Selecting Shortcuts for a Smaller World

Nikos Parotsidis[*] Evaggelia Pitoura[*] Panayiotis Tsaparas[*]

Abstract

The small world phenomenon is a desirable property of social networks, since it guarantees short paths between the nodes of the social graph and thus efficient information spread on the network. It is thus in the benefit of both network users and network owners to enforce and maintain this property. In this work, we study the problem of finding a subset of k edges from a set of candidate edges whose addition to a network leads to the greatest reduction in its average shortest path length. We formulate the problem as a combinatorial optimization problem, and show that it is NP-hard and that known approximation techniques are not applicable. We describe an efficient method for computing the exact effect of a single edge insertion on the average shortest path length, as well as several heuristics for efficiently estimating this effect. We perform experiments on real data to study the performance of our algorithms in practice.

1 Introduction

In the past decade there has been an explosive growth of data in the form of networks. An important property of such networked data is the *small world phenomenon*: even in networks of large size, the average shortest path length between two nodes is small. This is a highly desirable property since it facilitates and amplifies dynamic processes in the network, such as information dissemination, or viral marketing. A well connected network is of value to both the users and the owners of the networked data, and thus it is in their best interest to enforce and maintain this property.

Motivated by such considerations, in this paper, we consider the following novel network analysis problem. Given a graph G representing a network, and a set of candidate edges C not present in the graph, we look for a small subset of edges in C that have the greatest impact on G, i.e., if added to G, they would cause the largest decrease in the average shortest path length, thus making G a smaller world. We call such edges *shortcut edges* and the problem of identifying the best set of k such edges the SHORTCUTSELECTION problem.

[*]Department of Computer Science & Engineering, University of Ioannina, Greece. E-mail: {nparotsi,pitoura,tsap}@cs.uoi.gr.

The SHORTCUTSELECTION problem has several applications. Take for example a graph G that evolves over time. This can be a social network, where new friendships are constantly created, or a criminal network where new associations are formed between known criminals. The candidate set C is the set of new edges added between two successive network snapshots, and the shortcut set consists of the subset of new edges that have the greatest effect in bringing the nodes of the graph G closer together. These edges convey important information about the graph evolution, since they provide a summary of the important changes that happened in the graph. For a social network, these are the connections that the network owner would like to proactively nurture and maintain. For a terrorist network, these are the links that need to be investigated.

Furthermore, the SHORTCUTSELECTION problem has applications in physical networks, such as transportation networks, or communication networks. In such networks, creating new connections between nodes (building new roads, laying cables) is limited by physical constraints, and the selection of the new connections requires careful consideration. In all such cases we want to select a few connections that will maximize the overall benefit by making the network a smaller word.

Finally, SHORTCUTSELECTION may also be seen as an amendment to link recommendations. Let C be a set of candidate link recommendations computed using any of the many related algorithms [1, 8]. So far, link recommendations are selected based solely on their prediction accuracy, that is, C consists of the links with highest prediction "score". Selecting from C a subset S of link recommendations that improve the overall "quality" of the network (by making it a smaller world) introduces a new dimension to the link prediction problem, where the benefit of the network owner is also taken into account.

In this paper, we formalize the SHORTCUTSELECTION as a combinatorial optimization problem. We show it is NP-hard, and we demonstrate why known approximation techniques are not applicable. We study in detail the case of a single edge addition, and we propose an algorithm for computing the exact reduction of the average shortest path. The algorithm provides a characterization of the structure of the affected nodes.

Using this characterization, we propose a heuristic that computes an estimate of the effect of a single edge, and it is able to scale for graphs of large size. We evaluate our algorithms in terms of both effectiveness and efficiency on a variety of real datasets, and we compare with simple baselines, as well as competing approaches.

The remainder of the paper is structured as follows. In Section 2 we survey some of the related work in the literature. In Section 3, we define the problem, and study it theoretically. In Section 4, we consider the single edge addition, while in Section 5 we propose algorithms for our problem. In Section 6, we present an experimental evaluation on real graphs of the proposed methods. Finally, Section 7 concludes the paper.

2 Related work

The work most closely related to ours is that of Papagelis et al., [12], which considers the problem of selecting k edges among all possible edges not present in the graph, such that the sum of all-pairs shortest path distances is minimized. The main difference with our work is in the set of candidate edges that we consider. Although this difference may seem superficial, it is of critical importance to the design of algorithms for the problem. The candidate set in [12] is of size quadratic in the size of the graph, while we assume a manageable set of candidate edges (sub-linear in the size of the graph). Considering all missing edges as candidates is highly restrictive. No algorithm that performs some computation per candidate edge can scale for real graphs of large size. This restriction guides the approach in [12]. As we will show in Section 6, even after refinements that we make, their algorithm is not as efficient or effective when applied to the shortcutSelection problem.

The average shortest path length is related to the average *closeness centrality* of nodes in the graph. In [4] they consider the problem of finding the k edges whose addition to the graph maximizes the centrality of a *specific* target node. In [13] an incremental algorithm is proposed to efficiently update the closeness centrality values under changes in network topology. In [9] the problem is to find the k edges, incident to a specific node, whose addition leads to the greatest reduction in all-pairs shortest paths distances. In [15] they seek the k edges whose addition will best enable the propagation of information in the network under the SIS model. In [5] they identify the set of top-k pairs of nodes whose distances are reduced the most as the result of edge additions.

The idea of combining shortcut selection with link recommendations is explored in [7], where the authors re-rank link recommendations in order to promote information diffusion in a social network. Also, in [3] the authors take as input a set of recommended links and compute k edges per node that boost content spread in the network.

