A Framework for the Evaluation and Management of Network Centrality [*]

Vatche Ishakian[†] Dóra Erdős[‡] Evimaria Terzi[§] Azer Bestavros[¶]

Abstract

Network-analysis literature is rich in node-centrality measures that quantify the centrality of a node as a function of the (shortest) paths of the network that go through it. Existing work focuses on defining instances of such measures and designing algorithms for the specific combinatorial problems that arise for each instance. In this work, we propose a unifying definition of centrality that subsumes all path-counting based centrality definitions: e.g., stress, betweenness or paths centrality. We also define a generic algorithm for computing this generalized centrality measure for every node and every group of nodes in the network. Next, we define two optimization problems: k-Group Centrality Maximization and k-Edge Centrality Boosting. In the former, the task is to identify the subset of k nodes that have the largest group centrality. In the latter, the goal is to identify up to k edges to add to the network so that the centrality of a node is maximized. We show that both of these problems can be solved efficiently for arbitrary centrality definitions using our general framework. In a thorough experimental evaluation we show the practical utility of our framework and the efficacy of our algorithms.

1 Introduction

The notion of centrality of the nodes in a network has been key in analyzing different types of network data; e.g., online social and media networks, Internet, communication networks, transportation networks and many more. For example, the centrality of a node in a social network often correlates with the influence of the node on the network. Similarly, high-centrality nodes in the Internet and other communication networks are nodes that impact/shape most of the observed traffic, while in transportation networks they correspond to hubs.

Measures for quantifying the centrality of a node in a network date back to the 1950s, when Shimbel [19] proposed that the centrality of a node should be the total number of shortest paths that go through it. Ever since, researchers have proposed different measures of centrality, as well as algorithms for computing them [1, 2, 4, 10, 11]. The common characteristic of these measures is that they quantify a node's centrality by computing the number (or the fraction) of (shortest) paths that go through that node. For example, in a network where packets propagate through nodes, a node with high centrality is one that "sees" (and potentially controls) most of the traffic.

In many applications, the centrality of a single node is not as important as the centrality of a group of nodes. For example, in a network of lobbyists, one might want to measure the combined centrality of a particular group of lobbyists. For such applications, the notion of centrality needs to be generalized to measure the centrality of groups rather than single nodes. The technical difficulty [3, 4] of such generalization comes from the fact that the centrality of a group is not simply the sum of the centralities of its members; two nodes that lie on the same set of shortest paths have the same centrality as the centrality of a group consisting of one of these nodes.

In many applications, it is importantly to discover a central set of nodes. For example, consider an Internet service provider that wants to identify a set of nodes for effective placement of traffic-monitoring devices. Or alternatively, consider advertisement campaign designers that want to identify the set of nodes in a traffic network for putting gigantic advertisement boards so that they influence large proportion of the drivers. In such cases, the goal is not to measure the centrality of a group of nodes, but rather to identify the group with the highest centrality. We call this high-level problem the k-Group Centrality Maximization (k-GCM) problem. Of course, different definitions of centrality lead to different instantiations of the k-GCM problem.

Finally, consider real-life applications where entities have a special interest in the increase (or decrease) in the centrality of a network node. For example, consider a network of airports that connect different cities. In this case, a city's council, having as a goal to increase the centrality of the local airport might lobby towards more airline companies directing flights to and from the city's airport. In this case, the question is which edges

[*]This research was supported in part by NSF awards #0720604, #0735974, #0820138, #0952145, #1012798, and #101752.

[†]Boston University visahak@bu.edu
[‡]Boston University edori@bu.edu
[§]Boston University evimaria@bu.edu
[¶]Boston University best@bu.edu

need to be added in a graph so that the centrality of a particular node increases as much as possible. We call this problem the k-Edge Centrality Boosting (k-ECB) problem.[1] This problem is an instance of centrality-management problems, where the goal is to alter the centrality of a group of nodes by making local structural changes in the input graph. Again, different definitions of centrality give rise to different instantiations of the k-ECB problem.

Our first main contribution is that we provide a generalized framework for computing the centrality of a single node in a given network. That is, we show that existing centrality definitions are *specializations* of the following definition of centrality: "the centrality of a node is the number of *special paths* that go through it". Different definitions of special paths lead to different notions of node centrality. This observation allows us to design a generic algorithm for computing the centrality of a node under this general definition of centrality. This generic algorithm can then be instantiated trivially to account for the peculiarities of different notions and measures of centrality.

Our second main contribution is that we use the above framework to solve any instantiation of the k-Group Centrality Maximization problem. Although variants of this problem have been defined in the past [6, 9], we are the first to solve it for arbitrary centrality measures and provide efficient, constant-factor approximation algorithm for the problem.

Finally, we show that using our framework, one can formalize a new class of problems that we call *centrality-management problems*. We define and solve one such problem, namely the k-Edge Centrality Boosting problem. We instantiate the problem using different notions of centrality and we show that our framework can be again utilized to solve all these problems. To the best of our knowledge we are the first to introduce and solve this problem for network centrality.

Paper Outline: The rest of the paper is organized as follows: in Section 2 we present an extensive discussion of the related work. In Sections 3 and 4 we present the different notions of centrality and algorithms for computing them. In Section 5 we define and solve the k-Group Centrality Maximization problem and in Section 6 we define and solve the k-Edge Centrality Boosting problem. In Section 7, we present a thorough experimental evaluation of our framework and we conclude the paper in Section 8.

[1]While our approach for managing centrality allows for addition as well as removal of edges, and allows for maximizing as well as minimizing centrality, in this paper we restrict our attention to the addition of edges for the purpose of maximizing centrality.

2 Related Work

Ever since Shimbel [19] quantified the "importance" of a node in a network by defining stress centrality as the total number of shortest paths passing through that node, several measures of centrality of nodes have been introduced. For example, Anthonisse [1] and Freeman [10] introduced the notion of betweenness centrality to be the fraction of shortest paths passing through nodes. More recently, Goh *et al.* [11] introduced load centrality in order to to compute the load on a node in packet-switched networks. Other notions of centrality include current-flow centrality [5, 16] and bridging centrality [12, 17]. Borgatti [2] and Brandes [4] provide some excellent surveys on existing node-centrality measures. Our work does not aim towards proposing yet-another measure of node centrality. Rather, we focus on providing a general algorithmic framework that allows us to compute many of the existing centrality measures using slight modifications of a generic (template) algorithm. At the same time, we are interested in applying this framework in solving optimization problems, e.g., k-Group Centrality Maximization and k-Edge Centrality Boosting, rather than simply computing the centrality for every node.

The algorithmic challenge of efficiently computing the centrality of an arbitrary node has been considered by Brandes [3, 4], who proposed an algorithm (and slight modifications thereof) to compute different variants of betweenness and stress centrality. The algorithms in Brandes' work are instances of our general algorithm to compute betweenness and stress centrality. The novelty of our work is that it provides a general algorithm to compute centrality that can be applied to several different types of centrality.

