Centrality-Aware Link Recommendations

Nikos Parotsidis
Dipartimento di Ingegneria
Civile e Ingegneria Informatica
Università di Roma
"Tor Vergata", Roma, Italy
nikos.parotsidis@uniroma2.it

Evaggelia Pitoura
Department of Computer
Science & Engineering
University of Ioannina
pitoura@cs.uoi.gr

Panayiotis Tsaparas
Department of Computer
Science & Engineering
University of Ioannina
tsap@cs.uoi.gr

ABSTRACT

Link recommendations are critical for both improving the utility and expediting the growth of social networks. Most previous approaches focus on suggesting links that are highly likely to be adopted. In this paper, we add a different perspective to the problem by aiming at recommending links that also improve specific properties of the network. In particular, our goal is to recommend to users links that if adopted would improve the user centrality in the network. Specifically, we introduce the *centrality-aware link recommendation problem* as the problem of recommending to a user u, k links from a pool of recommended links so as to maximize the *expected* decrease of the sum of the shortest path distances of u to all other nodes in the network. We show that the problem is NP-hard, but our optimization function is monotone and sub-modular which guarantees a constant approximation ratio for the greedy algorithm. We present a fast algorithm for computing the expected decrease caused by a set of recommendations which we use as a building block in our algorithms. We provide experimental results that evaluate the performance of our algorithms with respect to both the accuracy of the prediction and the improvement in the centrality of the nodes, and we study the tradeoff between the two.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining; E.1 [**Data Structures**]: Graphs and networks

Keywords

Probabilistic Networks, Link Recommendations, Node Centrality, Social Networks

1. INTRODUCTION

Link recommendations constitute an important task in social networks. Recommending to network users connections of potential interest helps them grow their social circle and increases the added value of the social experience. At the same time, link recommendations are beneficial to the network as a whole, since they expedite the network growth, and they increase the overall user engagement. Therefore, substantial research effort has been devoted to producing recommendations of high accuracy [14]. Most previous research on link recommendation focuses on predicting or estimating the likelihood of the connections being adopted by the user. Various approaches have been proposed to this end, based on exploring structural properties of the network, attributes of the users, previous user interactions, or other factors [14, 2, 25].

In this paper, we add a new perspective to the problem by aiming at recommending links that if adopted by the users would lead to networks with the small world property, i.e., short paths between users. This is an important property, since it improves connectivity among the nodes of the network and facilitates and amplifies dynamic processes. For instance, users of professional networks want to be close to other nodes in the network as an indicator of their individual importance. Moreover, the small world phenomenon helps social network users spread their content faster on the network. At the same time, network owners can benefit from small average shortest paths, since this is one of the key properties that helps viral marketing and information diffusion. Therefore, it is in the interest of both the network as a whole, and of the individual users to maintain and strengthen the smaller world phenomenon.

In particular, we assume that we are given as set C of link recommendations for user u along with an estimation of the probability p_e of each link e in C being adopted by u and added to the network. We assume that these recommendations are produced by some existing link recommendation algorithm. Given a number k of links to recommend to u, we want to select k of the links in C so as to increase the "centrality" of u, that is, minimize its shortest path distance to the other nodes in the network. Since depending on p_e, the user may or may not accept a particular link, the addition of the links to the network results in a probabilistic graph, and the decrease in the shortest path lengths is a random variable. Thus, our objective becomes to maximize the *expected* reduction in the shortest path lengths.

There is a natural tradeoff between the accuracy (likelihood to be adopted) and the utility (improvement in centrality) of the link recommendations. Links that connect to "close-by" users are likely to be adopted but do not improve the centrality of the node. Links that link to "far-away" nodes are less likely to be adopted but they result in sig-

WSDM'16, February 22–25, 2016, San Francisco, CA, USA.
© 2016 ACM. ISBN 978-1-4503-3716-8/16/02…$15.00
DOI: http://dx.doi.org/10.1145/2835776.2835818

nificant decrease in the distances with the remaining graph. The expected reduction is an objective function that combines naturally the utility and the accuracy of the recommendations. We should note that our goal is not to propose yet another link recommendation algorithm. Instead, we are interested in selecting from the results of an existing link recommendation algorithm a set of links that result in a large reduction in the shortest path distances of a node in expectation. In this way, we combine the likelihood of the link to appear with its effect on the centrality of the node.

Although there has been some previous work on selecting edges to add to a graph so that specific criteria are met, such as minimizing the average all-pair shortest path distances (e.g., [18, 19]), or improving the average closeness centrality of a specific node ([4, 8]), our work is the first to consider the addition of probabilistic edges. We make the following contributions:

- We define the novel problem of Centrality-Aware Link Recommendations, where given a set of link recommendations and their likelihood, we aim to select k recommendations that maximize the reduction of the expected distances of the node to the rest of the graph.
- We show that our problem is NP-hard, but our objective function is monotone and submodular, and as a result the greedy algorithm gives a solution with constant approximation factor. We also propose an efficient algorithm for computing the expected reduction by exploiting the properties of our problem.
- We evaluate our approach experimentally using real datasets and link recommendations produced by two commonly used recommendation algorithms. We compare the efficiency and effectiveness of our algorithms under different settings, and we demonstrate that we can achieve a sizeable reduction in distances with an acceptable sacrifice in prediction accuracy. We also report experimental results that quantify the trade-off between link-prediction accuracy and centrality utility.

The remainder of this paper is structured as follows. In Section 2, we present related work, in Section 3, we provide the formal definition of our problem. Section 4 includes complexity results and Section 5 our algorithms. In Section 6, we present the results of our experimental evaluation. Section 7 concludes the paper.

2. RELATED WORK

The problem of minimizing the average all-pairs shortest path distance via edge addition was previously studied in [18, 19, 17]. Papagelis et al. [18, 17] study the problem of adding k edges, from the set of missing edges of the graph, in order to minimize the average shortest paths in the graph. In [19], we consider a variation of the problem where the set of candidate edges is given as an input, and the goal is to select a subset of k of them. These works do not focus on link recommendations, and they do not consider probabilistic edges.

The idea of promoting graph properties via link recommendations is explored by Li et al. [12], where they re-rank link recommendations in order to promote information diffusion in a social network. Also, Chaoji et al. [3] consider as input a set of recommended links and compute k edges per node that boost content spread in the network. Both these works focus on maximizing diffusion spread. Maximizing the reduction in average network distances poses a different set of challenges.

The average shortest path length is related to the average *closeness centrality* of nodes in the graph. Ishakian et al. [8] study the problem of finding the k edges whose addition to the graph maximizes the centrality of a *specific* target node. The same problem was considered by Crescenzi et al. [4], where they show that the greedy algorithm provides a $(1 - 1/e)$-approximation factor, and that this approximation bound is tight. Sariyüce et al. [21] propose an incremental algorithm to efficiently update the closeness centrality values under changes in network topology. Meyerson and Tagiku [15] study the problem of finding the k edges whose addition minimizes the shortest path distances between all pairs of nodes. They prove that several variations of the problem, including the single source shortest paths, are *NP-hard* and they provide a $O(1)$ approximation algorithm. Tong et al. [22] search for the k edges whose addition will best enable the propagation of information in the network under the SIS model [5]. Lazaridou et al. [10] identify the set of top-k pairs of nodes whose distances are reduced the most as the result of edge additions. None of these works considers the case of probabilistic edges and expected centrality, or link recommendations.