3 Problem Definition and Hardness

Let $G = (V, E)$ be a connected unweighted undirected graph representing a network. For a pair of nodes $x, y \in V$ let $d(x, y)$ denote the *shortest path distance* between the nodes, that is, the length of the shortest path between x and y in the graph. We define the *average shortest path length* in the graph G as

$$L(G) = \frac{1}{\binom{n}{2}} \sum_{x,y \in V} d(x, y)$$

where n denotes the size (number of nodes) of the graph. The value of $L(G)$ is a commonly used measure for the small world property in a network.

Now, let $C \subseteq V \times V$ denote a set of edges not in the graph G, i.e., $C \cap E = \emptyset$. These are the *candidate* edges to be added to the graph. The addition of any edge to the graph G can only reduce the value of the average shortest path length. We are interested in finding a subset $S \subseteq C$ of k edges whose addition to a graph G leads to the greatest reduction in $L(G)$. Abusing the notation, let $G' = G \cup S$ denote the graph G after the addition of a set of edges S. We define $R_G(S) = L(G) - L(G \cup S)$ to be the reduction in the average shortest path length caused by the addition of the edge set S in the graph G. We want the set S of size k that maximizes $R_G(S)$. Abusing the notation we will sometimes use $R_G(S)$ to denote the reduction in the *sum* of shortest path lengths. This has no effect in our problem definition.

We define our problem formally as follows.

Problem 1. (shortcutSelection) *Given a connected, unweighted, undirected graph $G = (V, E)$, a set $C \subseteq (V \times V) \setminus E$ of candidate edges, and an integer value k, find a subset of candidate edges $S \subseteq C$ of size $|S| = k$ that maximizes the reduction $R_G(S)$ in the average shortest path length.*

We will now show that shortcutSelection is *NP-hard*. For the proof we will show a reduction from the maxCoverage problem, which is known to be NP-hard. The maxCoverage problem is defined as follows. We are given as input a universe $U = \{u_1, u_2, ..., u_n\}$ of n elements, and a collection $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ of subsets of the elements of U, such that $\cup_{i=1}^{m} S_i = U$. We say that a set $S_i \in \mathcal{S}$ covers an element u if $u \in S_i$. We define the coverage of a sub-collection $\mathcal{V} \subseteq \mathcal{S}$, as $c(\mathcal{V}) = | \cup_{S_i \in \mathcal{V}} S_i|$. Given a budget value k, the maxCoverage problem asks for a collection $\mathcal{V}$ of size k that maximizes the coverage $c(\mathcal{V})$.

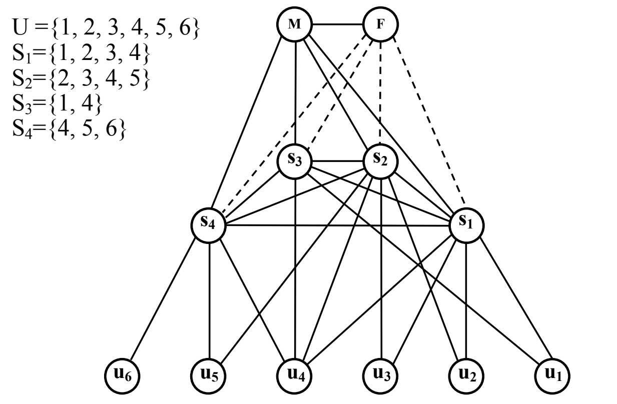


Figure 1: The maxCoverage instance and the corresponding shortcutSelection instance.

Theorem 3.1. *The* shortcutSelection *problem is* NP-hard.

Proof. [Sketch] Given an instance of the maxCoverage problem, we will construct an instance of the shortcutSelection problem. Let $(U, \mathcal{S})$ be the input to the maxCoverage problem, where $U = \{u_1, u_2, ..., u_n\}$ is the universe of elements, and $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ a collection of subsets. We construct the input (G, C) to the shortcutSelection problem as follows. The node set V of the graph G consists of $n+m+2$ nodes: (i) A node u_i for each element $u_i \in U$; (ii) A node s_i for each set $S_i \in \mathcal{S}$; (iii) Two additional nodes F and M. The set of edges E consists of the following edges: (i) An edge (s_i, u_j) for every $u_j \in S_i$; (ii) An edge (s_i, s_j) for every pair of nodes s_i, s_j; (iii) Edges (M, s_i) for every node s_i; (iv) The edge (F, M). Finally, the set of the candidate edges C consists of all edges (F, s_i), for all nodes s_i. An example of the construction is shown in Figure 1.

Note that in the original graph (before any candidate edges are added) node M acts as an intermediate between F and the nodes s_i. Therefore, for all s_i, $d(F, s_i) = 2$, and for all u_i, $d(F, u_i) = 3$. Consider now the addition of an edge (F, s_i) to the original graph. This has the effect of reducing by one the length of the shortest path between F and s_i, and between F and all the nodes u_p connected to s_i. These correspond to elements u_p that are *covered* by the set S_i. Therefore, when adding an edge (F, s_i), every pair of the type (F, u_p) whose distance is decreased by one corresponds to an element covered by the set S_i. The total reduction in the shortest paths by adding a set of k edges $E_k = \{(F, s_{i_1}), ..., (F, s_{i_k})\}$ is equal to the coverage of the sub-collection $\mathcal{V} = \{S_{i_1}, ..., S_{i_k}\}$ plus k. Maximizing $R_G(E_k)$ results in maximizing coverage $c(\mathcal{V})$.

