS (0.01%)	**12,845**	**247,829**	**643,749**	**2,976,239**

that the number of candidates shortcuts can be really big; in the case of the *youtube* network, this is approximately *640 billion candidate shortcuts*.

Efficiency: Tables VII and VIII present the time efficiency results for the *PSS (1%)* and the *PSS (0.01%)* method, respectively, for a number of large networks. It is clear that our *PSS* method can scale to large networks very well, while standard methods are not even applicable. For example, *PSS (0.01%)* needs less than 50min (i.e., 2,976,239 ms) to find which shortcuts to add in the *youtube* network that consists of ∼1.1 million nodes, ∼3 million edges and a total of ∼640 billion candidate shortcuts. Overall, PSS can be used to boost the performance of *PS*.

Accuracy: Accuracy results for this set of experiments are excluded, as they would require to report on the performance of our algorithms against alternatives methods. But, as discussed earlier, alternative methods do not scale well and therefore it is not possible to apply them on networks of this size.

6.2. The Cascade Effect

We have demonstrated that our path screening method outperforms the accuracy of sensible baselines and at the same time is extremely efficient. When we introduced the problem we made a connection between the efficiency of a network to propagate information and its network structure. In this set of experiments, we evaluate the impact of our methods in the cascade process. In particular, we evaluate the impact on *the graph conductance*, on *the Markov Chain mixing time* and on *the information propagation spread*.

6.2.1. Impact on Graph Conductance. We evaluate the impact of our methods in increasing the conductance of a graph. More specifically, we experiment with both real and synthetic networks of small, medium, and large size. For each case, we add 1, 5, or 10 shortcuts using the *Random*, *PS*, or *Greedy-B* method, following the *network-centric augmentation policy*. Figure 8 shows the effect of each method on the original graph's conductance. Note that we had to constrain the computations of the *Greedy-B* method to only small networks due to its inefficiency. The plots indicate that the graph conductance (minimum conductance of any cluster size) is increased more when we augment the graph using our path screening method *PS*, than the *Random* baseline and (almost) always this increase is only slightly smaller than that achieved by *Greedy-B*. Furthermore, conductance is increasing with k; this is depicted by the larger increase in the cases of adding 5 or 10 shortcuts, for all networks. Moreover, this increase is more expressed in the case of the community networks *comm-s, comm-m, comm-l*. This is to be expected, as these networks consist of only a few cross-cutting edges.

6.2.2. Impact on Markov Chain Mixing Time. We evaluate the impact of our methods in decreasing the mixing time of a graph using the same parameter settings as in Section 6.2.1. We report the theoretical mixing time of the fastest random walk in a network, in terms of the number of steps (i.e., random walk length) before it converges (i.e., reaching a specific relative point-wise threshold ϵ). For the needs of our experiments, we set $\epsilon = 0.01$. It is easy to see that a smaller threshold would end up in