The problem of finding shortest paths in a graph with independent randomly distributed edge lengths is known to be *#P-complete* [23]. Frieze and Grimmett [7] present a formula for computing the exact expected shortest path distance between two nodes, but it requires the knowledge of all the different paths between the two nodes. The enumeration of the paths between two nodes in a network is also known to be *#P-complete* [23]. Frank [6] uses Monte Carlo results to develop a method for computing shortest paths in probabilistic graphs.

There is a significant amount of work on problems over probabilistic graphs. For example, Potamias et al. [20] consider the problem of computing the k nearest neighbors, and Kollios et al. [9] study clustering on probabilistic graphs. However, the problem we study in this paper is novel.

Finally, there is a large body of work on the problem of link recommendations [14, 2, 25]. As mentioned earlier, our goal in this paper is not to introduce a new recommendation algorithm. Instead, we propose selecting from the recommendations of an existing algorithm those links that could improve network properties.

3. PROBLEM FORMULATION

We consider as input to our problem a graph $G = (V, E)$ and a set C of link recommendations for user u along with their probability of being adopted by u. In particular, $C = \{(e, p_e) : e \notin E, p_e \in [0, 1]\}$ consists of a set of candidate edges incident to u not present in G, where each edge $e \in C$ is associated with the probability p_e that e will appear in the network. We assume that the probability p_e of link $e = (u, v)$ is provided by some link recommendation algorithm that estimates p_e based on the structural properties of the graph (e.g., the number of neighbors that u and v have in common), on attribute values of the nodes (e.g., on how similar u and v are, by exploiting homophily), or other factors. We will refer to the edges of the candidate set C as *probabilistic* edges.

Under this setting, we study the problem of identifying the set $S \subseteq C$ of k edges such that the addition of the edges in S to graph G minimizes the *expected* shortest path lengths of node u to all other nodes in G.

Given a subset $S \subseteq C$ of the candidate edges, we use $\widetilde{G}_S$ to denote the *probabilistic* graph that results by adding the set of edges S to the graph G. Let $D_{\widetilde{G}_S}(u,v)$ denote the random variable whose value is the length of the shortest path distance between u and v in the graph $\widetilde{G}_S$. To compute the expectation $E[D_{\widetilde{G}_S}(u,v)]$ we need to consider all possible $2^{|S|}$ graphs that may result by the materialization of the edges in S. Each graph has a specific probability to appear. Given a subset of edges $F \subseteq S$, the probability of exactly these edges appearing in the graph is equal to:

$$P(F) = \prod_{e \in F} p_e \prod_{e \in S \setminus F} (1 - p_e)$$

Now, let $d_G(u,v)$ be the shortest path distance between nodes u and of v in the *deterministic* graph G, and let $G \cup F$ denote the *deterministic* graph that results from the addition of the edges in F to G. Then, the expected shortest path distance between u and v, after the addition of the probabilistic set of edges S to the graph G is equal to:

$$E[D_{\widetilde{G}_S}(u,v)] = \sum_{F \in 2^S} P(F) d_{G \cup F}(u,v) \tag{1}$$

where, 2^S denotes the powerset of the set S.

Let $L_G(u) = \sum_{v \in V} d_G(u,v)$ denote the sum of shortest path lengths of node u to all the nodes in a graph G. This value is directly connected to the centrality of node u; a node with short paths to other nodes is a central node in the graph. We also define $L_{\widetilde{G}_S}(u) = \sum_{v \in V} E[D_{\widetilde{G}_S}(u,v)]$ to be the *expected* sum of shortest path lengths of node u in the probabilistic graph $\widetilde{G}_S$. The addition of any edge in a graph can only reduce the shortest path lengths of u. This allows us to define $R_u(S) = L_G(u) - L_{\widetilde{G}_S}(u)$ to be the expected reduction in the shortest path lengths of u caused by the addition of the probabilistic edge set S to the graph G. We are looking for the set S of size k that maximizes $R_u(S)$, for a given u. We formally define our problem as follows.

PROBLEM 1. [CENTRALITY-AWARE LINK RECOMMENDATIONS (CALR)]. *Given a graph $G = (V, E)$, a user u, a positive integer k, and a candidate set $C = \{(e, p_e) : e \notin E, p_e \in [0, 1]\}$ of probabilistic edges incident to u recommended by a link-recommendation algorithm, select a set $S \subseteq C$ of size k to recommend to u, such that $R_u(S)$ is maximized.*

4. COMPLEXITY ANALYSIS

In this section we study theoretically the CALR problem. First, we give a formula for the efficient computation of the expected reduction in the shortest path length between two nodes. We then show that the CALR problem is NP-hard and we consider approximate solutions.

4.1 Expected single source shortest path length computation

The computation of the expected shortest path length between two nodes in a general probabilistic graph is known to be a #P-complete problem, since it requires the consideration of all possible paths between the two nodes, which, in the general case, may be exponentially many. However, in our setting, where the probabilistic edges are all incident on a single node u, and we are interested in the single source paths from u, we can avoid this computation thanks to the following property.

PROPERTY 4.1. *Given a deterministic graph G, a target node u, and a set of probabilistic edges S all incident on u, any simple path from u to any other node in the probabilistic graph $\widetilde{G}_S$ uses at most one probabilistic edge $e \in S$.*

Given Property 4.1, for a node u, and a target node v, we only need to consider $|S|$ shortest paths between nodes u and v. However, using Equation (1) to compute the expected shortest path length between u and v is still expensive, since it considers all $2^{|S|}$ possible materializations of the probabilistic edges. We now consider a more efficient computation of the shortest path length between node u and a node v.

For a random variable X that takes natural values between 1 and N, the expected value of X can be computed as $E[X] = \sum_{i=1}^{N} P[X \geq i]$. Therefore, we can compute the expected shortest path length between source node u and a node v in the probabilistic graph $\widetilde{G}_S$ as follows.

$$E[D_{\widetilde{G}_S}(u,v)] = \sum_{i=1}^{d_{\max}} P[D_{\widetilde{G}_S}(u,v) \geq i] \tag{2}$$

where $d_{\max} = d_G(u,v)$ is the length of the shortest path between u and v in the deterministic graph G.

To compute the probability $P[D_{\widetilde{G}_S}(u,v) \geq i]$ we need to consider the effect of the addition of each edge to the graph G. We define $C_\ell = \{e \in C : d_{G \cup e}(u,v) = \ell\}$ to be the set of candidate edges such that when added individually to the initial graph G the shortest path between u and v in the *deterministic* graph $G \cup e$ has length ℓ. Note that $C_{d_{\max}}$ is the set of edges whose addition to G does not reduce the shortest path between u and v. Also, for some i, $1 \leq i \leq d_{\max}$, we define $M_i = \bigcup_{1 \leq \ell < i} C_\ell$ to be the set of edges whose addition to the graph G results in shortest path length *strictly less* than i.