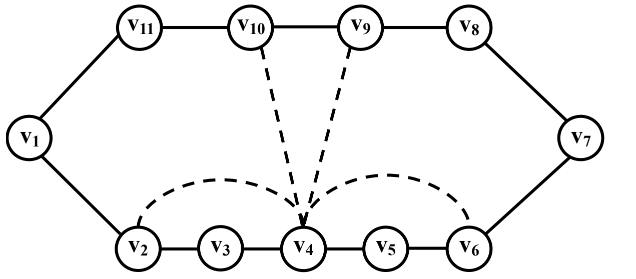


Figure 2: The counter-example for the non-submodularity and non-supermodularity of $R_G(S)$. The dashed edges are the candidate edges.

Since our problem is NP-hard, we look for approximation algorithms with provable guarantees. A common approach to handling similar problems, e.g., the maxCoverage problem, is by showing that the function to be maximized is submodular. Let Ω be a set, and let 2^Ω denote the power set of Ω. The set function $f : 2^\Omega \to \mathbb{R}$ is submodular, if for every $X, Y \subseteq \Omega$ with $X \subseteq Y$ and every $x \in \Omega$ it holds that $f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$. It is well known that when maximizing submodular set functions under cardinality constraints a simple greedy algorithm gives a constant factor approximation [11]. Unfortunately, as we show below, the reduction function $R_G(S)$ is not submodular. In fact, we show that the reduction function is also not supermodular. A function f is supermodular if for every $X \subseteq Y \subseteq \Omega$, and $x \in \Omega$, $f(X \cup \{x\}) - f(X) \leq f(Y \cup \{x\}) - f(Y)$.

Lemma 3.1. *The reduction function $R_G(S)$ is neither submodular nor supermodular.*

Proof. For the proof it suffices to give a counter-example for each case, where the property does not hold. We begin with the proof for non-submodularity. Consider the graph shown in Figure 2, consisting of the solid edges that form a cycle of eleven nodes. The dashed edges are the candidate edges. Consider now the effect of the addition of edges, (v_2, v_4) and (v_4, v_6). Going through all pairs, one can see that the addition of either edge by itself causes the length of the shortest paths between ten different pairs to decrease by one. However, if the edge (v_4, v_6) has already been added, the addition of (v_2, v_4) causes the distance of thirteen pairs to be reduced by one.

To understand the intuition behind this counter-example, consider the pair (v_1, v_7). When no candidate edges have been added, the shortest path between the pair (v_1, v_7) follows the top part of the cycle, and it has

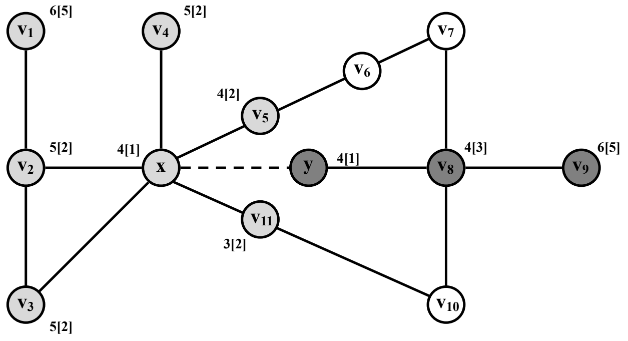


Figure 3: An example of the sets A_x and A_y for the insertion of the edge (x, y). Nodes in A_y are colored light gray, and nodes in A_x are colored dark gray. White nodes belong to neither set. The number outside the bracket denotes the length of the previous shortest path, while the number in the bracket is the length of the new shortest path after the insertion of (x, y).

length 5. The addition of the edge (v_4, v_6) creates a new shortest path of length 5 that follows the bottom part of the cycle. The distance between v_1 and v_7 remains unchanged. However, if we also add the edge (v_2, v_4), the lower shortest path has now length 4, reducing the distance between v_1 and v_7 by one. This reduction is achieved by the synergy of the two edges; the existence of the one in the graph amplifies the effect of the other. This effect is what makes the function non-submodular.

For the proof of non-supermodularity, we consider the candidate edges (v_{10}, v_4) and (v_9, v_4) in Figure 2. Adding the edge (v_{10}, v_4) to G results in a reduction of 23 in the sum of all-pairs shortest paths. From the symmetry in the graph, adding the edge (v_9, v_4) results in the same reduction. However, the insertion of (v_9, v_4) to the graph $G \cup \{(v_{10}, v_4)\}$ results in reduction equal to 7, smaller than that of adding (v_9, v_4) to G. To obtain some intuition, it is instructive to consider the pair (v_9, v_4). The addition of (v_{10}, v_4) has already cut the distance between v_9 and v_4 significantly. The addition of (v_9, v_4) supplements the effect of (v_{10}, v_4) rather than amplifying it, reducing the path by just one hop.

4 Computing the effect of a single edge

In this section, we describe a method for calculating the effect of a single edge addition on the all-pairs shortest paths. This algorithm is a modification of an incremental computation of shortest path distances in directed graphs, presented in [2]. The method consists of two phases. In the first phase, we identify affected pairs of nodes whose distance (shortest path length) is potentially reduced by the addition of the new edge.