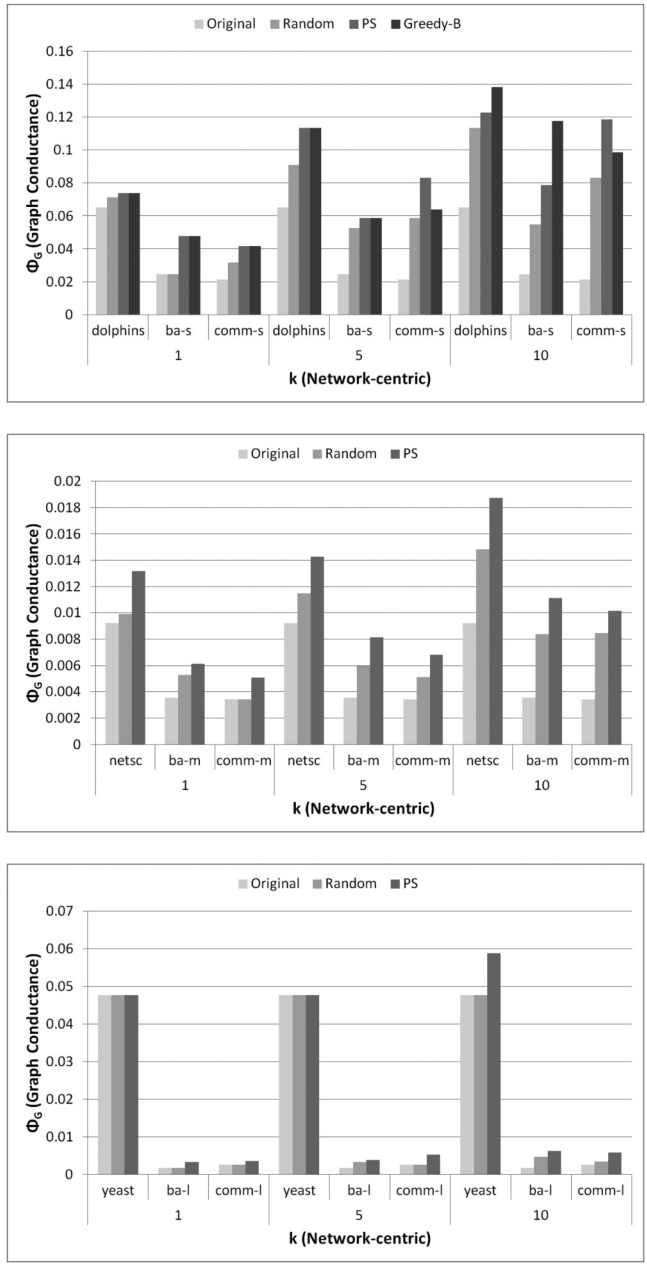


Fig. 8. Impact on graph conductance on small (top), medium (middle), and large (bottom) graphs.

larger savings and a larger one in smaller savings. Figure 9 shows the effect of each method on the original graph's mixing time. The plots indicate that the mixing time is decreasing more when we augment the graph using our path screening method *PS*, than the *Random* baseline and (almost) always this decrease is comparable to the one caused by *Greedy-B*. The trend is consistent for *yeast*, but does not appear properly due to small values. Furthermore, the mixing time is decreasing with the number of

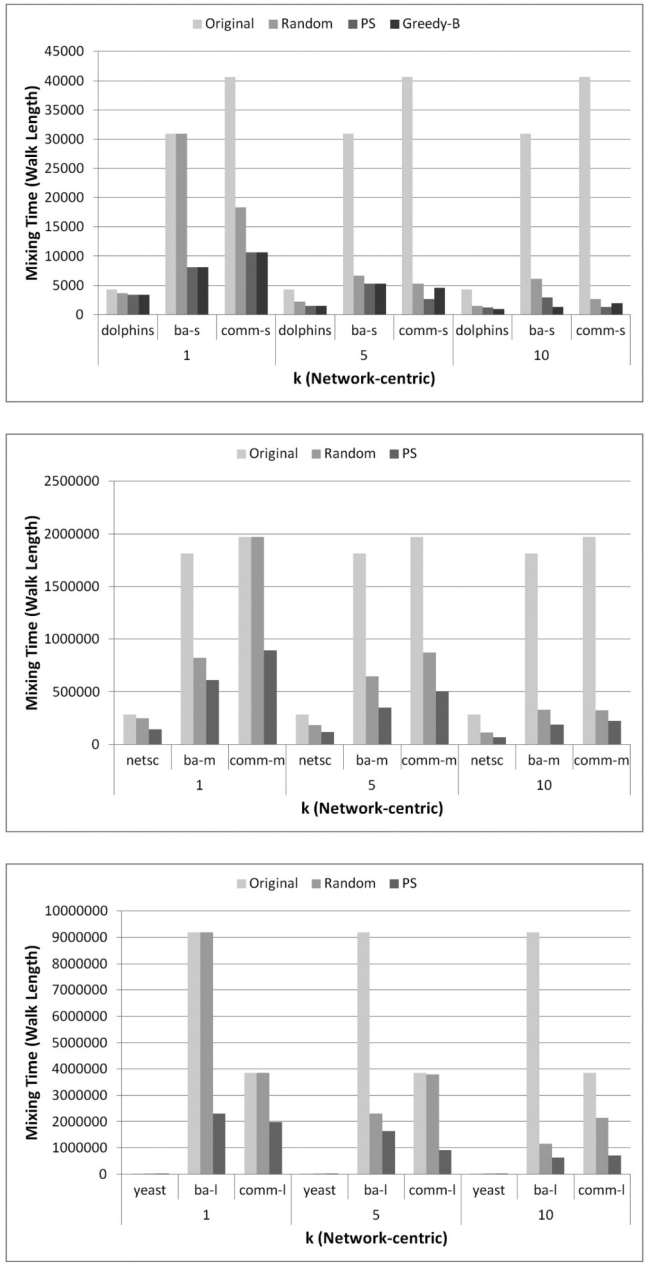


Fig. 9. Impact on Markov chain mixing time on small (top), medium (middle), and large (bottom) graphs.

shortcuts added and this decrease is, as with the graph conductance, more expressed in the case of the community-like networks.