Now, consider a set $S \subseteq C$ of candidate edges, and the resulting probabilistic graph $\widetilde{G}_S$. The probability that the distance between u and v in the graph $\widetilde{G}_S$ is at least i is equal to the probability that none of the edges in $S \cap M_i$ appear in the graph. Therefore, we have:

$$P[D_{\widetilde{G}_S}(u,v) \geq i] = \prod_{e \in S \cap M_i} 1 - p_e \tag{3}$$

Combining Equations (2) and (3) we can compute the expected length of the shortest path between source node u and a node v in the probabilistic graph $\widetilde{G}_S$ in polynomial time. We can then compute the expected sum of shortest path lengths $L_{\widetilde{G}_S}(u)$, and the expected reduction $R_u(S)$ in the sum of shortest paths.

We note that Property 4.1, and all the ensuing analysis holds also for directed graphs, since any simple path from the target vertex u to any other vertex uses at most one outgoing probabilistic edge from u. Thus, the algorithms described in Section 5 for undirected graphs can easily be modified to work for the directed case.

4.2 Hardness and Approximation

Our problem contains as a special case the deterministic case where all candidate edges have probability one. This problem is known to be NP-hard [15]. Given that the problem is NP-hard we look for approximation algorithms with provable approximation guarantees. To this end we will show that our optimization function $R_u(S)$ is *monotone* and *submodular*.

Let Ω be a set, and let $f : 2^\Omega \rightarrow \mathbb{R}$ denote a set function over Ω. The function f is (increasingly) monotone if for every set $S \subset \Omega$ and every $x \in \Omega \setminus S$ it holds that $f(S) \leq f(S \cup x)$. The function f 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 [16] that when maximizing a monotone and submodular set function under cardinality constraints, the simple greedy algorithm that always adds to the set the element with the largest marginal gain has approximation factor $1 - 1/e$ where e is the base of the natural logarithm.

We will now show that our function is monotone and submodular.

Lemma 4.2 (Monotonicity). *The function $R_u(S)$ is increasingly monotone.*

Proof. Given a set $S \subseteq C$ and an edge $eC \setminus S$, we will prove that $P[D_{\widetilde{G}_{S \cup e}}(u, v) \geq i] \leq P[D_{\widetilde{G}_S}(u, v) \geq i]$, that is, the probability of having a long path between u and v decreases as we add edges to the set S. Then, from Equation (2) it follows that for every node v, $E[D_{\widetilde{G}_{S \cup e}}(u, v)] \leq E[D_{\widetilde{G}_S}(u, v)]$, and thus $L_{\widetilde{G}_{S \cup e}}(u) \leq L_{\widetilde{G}_S}(u)$. Since $L_G(u)$ is constant, it follows that $R_u(S \cup e) \geq R_u(S)$, that is, the function is increasingly monotone.

Without loss of generality assume that $e \in C_\ell$ for some ℓ, $1 \leq \ell \leq d_{\max}$. It follows that $e \in M_i$ for all $i > \ell$. From Equation (3) it follows that $P[D_{\widetilde{G}_{S \cup e}}(u, v) \geq i] = P[D_{\widetilde{G}_S}(u, v) \geq i]$ for $i \leq \ell$, and $P[D_{\widetilde{G}_{S \cup e}}(u, v) \geq i] = (1 - p_e)P[D_{\widetilde{G}_S}(u, v) \geq i]$ for $i > \ell$, which proves our claim. □

Lemma 4.3 (Submodularity). *The function $R_u(S)$ is submodular.*

Proof. Similar to before, in order to show that the function $R_u(S)$ is submodular, we will work with the probabilities $P[D_{\widetilde{G}_S}(u, v) \geq i]$. In the following, for a fixed pair (u, v), we will use $P_S(i)$ to denote the probability $P[D_{\widetilde{G}_S}(u, v) \geq i]$.

Let S and S' be two sets of probabilistic edges such that $S \subseteq S'$, and let e be a candidate edge not in S'. It is not hard to see that in order to prove that $R_u(S \cup e) - R_u(S) \geq R_u(S' \cup e) - R_u(S')$, it suffices to show that for every (u, v), $P_S(i) - P_{S \cup e}(i) \geq P_{S'}(i) - P_{S' \cup e}(i)$, for $1 \leq i \leq d$.

Without loss of generality assume that $e \in C_\ell$ for some ℓ, $1 \leq \ell \leq d_{\max}$. Similar to the previous proof, for $i \leq \ell$, we have that $P_{S \cup e}(i) = P_S(i)$, and $P_{S' \cup e}(i) = P_{S'}(i)$ and thus our claim holds. For $i > \ell$ we have that $P_{S \cup e}(i) = (1 - p_e)P_S(i)$, and therefore $P_S(i) - P_{S \cup e}(i) = p_e P_S(i)$. Similarly, $P_{S'}(i) - P_{S' \cup e}(i) = p_e P_{S'}(i)$. From the monotonicity property we have that $P_S(i) \geq P_{S'}(i)$, and thus our claim holds. □

Corollary 4.4. *The greedy algorithm for the CALR problem achieves $(1 - 1/e)$ approximation ratio.*

5. ALGORITHMS

In this section we present algorithms for the CALR problem. We consider the Greedy algorithm and other more efficient heuristics.

Algorithm 1 The *Greedy* algorithm

Input: Graph G, node u, set of probabilistic edges C
Output: Set S of k recommendations to u

```
 1: $S = \emptyset$
 2: for $1 \leq i \leq k$ do
 3:    for $e \in C$ do
 4:       Compute $R_u(S \cup e)$
 5:    end for
 6:    $e^* = \arg\max_{e \in C}\{R_u(S \cup e)\}$
 7:    $S = S \cup e^*$
 8:    $C = C \setminus e^*$
 9: end for
10: return $S$
```

Algorithm 2 The ESSSP algorithm

Input: Graph G, node u, set of probabilistic edges S
Output: $L_{\widetilde{G}_S}(u)$

```
Compute $d_G(u, v), \forall v \in V$
Initialize $b_v(\ell) = 1$, $\forall v \in V$, $1 \leq \ell \leq d_G(u, v)$
for $e = (u, x) \in S$ do
   Compute $d_G(x, v)$, $\forall v \in V$
   for $v \in V \setminus u$ do
      if $d_G(x, v) + 1 < d_G(u, v)$ then
         $\ell = d_G(x, v) + 1$
         $b_v(\ell) = b_v(\ell) * (1 - p_e)$
      end if
   end for
end for
$SUM = 0$
for $v \in V \setminus u$ do
   $prob = 1$
   for $\ell = 1 ... d_G(u, v) - 1$ do
      $prob = prob * b_v(\ell)$
      $SUM = SUM + prob$
   end for
end for
return $L_{\widetilde{G}_S}(u) = SUM$
```

5.1 The Greedy algorithm

The Greedy algorithm builds the solution set S incrementally, starting with the empty set. Given as input the node u and the set of candidate edges C, in each iteration it selects the edge $e^* \in C \setminus S$ whose addition causes the largest reduction in the expected distances of node u. The outline of the algorithm is shown in Algorithm 1.

An important step in the execution of the Greedy algorithm is step 4, where we compute the expected reduction caused by the addition of edge e. This entails the computation of the expected lengths of the shortest paths between u and all nodes in the graph. In the general case this is hard, since it would require considering an exponential number of paths. However, as we have discussed in Section 4 we can compute this quantity efficiently by exploiting the properties of our problem.