The notion of group centrality, i.e., the centrality of a group of nodes, has been also proposed by Everett *et al.* [8]. In their work, they provide a description of group-centrality measures that extend previous node centrality measures for groups. Brandes [4] gives an algorithm for computing group betweenness centrality, while Puzis [18] defines the optimization problem for identifying the group of nodes with the largest betweenness centrality. Puzis' algorithm is a branch-and-bound approach that searches the space of all possible groups. Dolev *et al.* [6] suggest a greedy heuristic to find a group with maximum betweenness centrality and show its approximation ratio, while Fink *et al.* [9] generalizes this to the probabilistic version of the problem, which maximizes expected group centralities. Although very related, all the above papers focus on a particular group-centrality measure, namely betweenness centrality. In a way, our work is a generalization the existing literature since we provide an umbrella framework that allows us to define and compute different group-centrality measures for different groups. At the same time, we can utilize this framework to also solve the k-Group Centrality Maximization optimization problem for different centrality measures. Even further, our generic algorithm for solving k-Group Centrality Maximization provides a constant-factor approximation algorithm for all centrality measures we consider.

Finally, the question of modifying the centrality of a node or a group of nodes, by changing the structure of the underlying graph has been posed by Everett *et al.* [7]. However, in their work, they only allude to this question without actually formalizing or solving it. To the best of our knowledge, we are the first to investigate the k-Edge Centrality Boosting problem and provide algorithms for solving it.

3 Node Centrality

In this paper, we examine the notion of centrality in *directed, acyclic* graphs (DAGs). Let $G(V, E)$ be such a graph. We assume that some nodes in V have special roles assigned; i.e., some of the nodes are *sources* or *destinations*. Throughout the paper we denote the set of sources by $S \subseteq V$ and the set of destinations by $T \subseteq V$.

Since our graph is acyclic, there is a topological order of the nodes. For a node v, we define the *ancestors* (resp. descendants) of v to be the nodes that are before (resp. after) v in the topological order and there is a path from them to v.

Given a graph, and sets S and T, it is often required to find a set of "central" nodes that are along paths that connect the nodes in S to the nodes in T. Such central nodes may be the nodes that are in many (shortest) paths between sources and destinations. There has been a lot of work on defining measures of a node's *centrality*. The common trait in these definitions is, that they all focus on counting the number (or the fraction) of a set of "important" paths that go through this node. The higher this number the higher the centrality of the node. In the literature, centrality of a node v is often computed as the number of (shortest) paths between *all* node pairs in the graph, that v is on. Our definition is a generalization of that, since we have no restriction on S and T. Both sets could contain the whole of V.

For completeness, we summarize some of these definitions below.

`#SP` **centrality:** For a node v, the `#SP` centrality of v, denoted by $C_{sp}(v)$, is the number of shortest paths between node pairs $(s, t) \in S \times T$ that contain v. Such a centrality measure is particularly important in communication networks where shortest-path routing is used for the transmission of information. In such networks, disabling nodes with high $C_{sp}(v)$ value causes the largest harm in the network. Similar to the above is the notion of **betweenness** centrality. The betweenness centrality of node v is the fraction of shortest paths from S to T that contain v.

`#P` **centrality:** For a node v, the `#P` centrality of v, denoted by $C_p(v)$ is the number of paths between nodes in S and T that contain v. This measure is also known as **stress** centrality.

Both the above definitions of centrality count the number of (shortest) paths between sources and destinations that a node is on. In addition to the above, there are also measures of centrality that have more of a set-cover flavor. In these measures, the centrality of a node v is determined by whether or not v is in at least one (shortest) path from nodes in S to nodes in T. We summarize these centrality definitions below.

`SP` **centrality:** For a node v, the `SP` centrality of v, denoted by $C_{1sp}(v)$, is the number of node pairs $(s, t) \in S \times T$ for which v is on at least one shortest paths between s and t. The notion of `SP` centrality has applications in transportation; people prefer to travel on the shortest route to their destination. Thus, for transportation companies it makes sense to establish major hubs in cities that lie on at least one shortest route for many destinations.

`P` **centrality:** For a node v, the `P` centrality of v, denoted by $C_{1p}(v)$, is the number of node pairs $(s, t) \in S \times T$ for which v is on at least one paths between s and t. Typically, broadcast messages travel through all paths in a network. For purposes of collecting information about the general state of the network (for example learning about congestion or inactive links), it is enough to receive at least one of these broadcast messages.

Recall, that our focus in this paper is on *directed* and *acyclic* graphs. This choice is informed both by motivating applications and the notions of centrality themselves. Specifically, `#P` and `P` centrality cannot be defined on graphs, that contain cycles. In case there was a cycle in the graph, a path could contain the edges of the cycle arbitrary many times. It would not be clear, what the centrality of a node in the cycle would be. The two measures `#SP`, `SP`, and betweenness centrality do not suffer from this problem, since shortest paths cannot contain any cycles. For this reason, without loss of generality we can assume the acyclicity of the graph.

3.1 General framework for centrality.

While the above centrality measures are all slightly different and have different uses, they have many common properties. In this section we highlight these common traits and show how these characteristics allow these measures to fit in a common framework.

As before, let $G(V,E)$ be a directed, acyclic graph with source and destination sets of nodes S and T respectively. The common characteristic in the above centrality measures is that each corresponds to the number of special-type of paths between S and T that a node v is on. Let PROP be a property, that a directed path may have (e.g. it is a shortest path, or it is a simple directed path, etc.). PROP depends on the specific centrality measure. Let $\mathcal{P}$ be a subset of directed paths in G that connect nodes in S to nodes in T and which have property PROP. We refer to the paths in $\mathcal{P}$ as *special paths*. Observe that for the measures #SP, betweenness and SP centrality $\mathcal{P}$ contains the set of shortest paths between S and T. In the case of #P and P centrality, $\mathcal{P}$ consists of all directed paths from S to T.

We denote by $\mathcal{P}(s,t)$ the set of special paths, that have nodes $s \in S$ and $t \in T$ as their endpoints. We say that v *covers* a path p in $\mathcal{P}(s,t)$, if v is a node on p. The set $\mathcal{P}_v(s,t)$ denotes the special paths in $\mathcal{P}(s,t)$ that are covered by v. We define the *centrality* $C(v)$ of a node v as a function $\mathcal{F}$ of $\mathcal{P}$.

$$C(v) = \sum_{(s,t)\in S\times T} \mathcal{F}(\mathcal{P}_v(s,t)). \tag{3.1}$$

For the first type of centrality measures, i.e., #SP and #P, the function $\mathcal{F}$ is simply the number of special paths v covers. That is,

$$C(v) = \sum_{(s,t)\in S\times t} |\mathcal{P}_v(s,t)|.$$

In the case of SP and P centrality, $\mathcal{F}$ is an indicator function that takes value 1 for a source-destination pair s,t $|\mathcal{P}_v(s,t)| > 0$. That is,

$$C(v) = \sum_{(s,t)\in S\times t} \delta(|\mathcal{P}_v(s,t)| > 0).$$

4 Computing Node Centrality

In this section we describe how to compute the centrality of a node v with respect to the general definition of $C(v)$. At the end of the section, we show how this computation can be used to compute specific centrality measures.