Algorithm 1 *EdgeEffect* Algorithm

Require: Graph G, edge $e = (x, y)$
Ensure: Edge effect $R_G(e)$

```
1:  Compute BFS trees $T(x)$ and $T(y)$
2:  $A_x \leftarrow \{u \in T(y) : d_{old}(u,y) + 1 < d_{old}(u,x)\}$
3:  $A_y \leftarrow \{v \in T(x) : d_{old}(v,x) + 1 < d_{old}(v,y)\}$
4:  $R_G(e) = 0$
5:  for $u \in A_x$ do
6:      Compute BFS tree $T(u)$
7:      for $v \in T(u)$ do
8:          $d_{new}(u,v) = d_{old}(v,x) + d_{old}(u,y) + 1$
9:          if $d_{new}(u,v) < d_{old}(u,v)$ then
10:             $R_G(e) += d_{old}(u,v) - d_{new}(u,v)$
11:         end if
12:     end for
13: end for
14: return $R_G(e)$
```

In the second phase, we determine for which of the potentially affected pairs the distances were actually reduced, and we calculate this reduction.

Let $e = (x, y)$ be the edge to be added, and let $d_{old}(u, v)$ be the distance between two nodes u, v before the insertion of e. We define two sets of affected nodes, namely sets A_x and A_y, such that $A_x = \{u \in V : d_{old}(u, y) + 1 < d_{old}(u, x)\}$, and $A_y = \{v \in V : d_{old}(v, x) + 1 < d_{old}(v, y)\}$. For a node $v \in A_y$ the addition of edge (x, y) creates a path from v to y that goes through the edge (x, y) and has length $d_{old}(v, x) + 1$ which is shorter than the length $d_{old}(v, y)$ of the previous path from v to y. Similarly for a node $u \in A_x$ with respect to x. Therefore, the sets A_x and A_y are the sets of nodes for which the addition of the edge (x, y) causes their distance to x and y, respectively, to be reduced. For example in Figure 3, $A_x = \{y, v_8, v_9\}$ and $A_y = \{x, v_1, v_2, v_3, v_4, v_5, v_{11}\}$.

We can show that if the distance between a pair of nodes (u, v) has been reduced then u and v must belong to sets A_x and A_y, respectively. The proof of the following lemma is omitted due to space constraints.

LEMMA 4.1. *The insertion of edge (x, y) affects the distance of two nodes u and v, only if, either $u \in A_y$ and $v \in A_x$, or $v \in A_y$ and $u \in A_x$.*

Following Lemma 4.1 the potentially affected pairs are the ones in the cross-product $A_x \times A_y$. To find the truly affected pairs, and the effect of the edge (x, y), we need to check for which pairs their distance actually decreased.

Algorithm 1 shows the outline of the full algorithm for computing the reduction $R_G(e)$ caused by the addition of edge $e = (x, y)$. The algorithm first computes the sets A_x and A_y. This can be done by performing two BFS traversals from x and y. We store the resulting BFS trees $T(x)$ and $T(y)$. To compute A_x (resp. A_y) we traverse the tree $T(y)$ (resp. $T(x)$) and keep the nodes u for which $d_{old}(u, y) + 1 < d_{old}(x, u)$ (resp. $d_{old}(u, x) + 1 < d_{old}(y, u)$).

For a pair $(u, v) \in A_x \times A_y$, let $d_{xy}(u, v) = d_{old}(u, y) + d_{old}(v, x) + 1$ be the length of the path between u, v that goes through the edge (x, y). Assume without loss of generality that $|A_x| < |A_y|$ (as in Figure 3). In the second phase, we perform a BFS traversal for all the nodes $u \in A_x$. For every node v that we visit in the BFS tree of u, if $d_{xy}(u, v) < d_{old}(u, v)$, then we conclude that the distance between (u, v) was reduced, and we update the reduction $R_G(e)$ appropriately.

The time complexity of the proposed method is $O(mn)$, since in the worst case A_x and A_y have size $O(n)$, and the BFS traversal in step 6 takes $O(m)$. However, in practice the algorithm is significantly faster. Note that in the traversals of the BFS trees (lines 2, 3, 8) we can stop the exploration of a node if the condition we are interested in is not satisfied. This results in a significant speed-up of the algorithm.

5 Algorithms

In this section, we describe algorithms for the shortcutSelection problem. The main goal is to compute efficiently an estimate of the exact solution even for large-scale graphs. To be practical, our algorithms should aim for linear time and space complexity. With the exception of the greedy algorithm we describe below, the main idea behind all of our methods is to compute a *score* for each candidate edge in C and select the k edges with the highest scores. The difference between the various methods is in the score computation. The space complexity of all algorithms is $O(m + n)$.

We now describe our algorithms in detail.

***Greedy*:** The greedy algorithm selects each time the edge that causes the largest reduction. The algorithm maintains a set of edges $S \subseteq C$ selected so far, initialized to the empty set. It then proceeds iteratively, where at each iteration, it computes the exact effect $R_{G \cup S}(e)$ for each candidate edge $e \in C \setminus S$ using the *EdgeEffect* algorithm from Section 4. It selects the edge with the greatest gain and inserts it to S. The algorithm stops when k edges have been selected. The worst-case time complexity of the greedy method is $O(mnrk)$, where n and m is the number of nodes and edges respectively, and $r = |C|$. The factor mn comes from the worst-case complexity of *EdgeEffect*.

***EdgeEffect*:** In this method we use the *EdgeEffect* algorithm to compute the exact reduction $R_G(e)$ for each edge $e \in C$ in the candidate set, and we select the k edges with the greatest gain $R_G(e)$. The time complexity of the algorithm is $O(mnr)$, where $r = |C|$.