We now describe the computation of $L_{\widetilde{G}_S}(u)$, the expected length of single-source shortest paths for a node u for a given set S, in detail. For convenience in this computation we use a slight variation of the Equation (3). Namely, given the sets C_ℓ, $1 \leq \ell \leq d_{\max}$, and a set of edges S, let $S_\ell = C_\ell \cap S$.

For some value $1 \leq i \leq d_{\max}$, we have:

$$P[D_{\widetilde{G}_S}(u,v) \geq i] = \prod_{\ell=1}^{i-1} 1 - P(S_\ell) \tag{4}$$

where $P(S_\ell) = 1 - \prod_{e \in S_\ell}(1 - p_e)$ is the probability that at least one edge in S_ℓ appears in the graph, and $1 - P(S_\ell)$ is the probability that no edge in S_ℓ appears in the graph.

The Expected Single Source Shortest Path (ESSP) Algorithm (shown in Algorithm 2) proceeds as follows. We first compute all single-source shortest paths from u. We refer to the length of the shortest path between u and another node v, in the initial deterministic graph G, as $d_G(u,v)$. For each node v, we maintain a vector b_v, of size $d_G(u,v)$, where $b_v(\ell)$ contains the probability that there is no edge $e \in S$ whose addition to G results in a shortest path between u and v of length ℓ in the deterministic graph $G \cup e$. For a given v, this value corresponds to the probability $1 - P(S_\ell)$. Given the $b_v(\ell)$ values, for all $v \in V$, $1 \leq \ell \leq d_G(u,v)$, we can apply in turn Equations (4) and (2) to compute the expected shortest path length between u and v as $E[D_{\widetilde{G}_S}(u,v)] = \sum_{i=1}^{d_G(u,v)} \prod_{\ell=1}^{i-1} b_v(\ell)$. Summing over all nodes $v \in V$ gives us $L_{\widetilde{G}_S}$.

We compute the $b_v(\ell)$ values incrementally. Initially, we set $b_v(\ell) = 1$, for all $v \in V$, $1 \leq i \leq d_G(u,v)$. We then process the edges in S one by one. For each edge $e = (u,x) \in S$, with probability p_e, we compute all single-source shortest paths from x in the deterministic graph G, and we identify the nodes v for which $d_G(x,v) + 1 < d_G(u,v)$, that is, the addition of e created a new shortest path between u and v through node x. In this case, the edge e creates a shorter path between u and v of length $\ell = d_G(x,v) + 1$ with probability p_e. We update the value $b_v(\ell)$ to represent the new correct probability of non existence of an edge whose addition in G creates a shortest path of length ℓ, i.e., $b_v(\ell) = (1 - p_e) \cdot b_v(\ell)$. After processing all edges in S, the vector b_v for node v has values $b_v(\ell) = 1 - P(S_\ell)$.

The ESSSP algorithm has complexity $O(|S|(m+n) + n\delta)$, where m is the number of edges and δ is the diameter of the deterministic graph G, and uses space $O(n\delta)$. The m factor comes from the BFS traversal in step 4. The Greedy algorithm performs $k|C|$ calls of the ESSSP algorithm, where $|S|$ takes values $1, ..., k$, resulting in complexity $O(k^2|C|(m+n) + k|C|n\delta)$. The space complexity of the Greedy algorithm is also $O(n\delta)$.

5.2 Efficient Heuristics

The Greedy algorithm has known approximation guarantees: it computes an $(1 - 1/e)$ approximate solution for the CALR problem. However, since it needs to recompute the expected reduction caused by an edge in each iteration, it is slow, and does not scale well for large networks. We now consider some efficient heuristics, with no provable guarantees, in order to compare the trade-off between efficiency and effectiveness of our methods.

Top Edges: This method assigns to each edge $e \in C$, a score equal to the expected reduction in the shortest paths of u achieved by the addition of edge e to G, i.e., $R_u(e)$. Since we consider every edge independently, we do not need to run Algorithm 2 for computing the expected reduction $R_u(e)$. We can instead compute for each node v where $d_G(u,v) < d_{G \cup e}(u,v)$ the difference $d_G(u,v) - d_{G \cup e}(u,v)$, sum the differences over all such nodes, and multiply by p_e. Computing the values $d_G(u,v)$ and $d_{G \cup e}(u,v)$ can be done easily by performing two BFS traversals, one on the graph G and one on the graph $G \cup e$. Then, the edges are sorted according to their score and the k edges with the greater score are returned. The time complexity of this method is $O(|C|(m+n))$.

Deterministic Greedy Algorithm: This is a variation of the greedy algorithm that ignores the probabilities of the edges. This method also builds the solution set S incrementally, starting with the empty set. In each iteration it selects the edge $(u,v) \in C \setminus S$ that maximizes the reduction $L_{G \cup S}(u) - L_{G \cup S \cup e}(u)$ on the shortest paths of u between the deterministic graphs $G \cup S$ and $G \cup S \cup e$. The edge e with the maximum reduction is added to the solution set S. Computing the reduction caused by an edge can be done with a BFS traversal. Each iteration takes time $O(|C|(m+n))$, which implies a total running time $O(k|C|(m+n))$; the space complexity is linear.

6. EXPERIMENTS

The goals of our experiments are three-fold. First, we address the practical problem of estimating edge probabilities from the scores of a link recommendation algorithm. Then, we focus on the effectiveness of the algorithms and of the CALR approach in general. In particular, we highlight how the CALR approach achieves a balance between the accuracy of the recommendations, and the utility of the network resulting from the successful recommendations. We study and quantify this tradeoff. Finally, we evaluate the efficiency of our algorithms for the CALR problem.

6.1 Baselines

In our experiments we consider all the algorithms that we defined in Section 5. We use *Greedy* to denote the Greedy algorithm, *TopEdges* to denote the algorithm that selects the top-k edges according to their $R_u(e)$ score, and *D-Greedy* to denote the deterministic Greedy algorithm. We also consider the following baselines that do not take into account the effect of the edge addition on the node centrality.

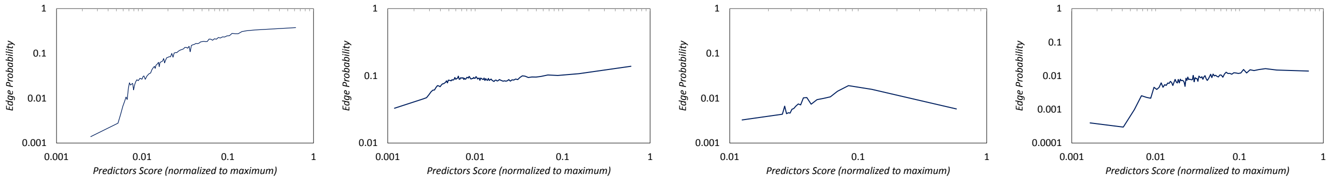
Most probable: In this method we select the k edges with the highest probability. The algorithm simply sorts the edges in the candidate set C according to their probability; ties are resolved arbitrarily. This method corresponds to a traditional link recommendation algorithm. It serves the purpose of understanding the trade-off between prediction accuracy and the decrease in the shortest path distances. We will refer to this method as *MostProb*.