4.1 A generic computation of the impact of a node. We denote the number of distinct directed paths from any node x to y by $\#\text{PATHS}(x,y)$. For a set $X \subseteq V$, let $\#\text{PATHS}(X,y) = \sum_{x\in X} \#\text{PATHS}(x,y)$ denote the number of paths starting in X. Then $\#\text{PATHS}(S,v)$ denotes the number of distinct paths that lead from any source to v. We call this the PREFIX of v, and by definition $\text{PREFIX}(v) = \sum_{s\in S} \#\text{PATHS}(s,v)$. We denote by $\text{SUFFIX}(v)$ the total number of distinct directed paths, that start in v and end in T: $\text{SUFFIX}(v) = \sum_{t\in T} \#\text{PATHS}(v,t)$. Observe, that the total number of source-destination paths that v covers is equal to the product of the two values:

$$I(v) = \text{PREFIX}(v) \times \text{SUFFIX}(v).$$

We call $I(v)$ the *impact* of v. We will use the same terminology later: the impact of v is the number of special paths that v covers.

In order to compute the impact, we need to compute the PREFIX and SUFFIX of every node. For computing the PREFIX we rely on the observation that any path from a source s to a node v has go through one of v's parents (s may be one of the parents of v). This implies, that $\#\text{PATHS}(s,v)$ is equal to the total number of distinct paths leading to the parents of v.

Let Π_v denote the set of parents of v. Then $\text{PREFIX}(v)$ can be computed with equation (4.2).

$$\text{PREFIX}(v) = \sum_{x\in\Pi_v} \text{PREFIX}(x). \tag{4.2}$$

We need to evaluate (4.2) sequentially, by first computing the PREFIX of v's ancestors. To do this, we fix a *topological* order σ of the nodes. (A topological order of nodes is an order in which every edge is directed from a smaller to a larger ranked node in the ordering.) This order naturally implies that the parents of a node precede it in the ordering. Formula (4.2) can now be evaluated, while traversing the nodes of G in the order of σ.

Recall that the PREFIX of a node can also be expressed as $\#\text{PATHS}(S,v)$, thus formula (4.2) is equivalent to

$$\begin{aligned} \text{PREFIX}(v) &= \#\text{PATHS}(S,v) \\ &= \sum_{x\in\Pi_v} \#\text{PATHS}(S,x). \end{aligned} \tag{4.3}$$

As we established before, $\text{SUFFIX}(v)$ is equivalent to the total number of directed paths starting from v. This can be computed efficiently by doing some bookkeeping during the sequential computation of (4.2): For every node v, we maintain a list, PLIST_v, that contains for every ancestor y of v the number of paths that go from y to v. Thus, $\text{PLIST}_v[y] = \#\text{PATHS}(y,v)$. Observe now, that for an ancestor y of v, $plist_v[y]$ can be computed as the sum of the PLIST of the parents (see Formula (4.4)).

$$\forall y \in V : \text{PLIST}_v[y] = \sum_{x\in\Pi_v} \text{PLIST}_x[y]. \tag{4.4}$$

Observe, that for every node, $\textsc{plist}_v$ can be computed during the same recursion as (4.2).

To compute the $\textsc{Suffix}$ of a node v, we need to sum the number of paths that start in v and end in T. This is simply the sum of the $\textsc{plist}_t$ entries, that correspond to v for every $t \in T$ (see Formula (4.5)).

$$\text{(4.5)} \qquad \textsc{Suffix}(v) = \#\textsc{paths}(v, T) = \sum_{t \in T} \textsc{plist}_t[v].$$

As a technical detail, in order to use this recursive formula, every node's $\textsc{plist}$ contains itself with value one: $\textsc{plist}_v[v] = 1$. As a special case, a sources list would contain only the entry corresponding to itself.

The topological order σ of the nodes can be computed in linear time. Formulas (4.2) and (4.4) are updated along every edge of the graph. Formula (4.2) can be updated in constant time along an edge, while formula (4.4) requires $O(\Delta)$ lookups and additions. (Where Δ corresponds to the maximal degree int he graph.) $\textsc{Suffix}(v)$ can be computed by doing $|T|$ lookups in the $\textsc{plist}$'s of T and using formula (4.5). This yields a total running time of $O(|E| \cdot \Delta)$ to compute the impact of every node. This can be $O(n^3)$ in worst case, but in practice, for most graphs $|E| < O(n \cdot \log n)$, which results in a $O(n \cdot \log n)$ running time.

4.2 Computing centrality.

In this section, we show how the notion of impact can be tailored to the different centrality measures, to compute the centrality of a node efficiently. In general, the centrality of every node $v \in V$ will be equal to the impact of v, where the impact is computed by the formula $I(v) = \textsc{Prefix}(v) \times \textsc{Suffix}(v)$. The difference between the different centralities shows only in the way formulas (4.2), (4.4) and (4.5) are computed.

#SP centrality: Recall that $C_{sp}(v)$ denotes the number of *shortest* paths between pairs $(s, t) \in S \times T$. To compute this we only have to add a simple criterion when computing $\textsc{Prefix}(v)$ in formula (4.2) and $\textsc{Suffix}(v)$ in formulas (4.4) and (4.5): observe that a path $(s, x_1, x_2, \ldots x_r, v)$ from source s to node v can only be a shortest path if $(s, x_1, x_2, \ldots x_i)$ is also a shortest path for every intermediate node x_i. Thus, when we compute the sums in these formulas, we only add the values for the subset $\Pi'_v \subseteq \Pi_v$ of parents, that are on a shortest paths. Thus, formula (4.2) becomes

$$\textsc{Prefix}(v) = \sum_{x' \in \Pi'_v} \textsc{Prefix}(x').$$

Similarly, formula (4.4) is replaced by

$$\forall y \in V : \textsc{plist}_v[y] = \sum_{x' \in \Pi'_v} \textsc{plist}_{x'}[y].$$

Observe, that using the $\textsc{plist}$ values corresponding to the number of shortest paths is sufficient to compute the $\textsc{Suffix}$ for shortest paths in formula (4.5). The set Π'_v can be found by comparing the distance $d(s, v)$ of v from s and $d(s, x)$ for every candidate parent $x \in \Pi_v$. Node x lies along a shortest path to v if and only if $d(s, v) - d(s, x) = 1$. The distances $d(s, v)$ only need to be computed once. Observe that the shortest paths from different source nodes to a node v are different (and may have different length). For this reason we have to compute the impact of a node v for every source $s_i \in S$ separately and then aggregate those. Thus, if we denote the $\textsc{Prefix}$ and $\textsc{Suffix}$ corresponding to source s_i with a subscript i (thus $\textsc{Prefix}_i(v)$ and $\textsc{Suffix}_i(v)$), then

$$\text{(4.6)} \qquad I(v) = \sum_{s_i \in S} \textsc{Prefix}_i(v) \times \textsc{Suffix}_i(v).$$

The *betweenness* $C_B(v)$ of a node with regard to a pair (s, t) can be computed by dividing $C_{SP}(v)$ by the total number of shortest paths.