***EffectEstimation*:** This algorithm computes for each candidate edge $e \in C$ a score $\tilde{R}_G(e)$ which acts as an estimate of the exact reduction $R_G(e)$. The algorithm makes use of the following lemma, which we state here without proof due to space constraints. In the following, $d_{new}(u, v)$ denotes the distance between nodes u and v in the graph G after the insertion of the edge e.

Lemma 5.1. *Let (x, y) be a newly inserted edge and let $u \in A_x$ be a node which reduced its distance from x by i, and $v \in A_y$ be a node which reduced its distance from y by j. Then $d_{old}(u, v) - d_{new}(u, v) \leq \min\{i, j\}$.*

To compute the estimated effect $\tilde{R}_G(e)$, the algorithm makes use of the sets A_x, A_y described in Section 4, containing the set of nodes whose distance from x and y, respectively, was reduced. Given A_x and A_y, the algorithm computes the sets $A_x[i]$ and $A_y[j]$ for $1 \leq i, j < d_{old}(x, y)$, containing all the nodes in A_x and A_y whose distance from x and y was reduced by exactly i and j respectively. From Lemma 5.1, for all $u \in A_x[i]$,$v \in A_y[j]$, we have $d_{old}(u, v) - d_{new}(u, v) \leq \min\{i, j\}$.

Following the above discussion, one way to estimate the effect of the edge (x, y) would be to add to $\tilde{R}_G(e)$ the edge effect $|A_x[i]| \cdot |A_y[j]| \cdot \min\{i, j\}$, for all $1 \leq i, j < d_{old}(x, y)$. The problem is that this approximation is over-optimistic, and it does not take into account the alternative paths that exist between nodes in A_x and A_y, which in many cases are shorter than the new paths.

Consider a node $u \in A_x[i]$. The node u has an alternative path to x that does not pass through (y, x), that is, the shortest path to x before the insertion of (x, y). Since the new path to x goes through (y, x) we have $d_{new}(u, x) = d_{old}(u, y) + 1$. Since the new path is shorter that the old one by i, it follows that $d_{old}(u, x) = d_{old}(u, y) + 1 + i$. However, in approaching node x the old path must go through nodes in A_y. When i is small, for the nodes $v \in A_y$ that appear in the alternative path from u to x, the distance $d_{old}(u, v)$ will most likely not be reduced. That is, for smaller i, fewer nodes $v \in A_y$ reduce their distance from u. Things are completely symmetric when examining a node $v \in A_y[j]$ and its paths to nodes in A_x. Combining the above, for pairs of nodes (u, v), $u \in A_x[i]$,$v \in A_y[j]$, where i, j are both small, it is unlikely to have a reduction in their distance. In our algorithm, we assume that the distance of the pair is affected when $(i + j)$ is sufficiently large. As a result, we estimate the effect of the edge by adding $|A_x[i]| \cdot |A_y[j]| \cdot \min\{i, j\}$, for all $1 \leq i, j < d_{old}(x, y)$, such that $(i + j) > d_{old}(x, y)$.

Algorithm 2 *EffectEstimation* Algorithm

Require: Graph G, edge $e = (x, y)$
Ensure: Estimated edge effect $\tilde{R}_G(e)$

```
1: $\tilde{R}_G(e) = 0$
2: Compute $d_{old}(x,y)$, $A_x[i], A_y[i], 1 \leq i < d_{old}(x,y)$
3: if $d_{old}(x,y) > 2$ then
4:     for $1 \leq i < d_{old}(x,y)$ do
5:         for $d_{old}(x,y) - i < j < d_{old}(x,y)$ do
6:             $\tilde{R}_G(e) += |A_x[i]| \cdot |A_y[j]| \cdot \min\{i,j\}$
7:         end for
8:     end for
9: end if
10: if $d_{old}(x,y) = 2$ then
11:     $\tilde{R}_G(e) = |N_x \cap A_y| \cdot |N_y \cap A_x|$
12: end if
13: return $\tilde{R}_G(e)$
```

Note that for the edges (x, y), where $d_{old}(x, y) = 2$, we have $i, j \leq 1$, and therefore no score is computed. This is a special case that we need to take care of, since such edges may reduce distances (by one) between a large number of pairs and lead to a greater overall reduction than long-haul edges that reduce distances by a large number but affect only few pairs. To deal with this issue we assign a score to such edges equal to the product of the affected neighbors of x and y, that is, $|N_x \cap A_y| \cdot |N_y \cap A_x|$, where N_v consists of v and the neighbors of v. The full outline of our algorithm is shown in Algorithm 2.

We can compute the sets $A_x[i], A_y[j]$, $1 \leq i, j < d_{old}(x, y)$ during the computation of the sets A_x, A_y described in section 4. The complexity of the *EffectEstimation* algorithm is $O(rm)$, where $r = |C|$.

***PBG* algorithm:** This is the algorithm proposed by Papagelis, Bonchi and Gionis [12]. The *PBG* algorithm also approximates the effect of an edge, using the following idea: An edge (x, y) affects a shortest path between two nodes u and v only if it appears as a shortcut in their previous shortest path.

The algorithm in [12] was designed for scoring all missing edges in the graph and it works as follows. First, it computes and maintains the all-pairs shortest paths. Next, for each shortest path $p = \langle u, ..., v\rangle$, and for each non-adjacent pair of nodes x, y that appear in p, it adds to the estimated effect of (x, y) the quantity $d_{old}(x, y) - 1$. The implementation in [12] needs $O(n^2)$ space and $O(n^3)$ time. Since in the problem they consider the number of candidates is quadratic, this is a cost they can afford to pay.