6.2.3. Impact on Information Propagation Spread. We have demonstrated that our method can dramatically affect properties of the graph that are critical for characterizing its ability to carry on propagation processes, such as the graph conductance and its mixing

time. In this set of experiments, we evaluate the impact of our methods on the *cascade spread*; the number of nodes reached by a propagation process. It is important to note that our methods are generic and do not depend on a specific propagation model. Consequently, we first present a *simple generic propagation model* that is used in the evaluation and then we describe the evaluation in detail.

Propagation Model: We employ a basic threshold propagation algorithm based on an individual-level natural model of direct-benefit effects in networks due to Stephen Morris [Morris 2000]. The underlying consideration of this model is that a node has a certain social network neighbors and the benefits to adopt a new behavior increase as more and more of these neighbors adopt it. Conforming to the model, a set of initiator nodes I in the network has adopted a behavior. At each round, a node should adopt the new behavior once a sufficient proportion of its neighbors have done so. We model the decision making using a simple threshold rule; if a fraction of at least T of a node's neighbors have adopted, then it should, too. The semantics of the threshold are intuitive; the smaller the T is the more attractive the cascading behavior is, as you need fewer of your neighbors to engage before you do. Nodes that have already adopted the behavior cannot switch back. The propagation process terminates when no change is detected in a specific round (no new node has adopted). In a sense, the number of nodes that have adopted in the end of the propagation process represents a *sphere of influence* (*SOI*), of the initiator set I.

Given a graph, first we add new edges based on our path screening method, *PS*, and then add new edges based on *MFI*. For the needs of this experiment, we had to constrain the set of candidate shortcuts to represent FoFs ($CS = \{(x, y) : R(x, y) = 2\}$). We experiment with three medium size graphs, *netsc*, *ba-m*, and *comm-m*, and ask to add one shortcut to each node (*node-centric policy*, $k = 1$) for variable sizes of the initiator set I and values of the threshold T. For each case, we simulate the propagation model on the augmented network several times (x100), each time computing the number of nodes that have adopted, and averaging over all instances. In the end, we report the average size of the sphere of influence, $SOI(\%)$, as percentage of the total number of nodes in the network for finer representation:

$$SOI(\%) = \frac{|SOI|}{|V|}.$$

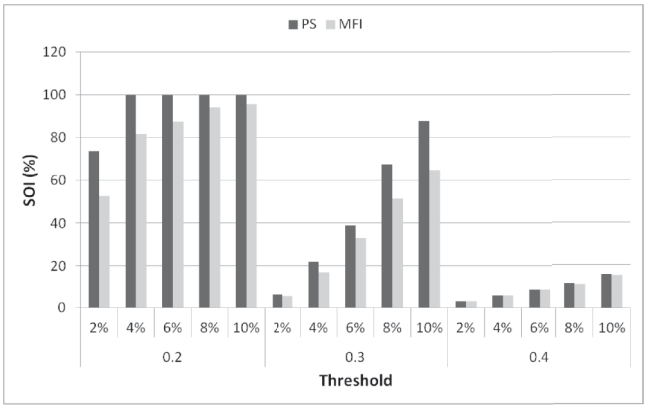
Figure 10 presents the results for the various networks. In all instances, *PS* outpeforms *MFI*. Note that the spheres of influence of the various initiator sets I (2%, 4%, 6%, 8%, 10% of the total number of nodes) in networks augmented by our *PS* method, are always larger than or equal to the ones in networks augmented by *MFI*—an increase that ranges between 0% and ∼80%. This behavior is demonstrated in all three networks, but the impact is larger in *comm-m* and *netsc*; the two networks that have a more profound clustering structure. On another note, as T varies (0.2, 0.3, 0.4), the sizes of the spheres of influence are getting smaller. This is expected as larger threshold indicates a less attractive behavior that is difficult to be adopted by nodes, so irrelevant of the structure of the augmented network, the behavior cannot easily cascade.