Random: In this method we randomly select k edges from the candidate set C. Note that this is not a completely random selection (as it is used in [13], where the random predictor selects randomly from all non-existent edges), since we select randomly from the set C which already contains links that are relevant to the user. We will refer to this method as *Random*.

6.2 Datasets and Recommendation Algorithms

We use two different datasets in our experiments, each corresponding to a different type of network. Both datasets contain information about the time when each edge was created.

- The *Facebook* dataset [24] consists of a social graph from the Facebook New Orleans network. Nodes rep-



(a) *AA* on *Facebook* (b) *RPR* on *Facebook* (c) *AA* on *DBLP* (d) *RPR* on *DBLP*

Figure 1: Empirical probability function for the normalized scores (log-log scale).

Table 1: Dataset characteristics. $r(AA)$ and $r(RPR)$: number of nodes for which *AdamicAdar* and *RootedPageRank* produce sufficient number of recommendations; F: the number of edges that appeared after the initial instance.

dataset	$\|V\|$	$\|E\|$	$r(AA)$	$r(RPR)$	F
Facebook	53,306	555,763	46,687	53,306	242,929
DBLP	96,516	364,907	47,121	96,515	175,238

resent users, and edges user friendships. The dataset contains the activity of the network from September 2006 until February 2009.

- The *DBLP* dataset[1] [11] consists of a collaboration graph of authors of computer science papers, where an edge between two authors represents a common publication. We consider the *DBLP* graph containing authors and co-authorship relationships between 1990 and 2013.

Our algorithms take as input an initial set of link recommendations where each link is associated with a score that determines the confidence of the algorithm in the prediction. We use two different link recommendation algorithms in our experiments: The *AdamicAdar* (AA) [1] algorithm, and the *RootedPageRank* (RPR) [13] algorithm[2]. For each node u the link recommendation algorithms generate a candidate set C_u, that contains the links with the highest scores. We set $|C_u| = 20$, since in practice the number of recommended links is small and therefore this way we can select diverse, yet relevant, links.

For each dataset we create two graph instances: One consisting of edges up to a specific time, and one consisting of all edges. The first graph instance is given as input to the link recommendation algorithm to produce the candidate link recommendations. We choose the initial instances so as to meet three conditions: (i) the number of nodes is sufficiently large; (ii) there is a significant number of links that appear after the initial instance, and (iii) the link recommendation algorithms can produce a sufficient number of recommendations for many of the nodes in the graph.

For the *Facebook* dataset, the initial instance is the largest connected component of the graph until November 2007. The characteristics of the graph are shown in Table 1. For this instance the *AdamicAdar* method produces at least 20 recommendations for 87.5% of the nodes, while the *RootedPageRank* algorithm provides a sufficient number of recommendations for all nodes. For the *DBLP* dataset, the initial instance is constructed by considering the edges that appeared until the year 2000. To deal with the sparsity of the data, which inhibits the link recommendation algorithms, we iteratively delete the nodes with degree less that 3. The characteristics of the resulting graph are shown in Table 1. For this dataset, the *AdamicAdar* method produces a sufficient number of link recommendation for about half of the nodes, while the *RootedPageRank* produces at least 20 recommendations for all nodes.

We evaluate the prediction accuracy of our methods by considering the new edges that appear in the final instance of the dataset among the nodes that are included in the first instance. For both datasets, the number of edges that appear in the final instance (the last column of the table) is sufficiently large.

6.3 The edge probability function

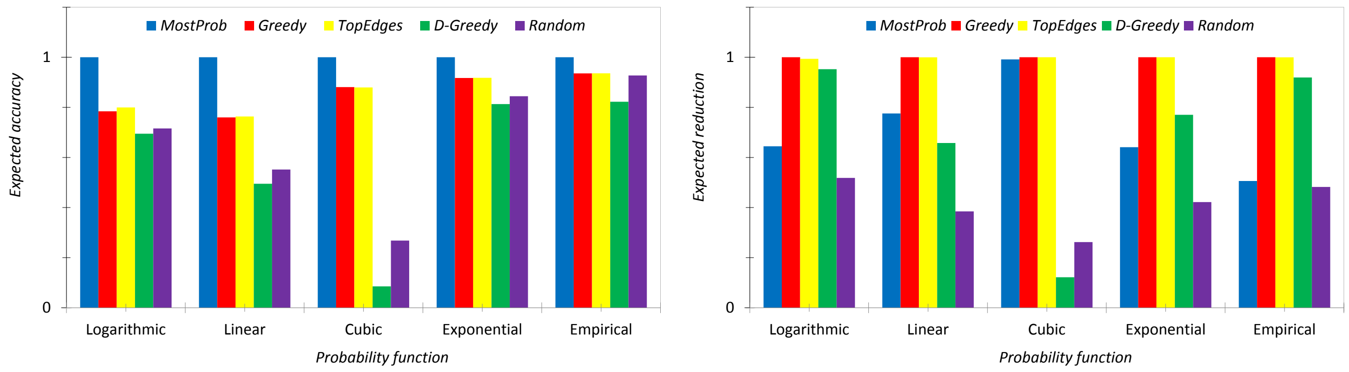
The link recommendation algorithms provide scores that correspond to the confidence of the algorithms in their recommendation. Intuitively, the higher the score, the more likely the recommendation to be accepted. However, the score itself does not give the probability that the recommendation is successful. We thus need a way to transform these scores into probabilities, as needed by the CALR problem. To this end, we follow the following process. We first normalize the scores by dividing with the maximum score over all predictions, so as to obtain values between 0 and 1. Given an edge with normalized score x, we then apply a function $f(x)$ to shape the edge probability function. We consider four different functions in our experiments: (i) The logarithmic function $f_1(x) = 0.4\log_{10}(x) + 1$; (ii) The linear function $f_2(x) = x$; (iii) The power-law function $f_3(x) = x^3$; (iv) The exponential function $f_4(x) = 2^{10(x-1)}$. Each function yields probabilities with different properties.

We also compute the *empirical probability function* of an edge to appear in the graph given its score. Given all the (normalized) scores for all predictions of the recommendation algorithm, we create an equal-frequency histogram, where each bucket contains $10,000$ scores. We then count the fraction of edges in each bucket that appear in the final instance of the network. This defines the probability of an edge with a score that falls within the bucket range to appear in the graph, yielding a probability function.

We computed the empirical probability function for all network and link recommendation algorithm pairs that we consider. The resulting probability functions (in log-log scale) are shown in Figure 1. No clear and consistent pattern emerges from these plots. The functions in Figure 1.a

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

[2] In our experiments, we used the implementations in the LPmade software package, available at http://mloss.org/software/view/307/ .



(a) *Expected Accuracy*

(b) *Expected Prediction*

Figure 2: Relative performance of the algorithms for different probability functions on the *Facebook* dataset, with *RootedPageRank* recommendations, for $k = 5$.

Table 2: Effectiveness of the methods in terms of the expected reduction $\bar{\mathcal{R}}$ and the expected accuracy $\bar{\mathcal{A}}$.