We provide a more efficient implementation of the *PBG* algorithm for the SHORTCUTSELECTION problem which works as follows. For a node v in the input graph G we construct the BFS tree $T(v)$. We then go through the set C of candidate edges and for each edge $e = (x, y) \in C$ we perform an ancestor-descendant test to determine if x is an ancestor of y in $T(v)$, or y is an ancestor of x in $T(v)$. This can be done in constant time [14]. If x is an ancestor of y in $T(v)$ (symmetrically for y), then we add to the score of e the quantity $(d_{old}(x, y) - 1) \cdot desc(y)$, where $desc(y)$ is the number of the descendants of y (including y) in $T(v)$. This process takes time $O(m + r)$ for each node $v \in G$, where $r = |C|$, resulting in overall time complexity $O(mn + nr)$.

***Distance*:** For this algorithm the score of each candidate edge (x, y) is the distance $d_{old}(x, y)$ in the original graph. The algorithm has time complexity $O(mr)$.

***Degree*:** For this algorithm the score of each candidate edge (x, y) is the product of the degrees $deg(x) \cdot deg(y)$. The complexity of the algorithm is $O(r)$.

6 Experimental evaluation

In this section, we study experimentally the performance of the algorithms described in Section 5 in terms of both efficiency and effectiveness, using real graphs of different types and sizes.

6.1 Datasets. We use three different datasets in our experiments that correspond to three different types of networks. All three graphs contain information about the time when each edge appeared.

- The *Facebook* dataset [16], consists of a social graph from the Facebook New Orleans network. Nodes represent users, and edges user friendships. The dataset contains the activity of the network from September 2006 until February 2009.
- The *DBLP* dataset[1] [6] consists of a collaboration graph of authors of computer science papers where an edge between two authors represents a common publication. The *DBLP* graph contains the authors and co-authorships between 1970 and 2013.
- The *Internet* graph [10] consists of a graph defined over Internet AS and the connections between them. The *Internet* graph maps the connections that were established in years 2004, 2005 and 2006.

For our experiments we consider the scenario we described in Section 1, where we have consecutive snapshots of an evolving graph, and the candidate edges are the edges added between the two snapshots. For

[1]KONECT, http://konect.uni-koblenz.de/

Dataset	nodes	edges	candidates	diameter
Internet	17,474	31,579	5,250	12
DBLP	31,422	69,428	2,444	26
Facebook	41,212	342,584	12,921	19

Table 1: Characteristics of the three graph instances.

Method	*Internet*	*Facebook*	*DBLP*
APSP recomputation	33,190	381,429	129,226
EdgeEffect	14	1,314	496

Table 2: Average running time (measured in milliseconds) for computing the effect of a single edge insertion.

each dataset, we produce a series of input instances $\{G_i, C_i\}$ to our problem, where $G_i = (V_i, E_i)$ is the graph up to time t_i, and the set of candidate edges C_i is the set of edges added between snapshots G_i and G_{i+1}. More specifically, let $LCC(G_i)$ be the largest connected component of graph G_i. Then $C_i = \{(u, v) \in E_{i+1} : (u, v) \notin E_i, u, v \in LCC(G_i)\}$. The number of snapshots per dataset is selected such that the candidate set C_i is sufficiently large compared to the size of the graph G_i. For the *Internet* and *DBLP* datasets we took yearly snapshots, while for *Facebook* monthly. We obtained 3 input instances for *Internet*, 42 for *DBLP*, and 29 for the *Facebook* dataset.

We apply our algorithms to all input instances. In order to study the effectiveness of the algorithms in depth, we use for each dataset the "middle" instance, which we view as representative. For the *Internet* dataset this corresponds to the instance between years 2005 and 2006, for *DBLP* the instance between the years 1990 and 1991, and for *Facebook* the instance between November and December of 2007. For the rest of this Section any reference to graphs *DBLP*, *Facebook* and *Internet* will mean those three instances, unless we state explicitly that we refer to the full instance sequence. The characteristics of these three input instances are shown in Table 1.

6.2 Effectiveness. We first study the effectiveness of the methods. To measure effectiveness we compute $R_G(C)$, that is, the reduction we can obtain by adding the full set of candidate edges to the graph G. Then, given a selected set of candidate edges $S \subseteq C$, we compute the *reduction percentage* $R_G(S)/R_G(C)$, which we use as our effectiveness measure. Using this measure, we can compare the different algorithms, but also evaluate them independently. To avoid calibrating the value of k for each different dataset, we set the size of the selected set to be a fraction $|S| = \gamma|C|$ of the candidate edges, for $\gamma = 1\%, 2\%, 3\%, 4\%, 5\%$.

Figure 4 presents the reduction percentage of our algorithms for the three datasets as a function of the size of the selected set. We also include a *Random* selection as a lower bound, where the set S is chosen randomly. The *Greedy* algorithm performs the best, and defines an upper bound for the effectiveness of our methods. The *EdgeEffect* algorithm follows closely, consistently in all three datasets. Of the remaining heuristics, the algorithm that performs best is *EffectEstimation*. Notably, for the *DBLP* and *Facebook* graphs, there is a sizeable gap between the effectiveness of *EffectEstimation* and the next best one. There is no clear winner among the remaining heuristics. The *PBG* method performs better on the *Internet* graph, but it is outperformed by the *Distance* method on the *Facebook* and *DBLP* graphs. Similarly *Distance* performs better than *Degree* on the *Facebook* and *DBLP* graphs, but it is outperformed by *Degree* on the *Internet* graph. All our heuristics perform better that the *Random* method, with the exception of *Degree* on the *Facebook* graph.