#P centrality: Recall that $C_p(v)$ corresponds to the number of distinct paths from S to T that v covers. Observe that this is the same notion as the impact of v, thus $C_p(v) = I(v)$.

SP centrality: For evaluating $C_{1sp}(v)$, when computing (4.2) and (4.4) we only add values for parents, that are on a shortest paths from s to v, and we use boolean addition, when doing so.

P centrality: Recall that the value of $C_{1p}(v)$ is the number of *pairs* $(s, t) \in S \times T$, which v covers. For a given source node s we are only interested if node v is along at least one paths starting in s. Thus, when computing formulas (4.2) and (4.4) we only do a boolean addition. This way when we compute $\textsc{Suffix}(v)$ in formula (4.5) (we emphasize, that here we use the conventional integer addition) we get the exact number of destinations for which v covers at least one paths between s and the destination. Similar to the shortest paths, we need to do this computation separately for every source node and then aggregate the results as in formula (4.6).

5 Group Centrality

Many applications make use of the combined centrality of a set of nodes. For this, we generalize the centrality measures to measure the centrality of a group of nodes, rather than individual nodes. We call these generalized versions of centrality measures *group-centrality* measures.

Let $A \subseteq V$ be a subset of nodes. Let $\mathcal{P}$ be the set of special path with property $\textsc{prop}$. We say that the group centrality $C(A)$ of set A is the number of special paths that nodes in A participate in.

$$C(A) = \sum_{(s,t)\in S\times T} \mathcal{F}(\mathcal{P}_A(s,t)).$$

5.1 Finding the most-central group of nodes. In many applications the goal is to find a set of nodes that are the most central amongst other groups with the same cardinality. This leads to a straightforward definition of the optimization problem.

Problem 5.1. (k-Group Centrality Maximization (k-GCM)). *Let $G(V,E)$ be a directed graph with sources $S \subseteq V$ and destinations $T \subseteq V$ and $\mathcal{P}$ the set of special paths in G. For integer k, find a set $A \subseteq V$ of size $|A| \leq k$, such that $C(A)$ is maximized.*

Different centrality measures lead to different versions of the k-GCM problem. We denote these by k-GCM(`#SP`), k-GCM(`#P`), k-GCM(`SP`), k-GCM(`P`) for `#SP`, `#P`, `SP` and `P` centralities respectively, We have the following result for the complexity of these instantiations.

Theorem 5.1. *The k-GCM(`#SP`), k-GCM(`#P`) and k-GCM(`SP`) problems are NPcomplete.*

The NPcompleteness proofs for all these problems are provided in the Appendix (Section 8) of this paper.

5.2 Approximating the k-GCM problem. In this section, we present a generic algorithm for approximating the k-GCM problem. This algorithm can be instantiated appropriately to solve k-GCM(`#SP`), k-GCM(`#P`) and k-GCM(`SP`) problems; all with the same approximation factor.

Our algorithm is a greedy heuristic, which expands the current set $A \subseteq V$ in every iteration with a node v, that results in the highest increase $I_A(v) = C(A \cup \{v\}) - C(A)$ of group centrality. We define the *conditional impact* $I_A(v)$ of node v with regard to group A as the increase in coverage by adding v to the group. We will see that $I_\emptyset(v) = I(v)$, the impact we defined in Section 4.1. Moreover, the computation of $I_A(v)$ requires the use of Formulas (4.2), (4.4) and (4.5) in the same way as in Section 4.1, with only slight modifications. The `Greedy` algorithm we propose (Algorithm 5.1) iterates the recalculation of the impact $I_A()$ and the choice of the (currently) highest impact node k times.

Algorithm 5.1.

```
**Input:** $G(V,E)$ and integer $k$.
**Output:** set of nodes $A \subseteq V$, where $|A| \leq k$.
$A = \emptyset$
**for** $i = 0 \ldots k-1$ **do**
    **for** $j = 1 \ldots n$ **do**
        compute $I_{A_i}(v_j)$
    $A_{i+1} = A_i \cup \{\text{argmax}_{v\in V} I_{A_i}(v)\}$
```

Approximation. The objective function $C(A)$ in the optimization Problem 5.1 is positive, monotone increasing, and submodular. Thus, by a theorem of Nemhauser *et al.* [15] our `Greedy` algorithm yields an $(1-\frac{1}{e})$-approximation for the optimal solution.

Updating the impact efficiently. Let us assume that at the start of iteration i the `Greedy` algorithm has already chosen nodeset $A_i \subseteq V$. In this iteration we need to pick a node v that covers the largest possible number of special paths, that were not covered by A_i. This is the value $I_{A_i}(v)$ that we mentioned above. Let $\mathcal{P}_{A_i} \subseteq \mathcal{P}$ be the set of special paths that are covered by A_i. Observe, that if a path $p \in \mathcal{P}_{A_i}$, then it is already covered, thus it is not counted in the conditional impact of the nodes. Let us assume that node $a \in A_i$ lies on p (other nodes in A_i might also cover it). Since none of the special paths is counted in the conditional impact, that are covered by node a, we would get the same value for the conditional impact, if we removed a from the graph. We introduce the notion of conditional $\textsc{Prefix}_{A_i}(v)$ and $\textsc{Suffix}_{A_i}(v)$, that are computed with this idea of node removing in mind. The algorithm proposed in Section 4.1 is used to compute the conditional impact in the following way. When $\textsc{Prefix}_{A_i}(v)$ is computed by Formula (4.2), the $\textsc{Prefix}$ values for nodes in A_i are replaced by 0. Thus, $\textsc{Prefix}_{A_i}(a) = 0$ for $a \in A_i$, and this 0 value is used for the $\textsc{Prefix}$ computation of a's children. Similarly, when computing $\textsc{plist}^i_v$, all entries in $\textsc{plist}^i_v(a)$ are set to 0 and this zero value is used, when evaluating Formula (4.4). Using these conditional $\textsc{plist}$ values in Formula (4.5) result in the conditional $\textsc{Suffix}_{A_i}(v)$, where paths going through A_i are not counted towards the $\textsc{Suffix}$. Observe that, $I_{A_i}(v) = \textsc{Prefix}_{A_i}(v) \times \textsc{Suffix}_{A_i}(v)$ counts exactly the number of paths covered by v and NOT covered by any node in A_i. The computation of the conditional impact takes the same time as in the case of the unconditional. Thus, the running time for one iteration of the `Greedy` algorithm is $O(|E| \cdot \Delta)$. Observe also that adding a new node v to the set A_i changes the conditional $\textsc{Suffix}$ of all nodes before v and the $\textsc{Prefix}$ of all nodes after v in the topological order. Thus, the conditional impact needs to be recomputed in every iteration. This results in a total running time of $O(k \cdot |E| \cdot \Delta)$ for k iterations.