	Facebook								*DBLP*							
	AdamicAdar				*RootedPageRank*				*AdamicAdar*				*RootedPageRank*			
	$k = 2$		$k = 5$		$k = 2$		$k = 5$		$k = 2$		$k = 5$		$k = 2$		$k = 5$	
methods	$\bar{\mathcal{A}}$	$\bar{\mathcal{R}}$	$\bar{\mathcal{A}}$	$\bar{\mathcal{R}}$	$\bar{\mathcal{A}}$	$\bar{\mathcal{R}}$	$\bar{\mathcal{A}}$	$\bar{\mathcal{R}}$	$\bar{\mathcal{A}}$	$\bar{\mathcal{R}}$	$\bar{\mathcal{A}}$	$\bar{\mathcal{R}}$	$\bar{\mathcal{A}}$	$\bar{\mathcal{R}}$	$\bar{\mathcal{A}}$	$\bar{\mathcal{R}}$
MostProb	5.485	0.050	4.813	0.107	3.821	0.369	2.904	0.604	11.888	0.102	10.044	0.223	16.265	0.761	13.134	1.223
Greedy	3.986	0.130	3.862	0.213	2.813	0.528	2.207	0.779	7.290	0.383	7.039	0.580	10.476	1.361	8.290	1.925
TopEdges	3.987	0.130	3.861	0.213	2.820	0.528	2.218	0.779	7.293	0.383	7.044	0.580	10.532	1.360	8.344	1.923
D-Greedy	3.639	0.124	3.644	0.209	1.407	0.256	1.437	0.512	6.492	0.369	6.671	0.573	6.160	0.856	6.415	1.562
Random	3.636	0.033	3.624	0.081	1.597	0.122	1.602	0.300	6.871	0.072	6.868	0.178	7.145	0.273	7.154	0.663

and 1.d seem logarithmic, while the function in Figure 1.b looks like a power-law with a small exponent. Surprisingly, in some cases (Figures 1.b and 1.c) the function is not increasingly monotone, meaning that edges with higher scores are assigned lower probabilities.

6.4 Expected Reduction vs Expected Accuracy

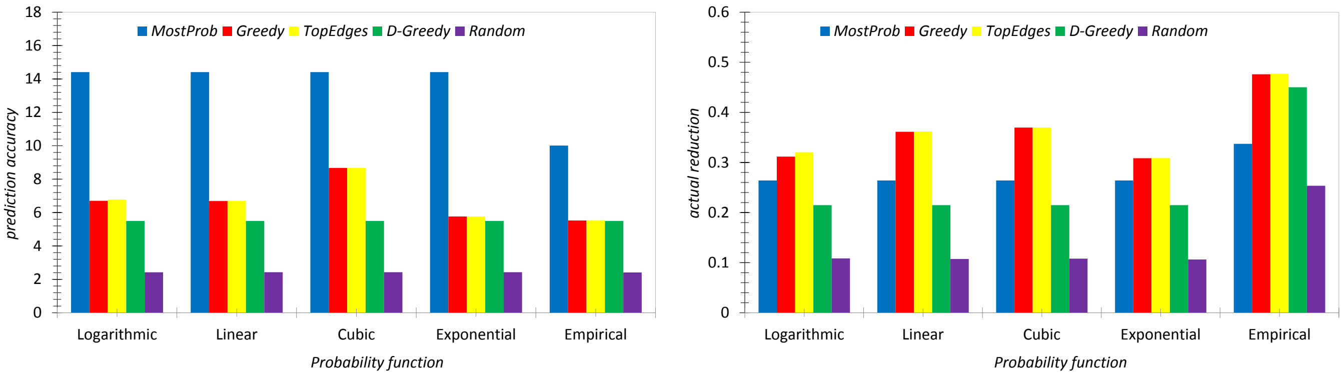
In this first set of experiments, we consider an idealized scenario where we assume that the probability functions that map scores to probabilities are perfectly accurate, and we have exact estimates for the probabilities of the links. Our goal is to understand, under this scenario, the relative performance of the different algorithms, and how the different probability functions affect their performance. Note that for this experiment, we do not make use of the second graph instance in order to assess the success of the recommendations. Instead, we assume that edges will appear in the graph with the probability they are assigned.

For a link recommendation algorithm L let $r(L)$ be the set of nodes for which L produces at least 20 recommendations. For a node u, let S_u be the set of links selected by our algorithm, where $|S_u| = k$. We consider $k = 2$ and $k = 5$ link recommendations per user. Given the set S_u, we compute the expected reduction $\mathcal{R}_u(S_u)$ for node u as described in Sections 3 and 4. To make numbers comparable across different nodes and datasets, we normalize the reduction by $L_G(u)$, the sum of shortest paths of node u before the addition of the edges in S_u. Finally, we compute the average reduction $\bar{\mathcal{R}} = \frac{1}{|r(L)|} \sum_{u \in r(L)} \mathcal{R}_u(S_u)/L_G(u)$, over all nodes in consideration.

Furthermore, for each selected set S_u we compute the *expected accuracy* of the selection as $\mathcal{A}_u(S_u) = \frac{1}{k} \sum_{e \in S_u} p_e$, that is, the fraction of the predictions that will appear in expectation. We then compute the average expected accuracy $\bar{\mathcal{A}} = \frac{1}{|r(L)|} \sum_{u \in r(L)} \mathcal{A}_u(S_u)$ by taking the average over all nodes in consideration.

In Figure 2 we see the performance of the different algorithms for the different probability functions for the *Facebook* dataset, when using the *RootedPageRank* recommendation algorithm, for $k = 5$. In this figure, we normalize the accuracy and reduction values by the maximum, so as to study the relative performance of the algorithms. As expected the *Greedy* algorithm achieves the best performance in terms of expected reduction, while the *MostProb* algorithm achieves the top performance in terms of expected accuracy.

The first observation is that *TopEdges*, which is a single iteration of the *Greedy* algorithm, performs essentially the same as *Greedy*. Therefore, we can obtain a significant speedup in the running time of the selection (see Section 6.7), without sacrificing performance. The only case where there is a noticeable difference is in the case of the logarithmic probability function. The logarithmic function is the flattest of the functions we consider, which means that it gives non-negligible probability even to edges with low scores. The *Greedy* algorithm is able to find such edges and improve the expected reduction. However, this results in a small loss in accuracy. Note that, in general, the *TopEdges* method tends to select slightly more probable edges. This is due to the fact that *Greedy* takes into account the interac-



(a) *Prediction Accuracy*

(b) *Actual reduction*

Figure 3: Performance of the algorithms for different probability functions on the *Facebook* dataset, with *RootedPageRank* recommendations, for $k = 5$.

Table 3: Prediction accuracy $\widehat{\mathcal{A}}$ and actual reduction $\widehat{\mathcal{R}}$ achieved by all methods.

	Facebook								*DBLP*							
	AdamicAdar				*RootedPageRank*				*AdamicAdar*				*RootedPageRank*			
	$k = 2$		$k = 5$		$k = 2$		$k = 5$		$k = 2$		$k = 5$		$k = 2$		$k = 5$	
methods	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$
MostProb	21.338	0.172	18.142	0.323	16.490	0.264	14.408	0.469	2.309	0.021	1.892	0.044	2.887	0.078	2.191	0.132
Greedy	11.843	0.274	11.095	0.446	8.957	0.370	8.671	0.594	1.777	0.061	1.547	0.099	2.286	0.152	1.786	0.220
TopEdges	11.844	0.274	11.096	0.446	8.957	0.370	8.671	0.594	1.776	0.061	1.543	0.099	2.286	0.152	1.785	0.220
D-Greedy	6.332	0.202	7.418	0.388	4.344	0.215	5.497	0.450	1.325	0.064	1.331	0.100	1.404	0.121	1.381	0.200
Random	1.167	0.076	2.906	0.188	0.972	0.108	2.431	0.263	0.123	0.012	0.314	0.032	0.130	0.029	0.321	0.078

tions between the edges in the selected set, while *TopEdges* considers only the individual effect of each edge.