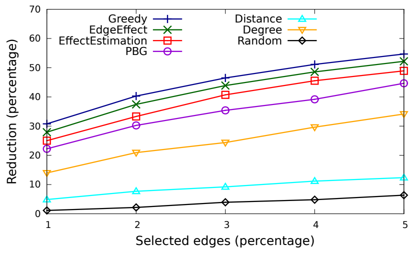
In Figure 5, we show the effectiveness of the methods *EdgeEffect*, *EffectEstimation*, *PBG* and *Distance* as a function of the size of the graph, for selected set size fixed to 5% of the candidate edges. We observe that the performance of our algorithms, both individually and comparatively, remains relatively stable as the size of the dataset changes.

Finally, it is interesting to evaluate the reduction percentage in absolute terms. We observe that with the insertion of a small fraction of the candidate edges it is possible to achieve a significant fraction (around 50%) of the total reduction achieved by the full candidate set C. This observation is consistent for all datasets.

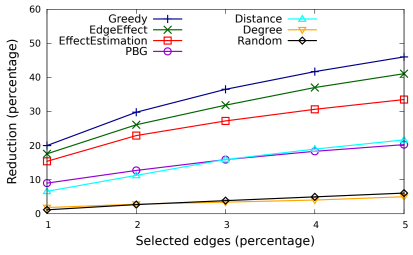
6.3 Efficiency. We implemented all our algorithms in `C++`, using `g++v.4.6.4` to compile the code. We report the running times on a GNU/Linux machine, with Ubuntu (SMP): a Intel(R) Xeon(R) CPU server 64-bit NUMA machine with four processors and 16GB of RAM memory. Each processor has 4 cores sharing a 8MB L3 cache, and each core has a 256KB private L2 cache and 2.40GHz speed. All times are reported in milliseconds.

First, we consider the performance of the *EdgeEffect* algorithm from Section 4, and we compare it with computing the all-pairs shortest paths (APSP) from scratch. Table 2 shows the average (over all candidate edges) running time for the *Internet*, *Facebook* and *DBLP* instances. Although both algorithms have worst-case complexity $O(nm)$, *EdgeEffect* is many orders of magnitude faster in practice. This observation is consistent for different network sizes.

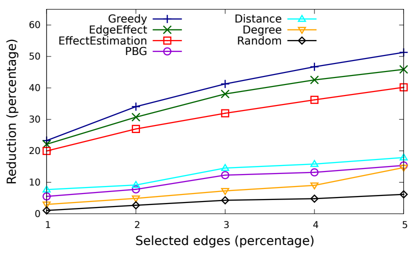
In Figure 6, we present the running times of our



(a) *Internet*

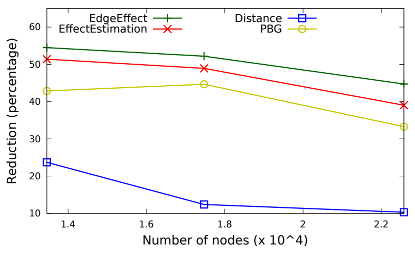


(b) *Facebook*

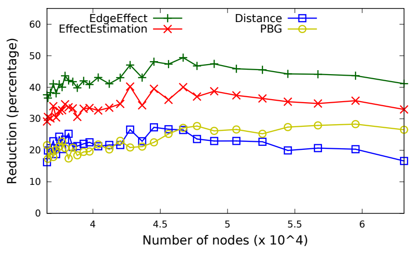


(c) *DBLP*

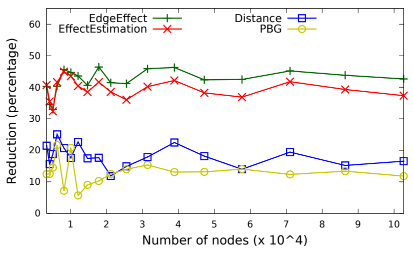
Figure 4: Effectiveness of the algorithms in Section 5 as a function of selection size.



(a) *Internet*



(b) *Facebook*



(c) *DBLP*

Figure 5: Effectiveness of *EdgeEffect*, *EffectEstimation*, *PBG*, and *Distance* as a function of network size.

algorithms for the three datasets as a function of the network size. To make our observations independent of the candidate set C, we normalize the running times with the size of C, and report the average running time per edge. We omit the *Greedy* method, since it is evidently slower than the rest (essentially k times the running time of *EdgeEffect*), and *Degree* since it has constant running time. The *Distance* algorithm defines a baseline for all methods, since it performs a single BFS traversal. The *EdgeEffect* method is the most inefficient method overall. The *EffectEstimation* is an order of magnitude faster than *EdgeEffect* on the *Internet* dataset, and two orders of magnitude faster on *Facebook* and *DBLP*. It is also faster than *PBG* on most instances. Its running time grows at the same rate as *Distance*. The behavior of *PBG* is erratic. This is due to the fact that it has complexity $O(mn + n|C|)$), so the running time per edge is sensitive to the size of C. *PBG* is more efficient for large candidate sets, and less for candidate sets that are small relative to the graph size. For example, this is the case in the last snapshot of the *Facebook* graph (Figure 6(b)).