7. RELATED WORK

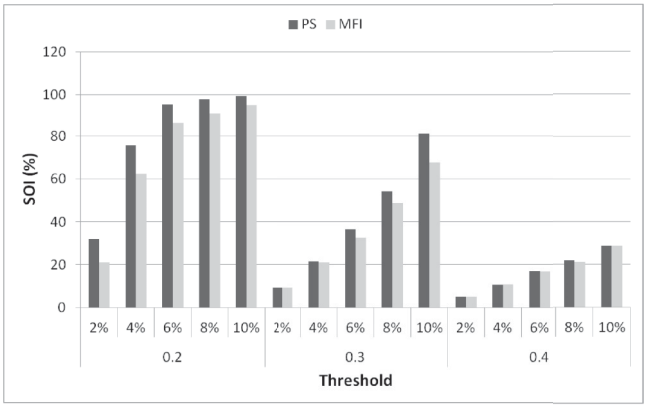
We have tried to provide pointers to work related to our research throughout the paper. In this Section, we provide a more concrete coverage of related work not mentioned earlier. In particular, the literature related to topics of graph augmentation, information propagation, and link prediction and recommendation.

7.1. Graph Augmentation

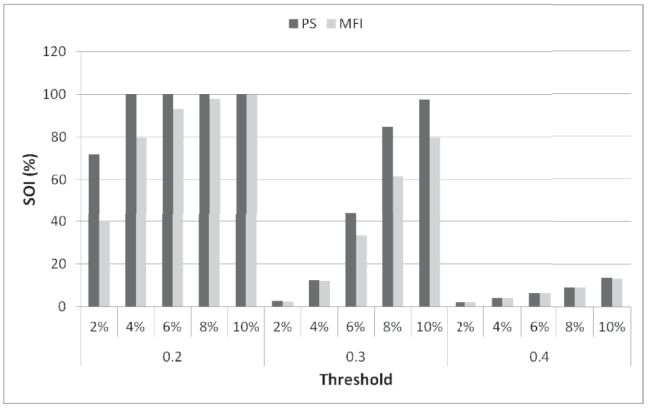
In traditional graph augmentation problems, we ask to find a minimum-cost set of edges to add to a graph to satisfy a specified property, such as biconnectivity,



(a) netsc



(b) ba-m



(c) comm-m

Fig. 10. Impact on information propagation spread.

bridge-connectivity or strong connectivity. These augmentation problems have been shown to be NP-complete in the restricted case of connected graphs [Eswaran and Tarjan 1976], while a number of approximation algorithms with favorable time complexity have been shown to have constant worst-case performance ratios [Frederickson and JáJá 1981; Khuller and Thurimella 1993; Nutov 2005]. In one of the first works that considers the problem with a hard limit on the number of edges to be added, Meyerson and Tagiku [Meyerson and Tagiku 2009] considered the problem of minimizing the average distance between the nodes. They obtained several constant-factor approximations using the k-median with penalties problem. They also improved the best known approximation ratio for metric k-median with penalties, to obtain better approximation factors for the other problems they considered. If α denotes the best approximation known for metric k-median with penalties, they presented an α-approximation for the single-source average-shortest-path problem, and a 2α-approximation for the general average shortest-path problem. In an another recent work, Demain and Morteza [Demaine and Zadimoghaddam 2010] studied the problem of minimizing the diameter of a graph by adding k shortcut edges, for speeding up communication in an existing network design. They developed constant-factor approximation algorithms for different variations of the problem and showed how to improve the approximation ratios using resource augmentation to allow more than k shortcut edges. Both [Meyerson and Tagiku 2009] and [Demaine and Zadimoghaddam 2010] observe a close relation between the single-source version of the problem, in which we want to minimize the largest distance from a given source vertex, and the well-known k-median problem. In the single-source version of the problem, a node may want to construct edges in order to minimize its distance from the other nodes. Laoutaris et al. [2008] studied the *bounded budget connection game*, where nodes in a network have a budget for purchasing links with the objective to minimize their average distance to the other nodes. A version of the problem for minimizing the maximum distance to the other nodes was considered as well. In a similar context, Zhou et al. [2013] develop network engineering algorithms that aim to provide performance gains of random walk simulations over networks by reducing their mixing time.