Observe that once the `Greedy` algorithm needs to compute the impact of nodes at different iterations. Depending on the particular centrality measure we adopt, the Greedy algorithm will use the appropriate impact computation for this measure, as described in Section 4.2. In that respect, Greedy is a generic algorithm for the k-GCM problem.

5.3 Computational speedups. In this section, we outline some of the techniques for reducing the computational aspects of our approach. All these heuristics are inspired by the Greedy algorithm, but approximate some of its steps and have significantly smaller running times. We describe these algorithms below.

The G_max algorithm: This heuristic is a simplified version of Greedy where we do not take into account the interdependency of the centralities of nodes. We compute the impact of every node once and then choose the k nodes with highest centrality. The running time is the time it takes to compute the impact of every node once, which is $O(|E| \cdot \Delta)$

The G_1 algorithm: In this heuristic the k nodes with the highest product of $C_d(v) = d_{in}(v) \times d_{out}(v)$ are picked. This is our fastest heuristic and can be computed in $O(|E|)$ time. $C_d(v)$ is the number of paths of length two that go through v. In the literature $C_d(v)$ is sometimes referred to as *degree centrality.* In our experiments we show that degree centrality, despite its name, is different in nature from other centrality measures, and in fact not a good indicator of "centrality".

The G_Sampled algorithm: In this heuristic we create a sample of the original input graph and then run the Greedy algorithm on the smaller sampled graph. The sampling is based on random walks started simultaneously from both source and destination nodes. We stop the sampling once a subgraph is traversed, where 15% of the original $\{s, t\}$ source and destination pairs are connected by a path.

6 Managing Centrality

In this section, we consider the problem of increasing the centrality of one particual node $u^* \in V$ by adding edges in G. More specifically, we consider the optimization version where the problem is to identify the subset of edges to be added to G so that $C(u^*)$ is increased the most. Since edge additions change graph G, we will enhance the notation of centrality to take the underlying graph as an argument. That is for node v and graph G we use $C(v, G)$ to denote the centrality of node v in graph G.

Formally, we define the problem as follows:

Problem 6.1. (k-Edge Centrality Boosting (k-ECB)). *Given DAG $G(V, E)$, node $u^* \in V$ and integer k, find k edges E_k to form $G' = (V, E \cup E_k)$, such that $C(u^*, G')$ is maximized.*

As before, the k-ECB problem can be instantiated to k-ECB(#SP), k-ECB(#P), k-ECB(SP) and k-ECB(P) problems when the centrality of a node is measured using the #SP, #P, SP and P respectively. We reiterate that the k-ECB problem is a special case of the general problem of maximizing (or even minimizing) a node's centrality by addition or deletion of edges. We also note that maximization (resp. minimization) of the #P centrality of a node can be only achieved by edge additions (resp. deletions).

We again propose a generic algorithm for solving the k-ECB problem. We call this greedy-heurstic algorithm Greedy_Add. The Greedy_Add algorithm adds the edge $(u \rightarrow v)$ that maximizes the centrality of u^*. Thus $(u \rightarrow v) = \text{argmax}\{C(u^*, G \cup (u \rightarrow v))\}$. Repeating this k times results in our Greedy_Add algorithm 6.1.

Algorithm 6.1. Greedy_Add algorithm for edge addition

```
Input: $G(V, E)$ and node $u^* \in V$.
Output: edges $E' = (u_1, v_1) \cup (u_2, v_2) \cup \ldots (u_k, v_k)$.
$E' = \emptyset$
for i=1...k do
    $G' = G(V, E \cup E')$
    $(u_i, v_i) = \text{argmax}\{C(u^*, G' \cup (u \rightarrow v))\}$
    $E' \leftarrow (u_i, v_i)$
```

Instead of selecting among all possible edges to add, the Greedy_Add algorithm can be restricted to choose amongst candidate edges that have one of their endpoints in u^*. We refer to this variant of the Greedy_Add algorithm as the G_Add_1. Since G_Add_1 has fewer choices to consider in every iteration, it is naturally a more efficient algorithm than Greedy_Add. We discuss the running time of G_Add_1 towards the end of this section.

Properties of Greedy_Add. Although Greedy_Add performs very well in practice, we show here some theoretical evidence why it is not a constant-factor approximation algorithm for the different variants of the k-ECB problems.

#SP: Here we show that the #SP centrality a node u^*, $C_{sp}(u^*)$, is not monotonically increasing with respect to edge additions. Consider the example in Figure 1: Let u^* be the node, whose centrality we wish to increase. In the solid-edge graph, there are two shortest paths from s to t. Node u^* covers both, hence $C_{sp}(u^*, G) = 2$. However, by adding the dotted edge, the length of the shortest paths has decreased to 4, and the number of shortest paths to 1. Now u^* is covering only this shortest path, and thus $C_{sp}(u^*, G \cup (v \rightarrow w)) = 1$.

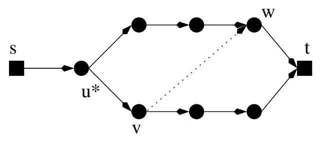


Figure 1: Adding edge $(v \rightarrow w)$ decreases the #SP centrality of node u^*.

#P: Here we show that eventhough the #P centrality of u^*, $C_p(u^*)$, is monotonically increasing with respect to edge additions, it is not submodular. Consider the solid-edge graph in Figure 2. The increase in centrality $C_p(u^*, G \cup (u \rightarrow v)) - C_p(u^*, G)$ caused by adding edge $(u \rightarrow v)$ in G is smaller, than the increase $C_p(u^*, G' \cup (u \rightarrow v)) - C_p(u^*, G')$ caused by adding the same edge $(u \rightarrow v)$ in $G' = G \cup (v \rightarrow w)$, in contrast to the definition of submodularity.