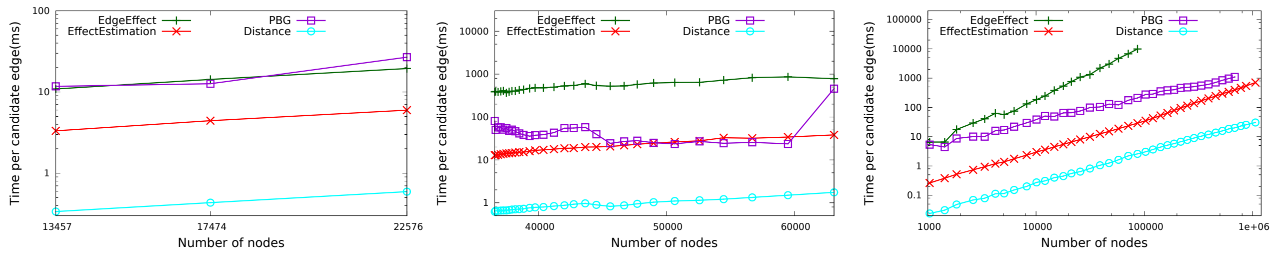
6.4 Link prediction application. Finally, we consider the link recommendation application scenario we described in Section 1. We use a link prediction algorithm to obtain a set C with the top recommended edges. We then apply our algorithms to select a set $S \subseteq C$ of k edges that reduce $R_G(S)$ the most. We use the *Facebook* graph for this experiment, since link prediction has a natural application on social networks.

Method	*RPR*	*AA*
Top	9.07 %	17.85 %
Greedy	35.03 %	64.63 %
EdgeEffect	33.19 %	64.51 %
EffectEstimation	29.68 %	59.05 %
PBG	18.11 %	61.40 %

Table 3: Effectiveness of shortcut selection for the link recommendation application.

In our experiment we used the link prediction algorithms *RootedPageRank (RPR)* [8], and *AdamicAdar (AA)* [1]. We use the top-1000 links with the highest prediction score as our candidate edges, and we ask for $k = 100$ shortcuts. Table 3 reports the effectiveness of our algorithms (we omit the baselines *Distance* and *Degree*) in terms of the reduction percentage. We also report the results for the set of the top-100 edges with the highest link-prediction score (denoted as *Top*).

The relative performance of our algorithms is consistent with our previous experiments. An interesting observation is that the *PBG* algorithm seems to be affected by the number of shortcuts at distance 2: in the case of *RPR* only 25% of the candidate edges are at distance 2, while in case of the *AA* all candidate edges are



(a) *Internet*(log-linear scale) (b) *Facebook*(log-linear scale) (c) *DBLP*(log-log scale)

Figure 6: Algorithm Efficiency: Running time per candidate edge as a function of the network size.

at distance 2. We also note that our algorithms achieve reduction ratio more than three times that of the top-100 links. It is clear that the existing link prediction algorithms do not take the benefit of the network into account in their recommendations. It is an interesting problem for further study to understand the trade-off between prediction accuracy and network benefit.

7 Conclusion

In this paper we defined the SHORTCUTSELECTION problem, and studied it theoretically and experimentally. We described the *EdgeEffect* algorithm for computing efficiently the effect of a single edge insertion on the average shortest path length. The algorithm provides a characterization of the affected nodes, which we utilize to construct a novel heuristic algorithm, *EffectEstimation*. Our experiments demonstrate our algorithms to be both efficient and effective. In the future, we are interested in better understanding the approximability of our problem, and explore the accuracy-benefit tradeoff for the link prediction application.

Acknowledgments This work has been supported by the Marie Curie Reintegration Grant project titled JMUGCS which has received research funding from the European Union.

References

[1] Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[2] Giorgio Ausiello, Giuseppe F Italiano, Alberto Marchetti Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *Journal of Algorithms*, 12(4):615–638, 1991.

[3] Vineet Chaoji, Sayan Ranu, Rajeev Rastogi, and Rushi Bhatt. Recommendations to boost content spread in social networks. In *WWW*, 2012.

[4] Vatche Ishakian, Dóra Erdös, Evimaria Terzi, and Azer Bestavros. A framework for the evaluation and management of network centrality. In *SDM*, 2012.

[5] Konstantina Lazaridou, Konstantinos Semertzidis, Evaggelia Pitoura, and Panayiotis Tsaparas. Identifying converging pairs of nodes on a budget. In *EDBT*, 2015.

[6] Michael Ley. The DBLP computer science bibliography: Evolution, research issues, perspectives. In *SPIRE*, 2002.

[7] Dong Li, Zhiming Xu, Sheng Li, Xin Sun, Anika Gupta, and Katia Sycara. Link recommendation for promoting information diffusion in social networks. In *WWW (Companion Volume)*, 2013.

[8] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

[9] Adam Meyerson and Brian Tagiku. Minimizing average shortest path distances via shortcut edge addition. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 272–285. Springer, 2009.

[10] Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the flickr social network. In *ACM SIGCOMM Workshop on Social Networks (WOSN)*, 2008.

[11] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, 1978.

[12] Manos Papagelis, Francesco Bonchi, and Aristides Gionis. Suggesting ghost edges for a smaller world. In *CIKM*. ACM, 2011.

[13] Ahmet Erdem Sariyüce, Kamer Kaya, Erik Saule, and Ümit V. Çatalyürek. Incremental algorithms for network management and analysis based on closeness centrality. *CoRR*, abs/1303.0422, 2013.

[14] Robert Tarjan. Finding dominators in directed graphs. *SIAM Journal on Computing*, 3(1):62–89, 1974.

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

[16] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *ACM SIGCOMM Workshop on Social Networks (WOSN)*, 2009.