The problem of interest in this paper, is not the same as the aforementioned research. In the traditional graph theory, connectivity (biconnectivity, triconnectivity, etc.) of a finite undirected graph refers to the minimum number of disjoint paths that can be found between any pair of vertices, as characterized by *Menger's theorem* [Menger 1927] and generalized by the *max-flow min-cut theorem* [Papadimitriou and Steiglitz 1998]. In our research, the focus is on bringing nodes closer to each other, a connectivity property that is better expressed as *the characteristic path length* of a graph. It is also different to the theoretical approach of Meyerson and Tagiku [2009] as the suggested method operates by always attaching shortcuts to a single node, leading to a star network topology; in our problem, we need to be able to control where shortcuts are added in the network. In the rest of the approaches, variations of the problem of interest are considered, where either the context of the nodes needs to be known (assumed) or the emphasis is on optimizing network properties different to its characteristic path.

Our research stems from and extends on Papagelis et al. [2011], where the problem of adding k shortcuts to a graph in order to minimize its characteristic path length was originally introduced. Interesting variations of the problem exist, as well. In Lazaridou et al. [2015], the authors consider the problem of identifying the set of the top-k pairs of nodes in a graph whose distances are reduced the most as the result of adding new edges. In Parotisidis et al. [2015], the authors suggest methods for solving the problem when a more manageable subset of the candidate edges is considered (sublinear in the size of the graph), while the candidate set in our problem consists of all nonexistent edges (quadratic in the size of the graph).

7.2. Maximizing Information Propagation

The idea of making structural changes in a network to enable information to travel easier from a node to another is closely related to the idea of finding influential nodes in a network to maximize information spread [Chaoji et al. 2012; Kempe et al. 2003; Kleinberg 2007; Richardson and Domingos 2002] or detecting outbreaks in a network [Leskovec et al. 2007]. Manipulating edges to control information spread is also the topic in [Tong et al. 2012]. In the work of Richardson and Domingos [Richardson and Domingos 2002], the problem of finding the most influential set of k nodes in a network was introduced, motivated by viral marketing. Viral marketing refers to marketing techniques that aim to increase a marketing objective (such as brand awareness) through replicating an epidemic process in social networks, similar to the diffusion of innovations. It is useful in applications that employ the diffusion process, such as maximizing the spread of influence in a social network [Kempe et al. 2003] and early detection of outbreaks in a network [Leskovec et al. 2007].

The problem of interest in this paper, is not the same as the problem of finding influential nodes in the network, as we focus on importance of edges and not nodes. More importantly, we do not make any assumption about the context of the nodes or factors that can affect an information propagation process or model (e.g., set of initiators or early adopters, how influential is a node to its neighbors, how susceptible is a node to its neighbors' influence and so on). On the contrary, our problem tries to improve on information propagation processes, by operating directly on the network structure.

7.3. Link Prediction and Recommendation

Our problem draws connections to the problem of *link prediction* [Backstrom and Leskovec 2011; Liben-Nowell and Kleinberg 2003; Popescul et al. 2003], in which the objective is to predict which new interactions among members of a social network are likely to occur in the near future. Backstrom and Leskovec [2011] study the problem of link prediction and recommendation in social networks by performing supervised random walks on the network. Their method combines network structural information with rich node and edge attribute data to guide random walks. In a similar context, Tian et al. [2010] study the *link revival* problem, where the objective is to turn already existing edges with a few interactions to be more active so that the resulted connection will improve the social network connectivity. The authors develop an algorithm that explores local properties of a an evolving graph and try to recommend links per node at a time. The basic idea of their algorithm is to first consider graph snapshots in distinct time intervals, by monitoring the social interaction graph. Then, predict the probability with which a social interaction will happen in the future and if it has a low probability to happen, then they try to estimate the gain in the connectivity by adding this edge and consider it for recommendation. Finally, they recommend edges that have the larger gain in connectivity.

The problem of interest in this paper, is not the same as the link prediction and recommendation problems described previously. We are not interested in using historical data of interactions to predict future links or to revive existing but weak links. Instead, we want to suggest new edges that minimize global structural properties of the network by operating solely on the network structure.

8. CONCLUSIONS

We considered the problem of adding a small number of shortcuts to a graph in order to optimize a connectivity property and improve its capacity of carrying on social processes. This is a quite novel, interesting, and challenging problem. We proposed a novel method for quickly evaluating the importance of candidate shortcuts in a graph.

More specifically, our method is able to approximate the number of shortest paths that would run along a candidate shortcut, if it was already in the network. Intuitively, our method approximates the betweenness centrality of these nonexisting edges [Newman 2005]. Betweenness centrality of an edge or simply edge betweenness is the number of shortest paths between pairs of nodes that run along it and is a measure of the influence of an edge over the flow of information among nodes, especially in cases where information flow over a network primarily follows the shortest available path [Girvan and Newman 2002; Newman 2005]. We proposed, as well, a sampling-based variant of the method that can be used to scale up the computation for larger graphs. Through experiments on various datasets, we demonstrated that our approach outperforms sensible baselines in both accuracy and efficiency, for a varying range of conditions.