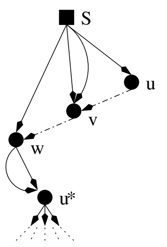


Figure 2: Adding edges (v, w) and $(u \rightarrow v)$ show, that $C_p(u^*, G)$ does not maintain submodularity during edge addition.

SP: The SP centrality of u^*, $C_{1sp}(u^*)$, is also not monotonic neither is it submodular with respect to edge additions. However, if we restrict edge additions to contain u^* as one of their endpoints, then SP is both monotonically increasing and submodular. To see this, let us assume that we add edge $(u^* \rightarrow v)$ to G and obtain G'. For every pair s, t, that the length $d(s, t|G')$ of shortest path between s and t is shorter than in G, it is true that u^* covers this shorter path. Thus the set of pairs for which u^* covers a shortest path in G' is a super set of those in G. This implies that the G_Add_1 algorithm is a constant-factor approximation algorithm for the k-ECB(SP) problem; the constant factor is as before $(1 - 1/e)$.

P: The P centrality of node u^*, $C_{1p}(u^*)$, is monotonically increasing with edge additions. This is because adding an edge only adds new paths to the set of special paths. However, the same function is not submodular (even when edges are restricted to have u^* as one of their endpoints). The example in Figure 2 can be again used to illustrate this.

6.1 Implementation details of Greedy_Add.

According to the definition, $C(u^*, G)$ is a function of the number of special paths, that are covered by u^*. Let us assume, that an edge $(u \rightarrow v)$ is added to G, resulting in graph $G' = G \cup (u \rightarrow v)$. Observe that G' contains all paths from S to T that are in G, and it may contain some additional paths. However, all additional paths will contain $(u \rightarrow v)$ as an edge. Thus, the centrality $C(u^*, G')$ of u^* in graph G' increases by the number of special paths, that are covered by u^* and contain edge $(u \rightarrow v)$. Since this increase in centrality is due to the addition of edge $(u \rightarrow v)$, we will call this value the *relative impact* of $(u \rightarrow v)$. Expressed with a formula, the relative impact of edge $(u \rightarrow v)$ is $I((u \rightarrow v)\,|u^*) = C(u^*, G \cup (u \rightarrow v)) - C(u^*, G)$.

The Greedy_Add algorithm for edge addition (Algorithm 6.1) can be reformulated, with help of the relative impact of node pairs. The edge $(u \rightarrow v)$ that maximizes $\text{argmax}\{C(u^*, G \cup (u \rightarrow v))\}$ can be found, by computing the relative impact of every node pair u, v and then choosing the largest. Once $(u \rightarrow v)$ is added to G, the centrality $C(u^*, G)$ and the relative impacts of potential edges need to be recomputed, before chosing the next edge to add. These steps can be repeated k times in order to add k edges.

Computing the relative impact. Here, we show how the relative impact of node pairs can be computed in a graph G. We describe the algorithm for the general centrality definition. This general algorithm can easily be adjusted to the specific centrality measures.

In order to compute the relative impact $I((u \rightarrow v)\,|u^*)$ we need to compute the number of paths that go through $(u \rightarrow v)$ and u^*. This suffices, because all special paths that are covered by u^* in G' but not in G are paths that contain edge $(u \rightarrow v)$ and node u^*. Let the notation $\#\text{PATHS}(x, y, z)$ denote the number of paths between x and z containing y. We define the relative PREFIX of a node u as $\text{PREFIX}^*_u(u) = \#\text{PATHS}(S, u^*, u)$. Observe, that $\text{PREFIX}^*_u(u) = \text{PREFIX}(u^*) \times \#\text{PATHS}(u^*, u)$, since it denotes the number of paths from S to u going through u^*. The relative SUFFIX is defined similarly, $\text{SUFFIX}^*_u(v) = \sum_{x \in V} \#\text{PATHS}(v, u^*, x)$. A similar observation can be made, $\text{SUFFIX}^*_u(v) = \#\text{PATHS}(v, u^*) \times \text{SUFFIX}(u^*)$.

Let us fix an arbitrary topological order σ of the nodes in V. Since G' is a DAG, it is impossible for a path to contain both edge $(u \rightarrow v)$ and u^* in G', if $\sigma(u) < \sigma(u^*) < \sigma(v)$. Thus, either both u and v have to be before or after u^* in the order of σ. Let $K(u \rightarrow v|u^*) = \text{PREFIX}(u) \times \text{SUFFIX}^*_u(v)$ and let $L(u \rightarrow v|u^*) = \text{PREFIX}^*_u(u) \times \text{SUFFIX}(v)$. Observe that $K(u \rightarrow v|u^*)$ is the relative impact of edge $(u \rightarrow v)$, if both u and v are before u^* in σ. Similarly $L(u \rightarrow v|u^*)$ denotes the relative impact of edge $(u \rightarrow v)$ if they are both after u^*. Now the relative impact can be computed as follows:

$$I((u \rightarrow v)\,|u^*) = \text{MIN}\{K(u \rightarrow v|u^*) + L(u \rightarrow v|u^*),\\ K(u \rightarrow v|u^*) \times L(u \rightarrow v|u^*)\}.$$

Observe, that this formula takes care of the case when $\sigma(u) < \sigma(u^*) < \sigma(v)$, since then $I((u \rightarrow v)\,|u^*) = 0$.

The computation of the relative impact of node pairs consists of three phases. Since the formula for evaluating $I((u \rightarrow v)\,|u^*)$ contains the general Prefix and Suffix of nodes, we need to compute that in the first phase, as described in Section 4.1. Second, we focus on computing Prefix^*_u and Suffix^*_u. For this we need to compute $\#\text{PATHS}(u, u^*)$ and $\#\text{PATHS}(u^*, v)$ for every node $u, v \in V$. Observe that $\#\text{PATHS}(u, u^*)$ can be determined by a simple lookup in PLIST^*_u; namely $\#\text{PATHS}(u, u^*) = \text{PLIST}^*_u[u]$. We assume that if u is not in the list, then $\text{PLIST}^*_u[u] = 0$. The PLIST can also be used to compute $\#\text{PATHS}(u^*, v)$; namely $\#\text{PATHS}(u^*, v) = \text{PLIST}_v[u^*]$. Now we know all terms needed to compute $K(u \rightarrow v|u^*)$ and $L(u \rightarrow v|u^*)$ and ultimately $I((u \rightarrow v)\,|u^*)$ in the third phase. This process is described in Algorithm 6.2.