The second observation is that the *Greedy* and *TopEdges* algorithms achieve performance competitive to the *MostProb* algorithm in terms of accuracy. At worse, they achieve around 80% of the accuracy of the *MostProb* algorithm, while at the same time they achieve 40% more in reduction. Therefore, they can achieve considerable reduction, with a manageable loss in accuracy.

The third observation is that the probability function has a significant effect in the performance of the algorithms. As the function becomes more skewed, the algorithms *Greedy*, *TopEdges*, and *MostProb* become more similar in both metrics. This is due to the fact that the more “steep” functions penalize heavily the small scores. Given that the score values tend to follow a power-law distribution, this results in many edges with very low probabilities. Such edges must cause a very large reduction in order to be selected. As a result all algorithms tend to select edges with high probability, which explains the converging behavior. This is especially pronounced in the cubic probability function, which is the most skewed in our collection.

The above discussion explains also the poor performance of *D-Greedy*. The *D-Greedy* algorithm does not take probabilities into account, and decides based solely on the shortest path reduction. As a result it performs the worst in terms of accuracy (worse than random), indicating that reduction is inversely correlated with prediction probability. Interestingly, *D-Greedy* performs poorly even in terms of the expected reduction in the case of skewed probability functions. This is due to the fact that high reduction edges have very low probabilities, since, in general, the link recommendation algorithms give high scores to links to near-by nodes.

Finally, for the empirical probability function the relative behavior of the algorithms in terms of the expected reduction is similar to that of the logarithmic function. This agrees with our observation in Figure 1.b that the probability function is a power-law with a very small exponent. With respect to the expected accuracy, the *MostProb* algorithm is clearly the best, but all algorithms follow closely. This may be a result of the fact that even for high scores, the probability function never assigns large probabilities. It is also possible that the irregularities of the empirical function affect the results.

Our observations carry over for both datasets, for the different recommendation algorithms, and for different values of k. Table 2 shows detailed results for the linear probability function. The *Greedy* and *TopEdges* achieve considerable increase in the expected reduction compared to the *MostProb*, while sacrificing little in terms of accuracy.

6.5 Prediction accuracy vs actual reduction

In this set of experiments, we assess the recommendations produced by our algorithms in terms of their accuracy in predicting links that will appear in the graph, and the actual reduction they achieve when added to the graph.

Given the initial graph instance G, let F denote the set of edges added to the graph in the later instance. For a set of recommendations S_u selected by our algorithm for node u, the set $F_u = S_u \cap F$ is the subset of the recommendations that u actually accepted. We define the accuracy of

Table 4: Prediction accuracy $\widehat{\mathcal{A}}$ and actual reduction $\widehat{\mathcal{R}}$ achieved by the hybrid k-link recommendation.

		Facebook				*DBLP*					
		AdamicAdar		*RootedPageRank*		*AdamicAdar*		*RootedPageRank*			
MostProb(l)	*TopEdges*(k-l)	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{A}}$	$\widehat{\mathcal{R}}$	$\widehat{\mathcal{RA}}$	$\widehat{\mathcal{RR}}$
0	5	8.634	0.427	6.695	0.592	1.420	0.102	1.595	0.228	0.000	1.000
1	4	10.637	0.424	8.141	0.580	1.580	0.098	1.810	0.222	0.275	0.938
2	3	12.467	0.416	9.701	0.566	1.697	0.094	1.975	0.214	0.504	0.853
3	2	14.286	0.406	11.201	0.545	1.790	0.087	2.078	0.199	0.693	0.716
4	1	16.110	0.389	12.730	0.513	1.862	0.077	2.142	0.174	0.856	0.502
5	0	18.142	0.323	14.408	0.469	1.892	0.044	2.191	0.132	1.000	0.000

the recommendation set S_u as $\mathcal{A}_u = |F_u|/|S_u|$, that is, the fraction of the recommendations that were accepted by u. We define the *actual* prediction accuracy of our algorithm as $\widehat{\mathcal{A}} = \frac{1}{|r(L)|}\sum_{u \in r(L)} \mathcal{A}(S_u)$, that is, the average prediction accuracy over all the set nodes $r(L)$, for which the original link prediction algorithm L generated sufficient number of recommendations.

Given the set F_u, we define the normalized reduction for node u, as $\mathcal{R}_u = (L_G(u) - L_{G\cup F_u}(u))/L_G(u)$, where $L_{G\cup F_u}(u)$ is the sum of shortest paths from u in the *deterministic* graph $G \cup F_u$. Note that this definition measures the effect of the set F_u in isolation from other edges in F. We define the *actual* reduction of our algorithm as $\widehat{\mathcal{R}} = \frac{1}{|r(L)|}\sum_{u \in r(L)} \mathcal{R}_u$, that is, the average normalized reduction over all nodes in cosideration.

To evaluate our algorithms, we repeat the experiments in Section 6.4, and we measure the actual prediction accuracy $\widehat{\mathcal{A}}$, and the actual reduction $\widehat{\mathcal{R}}$. Figure 3 shows the performance of the different algorithms for different probability functions on the *Facebook* dataset, with link recommendations provided by the *RootedPageRank* algorithm, for $k = 5$. The relative behavior of the algorithms remains consistent with what we observed in Section 6.4. The *MostProb* method achieves the highest accuracy rate (14.4%) and performs better than all the other algorithms by a factor of at least 2. The *Greedy* and *TopEdges* methods are again essentially identical. They outperform the rest of the algorithms in terms of actual reduction, while achieving around 50% of the accuracy of *MostProb*. *D-Greedy* performs the worst in terms of both metrics, but it is now consistently better than random. This indicates that, in real data, edges that cause high reduction have non-negligible probability to appear.

Among the different probability functions, all algorithms achieve the highest reduction for the empirical probability function. This is reasonable, but we should take into account that the empirical function is computed using the data from F. It is surprising that the prediction accuracy of *MostProb* drops for the empirical function. This is due to the non-monotonic behavior of the probability function for this dataset, and to the fact that binning creates ties between the probability values of scores that fall in the same bucket.

Among the remaining functions, the best performance in terms of both prediction accuracy and actual reduction is achieved for the cubic function. Note that the probability function can be used as a “knob” to regulate how much emphasis we put on the score of the edge vs. the reduction it causes in the graph. The experimental results suggest that it is important to use distributions that bias the algorithms towards high probability edges, since these edges also offer high reduction, while retaining high accuracy.

Our observations hold for all datasets, for different recommendation algorithms, and different values of k. Table 3 shows detailed results for the cubic probability function. We observe that *Greedy* obtains a considerable decrease in the shortest paths with an acceptable loss in accuracy.