Overall, the algorithms we described are *simple* to understand and implement, accurate, very *fast,* and *general*, so they can probably be easily adopted in a variety of strategies. As such, we expect our methods to be beneficial in diverse settings and disciplines, ranging from social to technological networks. Later, we discuss a few ideas related to our research that aim to enlarge or prolong its scope. To some degree they present interesting extensions to our work and offer alternative views and insights.

8.1. Data Parallelism

It is important to note that further speedup of our methods can be achieved by technologies that distribute data and computing tasks across different parallel computing nodes. Data parallelism is based on distributing the data and computing tasks across different parallel computing nodes.

The first component of our main algorithm, based on a variation of Johnson's algorithm, is easy to distribute because it runs Dijkstra's algorithm n times, one for each node, which can be done in parallel. For example, all-pairs shortest paths can be computed by employing distributed versions of *breadth-first* or *depth-first* algorithms [Awerbuch 1985; Awerbuch and Gallager 1985; Chandy and Misra 1982]. New programming models for processing large datasets with a parallel, distributed algorithm on a cluster, such as MapReduce [Dean and Ghemawat 2004] and Hadoop [Shvachko et al. 2010], can be used, as well, to compute the all-pairs shortest paths component of our main algorithm: the *Map()* procedure performs multiple parallel breadth-first searches simultaneously assuming source nodes $\{s_0, s_1, \ldots, s_n\}$ and instead of emitting a single distance, emits an array of distances with respect to each source, as well as, information about the nodes that constitute each path. Then, the *Reduce()* procedure selects a path with minimum length for each element from the array.

The second component of our main algorithm is to process the list of shortest paths (path screening) in parallel in order to compute the utility of each candidate shortcut. Algorithm 3 and Algorithm 4 show how to employ MapReduce in this step, as well: the *Map()* (Algorithm 3) performs the path screening of each shortest path in isolation and computes the utility of any candidate shortcut that occurs in the current path. Then, the *Reduce()* (Algorithm 4) performs a summary operation that computes the utility scores of candidate shortcuts among all shortest paths. We do not further elaborate on this issue due to lack of space and as it is orthogonal to our problem.

8.2. Disconnected Networks

We have constrained our discussion to connected networks. In the case of disconnected networks, one should first consider finding the connected components of the input network and then connect them with each other. Finding a smallest augmentation that connects an undirected graph is a well-known problem, this is the problem of finding a spanning tree, with many efficient algorithms available.

ALGORITHM 3: Data Parallelism Using MapReduce − *Map*()

Input: An integer i representing a batch of shortest paths
Output: Partial utility of any candidate shortcut found in the ith batch

for *each shortest path in the ith batch* **do**
  $(x, y) \longleftarrow$ a candidate shortcut that occurs in a path;
  $\delta \longleftarrow$ the range of the shortcut in G ($\delta = R(x, y)$);
  produce one output record $< (x, y), \delta >$;
end

ALGORITHM 4: Data Parallelism Using MapReduce − *Reduce*()

Input: Set of candidate shortcuts (x, y)
Output: The utility of each candidate shortcut

for *each candidate shortcut* $< (x, y), \delta >$ **do**
  accumulate in U the sum of δ;
end
produce one output record $< (x, y), U >$;

8.3. Friend Suggestion Implications

The paper also highlighted the fact that the global optimization problem of augmenting a graph by adding new edges can be seen as an alternative for *friend suggestion* in a social network. The premise is that in the augmented network users are more well connected and information can spread more efficiently in the network. In this Section, we detail a methodology to evaluate such an alternative and relevant implications.

Current state-of-the-art friend suggestion algorithms are mainly based on the idea of suggesting friends with whom an individual shares a large number of mutual friends (similar to the *MFI* algorithm we presented in Section 6); that way the social network tries to maximize the probability of a new connection to occur (be realized). However, from a graph perspective, such an algorithm operates only locally by making the local network (cluster) more dense, but does not succeed in optimizing global graph properties (such as better information propagation). An alternative friend suggestion algorithm would try to blend ideas of our graph augmentation algorithms and the current-state-of-the-art by suggesting a *set of mixed friends* including:

—friends that make the local network more dense (coming from current state-of-the-art friend suggest algorithms), and
—friends that try to optimize global network properties (coming from our graph augmentation algorithms).