Algorithm 6.2. Algorithm for computing the relative impact

```
**Input:** $G(V, E)$ and node $u^* \in V$.
**Output:** values $I((u \rightarrow v)\,|u^*)$ for every $u, v \in V$
Compute $\text{Prefix}(u)$ and $\text{Suffix}(v)$
Compute $\#\text{PATHS}(u, u^*)$ and $\#\text{PATHS}(u^*, v)$
Compute $K(u \rightarrow v|u^*)$ and $L(u \rightarrow v|u^*)$
Compute $I((u \rightarrow v)\,|u^*)$
```

Running time of Greedy_Add: This algorithm involves the computation of the Prefix and suffix of every node. As we have established in Section 4.1, this computation takes $O(\Delta \cdot |E|)$ time. The computation of $\#\text{PATHS}(u, u^*)$ and $\#\text{PATHS}(u^*, v)$ can be done along with the computation of Prefix and Suffix, and thus it does not increase the running time. At last we have to compute $K(u \rightarrow v|u^*)$, $L(u \rightarrow v|u^*)$ and $I((u \rightarrow v)\,|u^*)$ for every edge, which takes at most $O(n^2)$ steps. We repeat the algorithm k times, which results in a total running time of $O(k \cdot \Delta \cdot n^2)$.

Computational speedups: Since the running time of `Greedy_Add` has the same magnitude as that of `Greedy` we present here some computational speedup techniques that we apply in our experiments.

Sampling: We use the same sampling techniques to sample graphs as described in Section 5.3.

The `G_Add_max` algorithm: Similar to the `G_max` algorithm, this heuristic computes the relative impact of $(u \rightarrow v)$ edges once, and takes the k highest impact edges.

The `G_Add_1` algorithm: This algorithm is a simplified version of `Greedy_Add`, which can be computed significantly faster, and depending on the graph, can perform quite well compared to `Greedy_Add` (see Figure 4 in Section 7 for comparative experiments). The `G_Add_1` algorithm adds the edge adjacent to u^* with the largest relative impact. For this we need to compute $\text{Prefix}(v)$ for every node v preceding u^* in the topological order, and compute $\text{Suffix}(w)$ for every node w, succeeding u* in the topological order. Computing the Prefix and Suffix of the nodes takes $O(\Delta{\cdot}n)$ time. Since the edges adjacent to u^* are independent, we only need to compute these values once. Thus a set of k edges can be added in $O(\Delta \cdot n)$ time.

6.1.1 Increasing group centrality.

Instead of focusing on one node, an interesting generalization would be to study the effect of edge addition to group centrality. We believe that a greedy approach similar to `Greedy_Add` could be used to solve the problem of maximizing the centrality of a group $A \subseteq V$. Working out the details of this greedy algorithm and other heuristics are interesting problems that we are pursuing as follow-up work.

7 Experimental Evaluation

In this section, we demonstrate the utility of our framework by presenting a thorough experimental evaluation of the performance of the different alogrithms we propose.

Quote **dataset**: The Quote dataset [14] is a network of online media sites (e.g., news sites, blogs etc.). The links in the network are directed and they correspond to hyperlinks between the corresponding sites. Every hyperlink $u \rightarrow v$ is labeled with a quote, i.e., the piece of news that caused node u to connect to v. For the experiments we report here we select a particular quote: "lipstick on a pig" and pick the subgraph of the input graph that is defined by the edges that are labeled with this quote. Since sites may freely link to each other, the formed graph might contain cycles. We convert this graph into an acyclic graph as follows: From every node u we find a maximal acyclic subgraph using a DFS traversal, having u as its initiator. Then, we use the largest resulting DAG to work with. The DAG we end up with contains 21472 nodes and 81427 edges. We pick the set S to be the immediate neighbors of the initiator of the selected DAG; in this way we ended up with 36 source nodes. We formed the set of destinations T by picking 100 random nodes from the graph.

The maximum centrality group of nodes corresponds to media sites that are traversed by users the most in relationship to a specific phrase or idiom, and thus may be the best for placement of advertisement related to that idiom (e.g., by a political party).

TWITTER **dataset:** The TWITTER dataset [13] contains user ids and links between users, directed from the user to his/her followers. The complete dataset contains over 41 million user profiles. We again selected a subgraph by first running a breadth-first search up until six levels, starting from the user "sigcomm09". Our goal was to find a subnetwork of users related to the computer science community. For this, we created a list of keywords related to computer science, technology and academia and filtered the user profiles of the followers according to that. The resulting network is an acyclic graph with a single root "sigcomm09'. The graph contains about 90K nodes and 120K edges. The number of out-going edges from the different levels of the graph show an exponential growth: 2, 16, 194, 43993 and 80639 for levels 1,2,..., 5. We had to remove a small number of edges, in order to maintain an acyclic graph. Similar to the QUOTE dataset source selection, we drop the intial node corresponding to "sigcomm09" from the graph, which was connected to 2 nodes, and select 100 random destinations from the graph.

Similarly to the QUOTE dataset, central nodes in the twitter dataset corresponds to users that are on many information-propagation paths. In this dataset, nodes might want to increase their centrality by following other central nodes.

7.1 Evaluating algorithms for the k-GCM problem.

In this section we compare the performance of `Greedy` `G_max` and `G_1` algorithms by comparing the centrality of the groups of nodes they produce as solutions to the k-GCM problem. We also use the following algorithms as baseline:

In our random baseline algorithms, groups of size k in expectation are chosen according to the following heuristics:
(`Rand_K`): chooses k nodes from V uniformly at random.
(`Rand_I`): Every node is chosen with probability $\frac{k}{n}$.
(`Rand_W`): Every node v is chosen with probability $w(v) \times \frac{k}{n}$. The weight $w(v)$ is equal to weight $w(v) = \sum_{u \in C_v} \frac{1}{d_{\text{in}}(u)}$, where $C_v = \{u \in V | (v \to u) \in E\}$ is the set of children of v. The intuition behind this is, that the influence of node v on its child u, is inversely proportional to the in-degree of u.

To measure the performance, we define the *coverage ratio* (CR) as the fraction $\frac{C(A_{\text{other}})}{C(A_{\texttt{Greedy}})}$ – The closer this value is to 1, the better the performance of the algorithm. $A_{\texttt{Greedy}}$ is the group of size k chosen by the `Greedy` algorithm, and A_{other} corresponds to the group of (expected) size k chosen by the heuristic against which we compare `Greedy`. For the deterministic algorithms we simply report the achieved CR, while for random heuristics we report an average over ten runs.

The results from our comparative evaluation are shown in Figure 3, in which the X axis corresponds to the group size and the Y axis corresponds to CR. We report results on both the QUOTE and TWITTER datasets. For the former, we report experiments on all four types of centrality we describe in this paper. Due to the technique that the TWITTER dataset was generated with, almost all paths in that graph are also shortest paths. For this reason we only report results for `#P` and `P` centrality on the TWITTER data. Plots for `#SP` and `SP` look quite similar.