6.6 Hybrid link recommendations

In this set of experiments, we evaluate a *hybrid* recommendation system that aims to combine high accuracy recommendations with high reduction edge additions. In this hybrid link selection method, given a node u, a set C_u of candidate edges, and a value k, we generate a recommendation set S of size k, by taking the set $S_{MostProb}$ with the $\ell, \ell < k$, most probable edges in C_u, and a set $S_{TopEdges}$ with the top $k - \ell$ edges recommended by the *TopEdges* algorithm from the set $C_u \setminus S_{MostProb}$.

We run experiments with the cubic function for all pairs of datasets and recommendation algorithms, for $k = 5$. Table 4 shows our results in detail, for all $0 \leq \ell \leq k$. There is a clear tradeoff between prediction accuracy and actual reduction. As we recommend more edges from the set of the most probable edges, the prediction accuracy $\widehat{\mathcal{A}}$ increases, while the actual reduction value $\widehat{\mathcal{R}}$ decreases.

To evaluate the performance of the hybrid recommendations for different values of ℓ we also compute the fraction $\widehat{\mathcal{RA}}(\ell) = (\widehat{\mathcal{A}}(\ell) - \widehat{\mathcal{A}}(0))/(\widehat{\mathcal{A}}(5) - \widehat{\mathcal{A}}(0))$, where $\widehat{\mathcal{A}}(\ell)$ is the prediction accuracy for a given value of ℓ. This quantity indicates where the value $\widehat{\mathcal{A}}(\ell)$ lies relatively to the minimum and the maximum value of $\widehat{\mathcal{A}}$. Respectively, for the actual reduction $\widehat{\mathcal{R}}$, we compute the fraction $\widehat{\mathcal{RR}}(\ell) = (\widehat{\mathcal{R}}(\ell) - \widehat{\mathcal{R}}(5))/(\widehat{\mathcal{R}}(0) - \widehat{\mathcal{R}}(5))$. In the last two columns of Table 4 we report the average values of $\widehat{\mathcal{RA}}(\ell)$ and $\widehat{\mathcal{RR}}(\ell)$. For instance, the recommendation for $\ell = 3$, on average, the prediction accuracy increase is equal to the 69.3% of the maximum increase, while the actual reduction is 71.6% of the maximum reduction. We find that the value $\ell = 3$ offers the best tradeoff between accuracy and reduction.

6.7 Efficiency

In our final set of experiments we evaluate the efficiency of our algorithms. All algorithms were implemented in `C++` and compiled using the `g++v.4.6.4` compiler. 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

Table 5: Running times (in milliseconds) on the *Facebook* and *DBLP* datasets for $|C| = 20$ and $k = 5$.

	Facebook		*DBLP*	
methods	*AA*	*RPR*	*AA*	*RPR*
Greedy	253	455	109	304
TopEdges	69	99	25	35
D-Greedy	195	313	87	137

processor has 4 cores sharing a 8MB L3 cache, and each core has a 256KB private L2 cache and 2.40GHz speed.

We assess the efficiency of the *Greedy*, *D-Greedy*, and *TopEdges* methods. We exclude the *MostProb* method since it requires just sorting of 20 values. The running times of the algorithms (in milliseconds) are shown in Table 5 for $k = 5$. Clearly, the *Greedy* algorithm is the slowest, with the *D-Greedy* method a close second. The *TopEdges* algorithm performs significantly faster that both, and it always runs in less than 0.1 seconds for all our test cases.

7. CONCLUSIONS

In this paper, we considered the problem of recommending links to network users that are of interest to them, and also improve their centrality in the network. We proposed an efficient greedy algorithm for this task that exploits a fast algorithm for computing the expected reduction caused by the addition of a probabilistic link in a network. Our experiments show that our approach strikes a balance between the prediction accuracy of the recommendations, and their effect on the centrality of the user in the resulting network.

For future work, it would be interesting to extend our proposed solution to consider recommendations that minimize the expected *all-pais* shortest path lengths. This natural extension considers the utility of the network as a whole when recommending edges to a specific user. It would also be interesting to consider recommendations that optimize other criteria beyond shortest paths. Finally, in our work we touched upon the problem of transforming link recommendation scores into probabilities. This is an interesting problem that merits further investigation.

Acknowledgments

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

8. REFERENCES

[1] L. A. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25(3), 2003.

[2] L. Backstrom and J. Leskovec. Supervised random walks: predicting and recommending links in social networks. In *WSDM*, pages 635–644, 2011.

[3] V. Chaoji, S. Ranu, R. Rastogi, and R. Bhatt. Recommendations to boost content spread in social networks. In *WWW*, pages 529–538, 2012.

[4] P. Crescenzi, G. D' Angelo, L. Severini, and Y. Velaj. Greedily improving our own centrality in a network. In *SEA*, pages 43–55. 2015.

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

[6] H. Frank. Shortest paths in probabilistic graphs. *Operations Research*, 17(4):583–599, 1969.

[7] A. Frieze and G. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57 – 77, 1985.

[8] V. Ishakian, D. Erdös, E. Terzi, and A. Bestavros. A framework for the evaluation and management of network centrality. In *SDM*, pages 427–438, 2012.

[9] G. Kollios, M. Potamias, and E. Terzi. Clustering large probabilistic graphs. *IEEE Trans. Knowl. Data Eng.*, 25(2):325–336, 2013.

[10] K. Lazaridou, K. Semertzidis, E. Pitoura, and P. Tsaparas. Identifying converging pairs of nodes on a budget. In *EDBT*, 2015.

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

[12] D. Li, Z. Xu, S. Li, X. Sun, A. Gupta, and K. Sycara. Link recommendation for promoting information diffusion in social networks. In *WWW (Companion Volume)*, 2013.

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

[14] D. Liben-Nowell and J. M. Kleinberg. The link prediction problem for social networks. In *ACM CIKM*, pages 556–559, 2003.

[15] A. Meyerson and B. Tagiku. Minimizing average shortest path distances via shortcut edge addition. In *APPROX*, pages 272–285, 2009.

[16] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of the approximations for maximizing submodular set functions. *Mathematical Programming*, pages 265–294, 1978.

[17] M. Papagelis. Refining social graph connectivity via shortcut edge addition. *TKDD*, 10(2):12, 2015.

[18] M. Papagelis, F. Bonchi, and A. Gionis. Suggesting ghost edges for a smaller world. In *CIKM*, pages 2305–2308. ACM, 2011.

[19] N. Parotsidis, E. Pitoura, and P. Tsaparas. Selecting shortcuts for a smaller world. In *SDM*, 2015.

[20] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment*, 3(1-2):997–1008, 2010.

[21] A. E. Sariyüce, K. Kaya, E. Saule, and Ü. V. Çatalyürek. Incremental algorithms for network management and analysis based on closeness centrality. *CoRR*, abs/1303.0422, 2013.

[22] H. Tong, B. A. Prakash, T. Eliassi-Rad, M. Faloutsos, and C. Faloutsos. Gelling, and melting, large graphs by edge manipulation. In *CIKM*, pages 245–254, 2012.

[23] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[24] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi. On the evolution of user interaction in facebook. In *WOSN*, pages 37–42, 2009.

[25] Z. Yin, M. Gupta, T. Weninger, and J. Han. A unified framework for link recommendation using random walks. In *ASONAM*, pages 152–159, 2010.