It becomes clear that many hybrid strategies can be followed to combine and evaluate the aforementioned algorithms. The evaluation should seek to optimize on the number of friends suggested by each algorithm and on the best way to rank the suggested friendships before presenting them to the end user. We do not further elaborate on this issue due to lack of space and as it is orthogonal to our problem.

Another important aspect of the friend suggest problem is that while we suggest friends, we cannot be sure that these friendships will be really materialized (i.e., accepted by the users). Therefore, we have to reason in probabilistic terms. The probability model is based on the assumption that for each candidate shortcut (x, y), there is an associated probability $p(x, y)$ for it to be realized once suggested. The probability p can be given by any simple prediction method (e.g., it can be based on the number of

mutual friends between two users). Then, the objective is to suggest a set of edges that will maximize the *expected utility*, if added in the graph.

ACKNOWLEDGMENTS

We are thankful to Francesco Bonchi and Aristides Gionis for thoughtful and stimulating discussions and Nick Koudas for providing constructive feedback and proof-reading draft versions of the manuscript.

REFERENCES

Noga Alon. 1986. Eigenvalues and expanders. *Combinatorica* 6, 2, 83–96.

B. Awerbuch. 1985. A new distributed depth-first-search algorithm. *Inform. Process. Lett.* 20, 3, 147–150.

Baruch Awerbuch and Robert G. Gallager. 1985. Distributed BFS algorithms. In *FOCS*, 250–256.

Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, 635–644.

Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439, 509–512.

D. Bu, Y. Zhao, L. Cai, H. Xue, X. Zhu, H. Lu, J. Zhang, S. Sun, L. Ling, N. Zhang, G. Li, and R. Chen. 2003. Topological structure analysis of the protein-protein interaction network in budding yeast. *Nucl. Acids Res.* 31, 9, 2443–2450.

K. Mani Chandy and Jayadev Misra. 1982. Distributed computation on graphs: shortest path algorithms. *ACM Commun.* 25, 11, 833–837.

Vineet Chaoji, Sayan Ranu, Rajeev Rastogi, and Rushi Bhatt. 2012. Recommendations to boost content spread in social networks. In *WWW*. 529–538.

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. In *OSDI*. 107–113.

Erik D. Demaine and Morteza Zadimoghaddam. 2010. Minimizing the diameter of a network using shortcut edges. In *SWAT*. 420–431.

E.W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik* 1, 1, 269–271.

D. Easley and J. Kleinberg. 2010. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, Cambridge, U.K.

Kapali P. Eswaran and Robert Endre Tarjan. 1976. Augmentation problems. *SIAM J. Comput.* 5, 4, 653–665.

Uriel Feige. 1998. A threshold of ln n for approximating set cover. *J. ACM* 45, 4, 634–652.

Yuval Filmus and Justin Ward. 2012. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *FOCS*. 659–668.

Greg N. Frederickson and Joseph JáJá. 1981. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.* 10, 2, 270–283.

Michael L. Fredman and Robert Endre Tarjan. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* 34, 3, 596–615.

M. Girvan and M. E. J. Newman. 2002. Community structure in social and biological networks. *Proc. Nat. Acad. Sci.* 99, 12, 7821–7826.

Andrew V. Goldberg and Chris Harrelson. 2005. Computing the shortest path: A search meets graph theory. In *SODA*. 156–165.

Mark Jerrum and Alistair Sinclair. 1989. Approximating the permanent. *SIAM J.* 18, 6, 1149–1178.

Donald B. Johnson. 1977. Efficient algorithms for shortest paths in sparse networks. *J. ACM* 24, 1, 1–13.

Srikanth Kandula and Ratul Mahajan. 2009. Sampling biases in network path measurements and what to do about it. In *IMC*. 156–169.

Michael Kapralov, Ian Post, and Jan Vondrak. 2013. Online submodular welfare maximization: Greedy is optimal. In *SODA*. 1216–1225.

George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 1, 359–392.

David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD*. 137–147.

Samir Khuller and Ramakrishna Thurimella. 1993. Approximation algorithms for graph augmentation. *J. Algorithms* 14, 2, 214–225.

Jamie King. 2003. Conductance and rapidly mixing Markov chains. Technical report, University of Waterloo, 2003.

Jon Kleinberg. 2007. Cascading behavior in networks: Algorithmic and economic issues. In *Algorithmic Game Theory*, Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani (Eds.). Cambridge University Press, Cambridge, U.K.

J. Kleinberg, A. Slivkins, and T. Wexler. 2004. Triangulation and embedding using small sets of beacons. In *FOCS*. 444–453.

Nikolaos Laoutaris, Laura J. Poplawski, Rajmohan Rajaraman, Ravi Sundaram, and Shang-Hua Teng. 2008. Bounded budget connection (BBC) games or how to make friends and influence people, on a budget. In *PODC*. 165–174.

Konstantina Lazaridou, Konstantinos Semertzidis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2015. Identifying converging pairs of nodes on a budget. EDBT. 313–324.

Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 631–636.

Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In *KDD*. 420–429.

Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. (June 2014). Retrieved from http://snap.stanford.edu/data.

Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2008. Statistical properties of community structure in large social and information networks. In *WWW*. 695–704.

David Liben-Nowell and Jon Kleinberg. 2003. The link prediction problem for social networks. In *CIKM*. 1019–1031.

László Lovász. 1993. Random walks on graphs: a survey. *Combinatorics, Paul erdos is eighty* 2, 1 (1993), 1–46.

David Lusseau, Karsten Schneider, Oliver J. Boisseau, Patti Haase, Elisabeth Slooten, and Steve M Dawson. 2003. The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. *Beh. Ecol. Sociobiol.* 54, 4, 91–125.

Karl Menger. 1927. Zur allgemeinen kurventheorie. *Fundamenta Mathematicae* 10, 1, 96–115.

Adam Meyerson and Brian Tagiku. 2009. Minimizing average shortest path distances via shortcut edge addition. In *APPROX-RANDOM*.

Abedelaziz Mohaisen, Aaram Yun, and Yongdae Kim. 2010. Measuring the mixing time of social graphs. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. 383–389.

Stephen Morris. 2000. Contagion. *Rev. Economic Stud.* 67, 1, 57–78.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions-I. *Math. Program.* 14, 1, 265–294.

M. E. J. Newman. 2005. A measure of betweenness centrality based on random walks. *Soc. Netw.* 27, 1, 39–54.

M. E. J. Newman. 2006. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E* 74, 3, 036104.

Zeev Nutov. 2005. Approximating connectivity augmentation problems. In *SODA*. 176–185.

Christos H. Papadimitriou and Kenneth Steiglitz. 1998. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, Mineola, New York.

Manos Papagelis, Francesco Bonchi, and Aristides Gionis. 2011. Suggesting ghost edges for a smaller world. In *CIKM*. 2305–2308.

Manos Papagelis, Gautam Das, and Nick Koudas. 2013. Sampling online social networks. *IEEE Trans. Knowl. Data Eng.* 25, 3, 662–676.

N. Parotisidis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2015. Selecting shortcuts for a smaller world. In *SIAM International Conference on Data Mining (SDM)*. 28–36.

Alexandrin Popescul, Rin Popescul, and Lyle H. Ungar. 2003. Statistical relational learning for link prediction. In *IJCAI03 Workshop on Learning Statistical Models from Relational Data*. 109–115.

Miao Qiao, Hong Cheng, Lijun Chang, and Jeffrey Xu Yu. 2012. Approximate shortest distance computing: A query-dependent local landmark scheme. In *ICDE*. 55–68.

Matthew Richardson and Pedro Domingos. 2002. Mining knowledge-sharing sites for viral marketing. In *KDD*. 61–70.

K. Shvachko, H. Kuang, S. Radia, and R. Chansler. 2010. The Hadoop distributed file system. In *MSST*. 1–10.

Alistair Sinclair. 1992. *Improved Bounds for Mixing Rates of Markov Chains and Multicommodity Flow*. Springer, New York.

Yuan Tian, Qi He, Qiankun Zhao, Xingjie Liu, and Wang-chien Lee. 2010. Boosting social network connectivity with link revival. In *CIKM*. 589–598.

Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2012. Gelling, and melting, large graphs by edge manipulation. In *CIKM*. 245–254.

Jan Vondrak. 2008. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*. 67–74.

Zhuojie Zhou, Nan Zhang, Zhiguo Gong, and Gautam Das. 2013. Faster random walks by rewiring online social networks on-the-fly. In *ICDE*. 769–780.

Received May 2014; revised February 2015; accepted April 2015