The results show that `G_max` and `G_Sampled` algorithms' performance is comparable to `Greedy` with a CR of at least 70%. Also `G_Sampled` performs better than `G_max` . The performance of the random algorithms strongly depend on the type of centrality and the dataset, but it is always decisively lower, than the performance of all other algorithms.

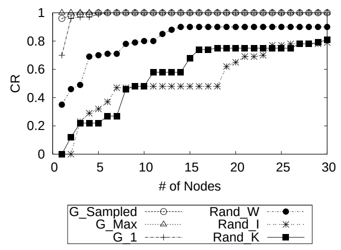
As another observation, we note that `G_1` performs well on finding groups with high `#P` and `P` centralities. However, the same algorithm performs poorly in identifying groups with high `#SP` and `SP` centralities. This shows that the in- and out-degree of a node are not good indicators of the number of special paths this node participates in.

7.2 Evaluating the utility of group centrality.

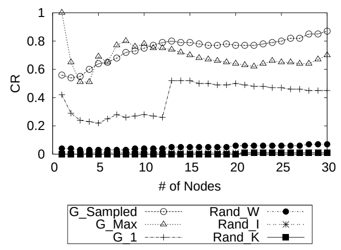
In this set of experiment we aim to show that the centrality of a group is different from the sum of centralities of individual nodes.

For that, we use the QUOTE dataset and the `#SP` and `SP` centralities. We first use the `Greedy` algorithm to solve the k-GCM problem for the two centrality measures and for $k = 1, 2, \ldots 10$. For every value of k we compute the group centrality achieved by the group $A_{\texttt{Greedy}}$ reported by `Greedy`. Then we traverse a list of the nodes - sorted by decreasing individual node centrality - and stop the traversal at position d. This position corresponds to the number of nodes required to cover at least as many special paths as the solution of `Greedy`. The group of traversed nodes is referred to as A_{top}. We report the values of d against k in Table 1.

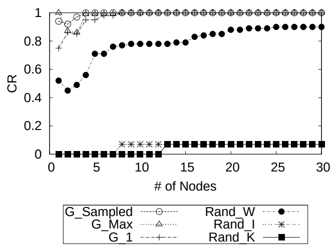
The results in Table 1 show that for relatively small



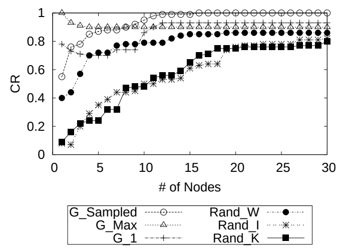
(a) #P; Quote dataset



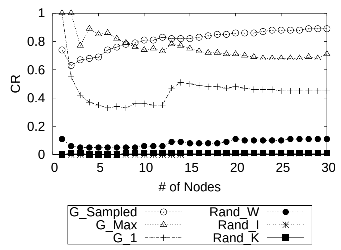
(b) #SP; Quote dataset



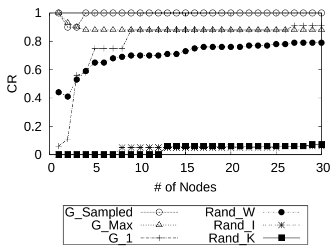
(c) #P; Twitter dataset



(d) P; Quote dataset



(e) SP; Quote dataset



(f) P; Twitter dataset

Figure 3: Group centrality in the k-GCM problem. X axis corresponds to the size k of the group. Y axis reports the *coverage ratio*

sizes of k, d is three times larger than k for comparable group centrality. In case of #P and P centrality we observed even more significant difference. A set of $d = 30$ nodes were needed to match the group centrality of the top 2 nodes chosen by the Greedy algorithm. For lack of space, we do not include the corresponding table.

Table 1: Size d of A_{top} compared to size k of $A_{\texttt{Greedy}}$ for #SP and SP centralities.

k	d (#SP)	d (SP)
1	1	1
2	5	2
3	7	4
4	8	11
5	9	13
6	23	13
7	24	18
8	28	28
9	30	30
10	31	33

7.3 Evaluating the algorithms for k-ECB problem. In this set of experiments, we chose a node $u^* \in V$ at random and applied the different heuristics for the k-ECB problem to increase its centrality. We ran the experiments choosing u^* to have low-, medium- or high-initial centrality. For lack of space we only report the results for #P centrality on the Quote dataset.

We compare Greedy_Add, G_Add_max and G_Add_1 algorithms with the following baselines:

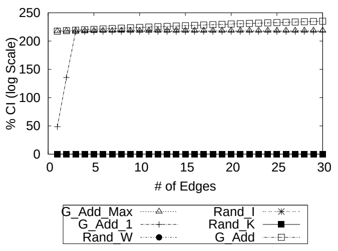
(Rand_K): chooses k nodepairs to add $(u \rightarrow v)$ edges uniformely at random.

(Rand_I): Every edge $(u \rightarrow v)$ is chosen with probability $\frac{k}{\binom{n}{2}}$.

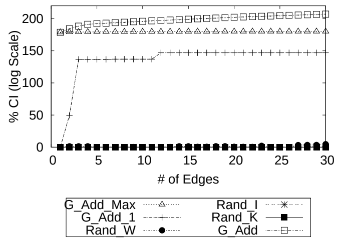
(Rand_W): An edge is created between a node v_1 which is chosen with probability $w(v_1) \times \frac{k}{n}$, and a node v_2 chosen with probability $w(v_2) \times \frac{k}{n}$ such that $\sigma(v_1) < \sigma(v_2)$. The weight of $w(v_1)$ is $= \sum_{u \in C_v} \frac{1}{d_{\text{in}}(u)}$, where $C_v = \{u \in V | (v \rightarrow u) \in E\}$ is the set of children of v_1.

We define the *percentage of increase* (CI) of the centrality of u^* as $C(u^*, G \cup A)/C(u^*, G)$. Where A is the set of edges selected by the algorithms to add to G. In Figure 4 we report the number k of edges added in the x-axis against the CI in the y-axis.

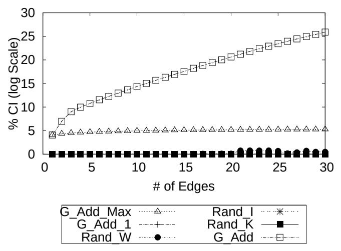
As expected, the baseline algorithms perform poorly in comparison with Greedy_Add and other heuristics. Results on the sampled graph was also poor in comparison and thus are not reported. Greedy_Add achieves the best performance increasing the centrality of nodes by more than 200 times (on a log scale) for low and medium centrality nodes. In addition, we observe



(a) low $C_p(u^*, G)$



(b) medium $C_p(u^*, G)$



(c) high $C_p(u^*, G)$

Figure 4: CI in the k-ECB problem on the Quote dataset, with regard to #P centrality. x-axis corresponds to the number of new edges k, y-axis reports the logarithm of the CI.

that nodes with lower centrality achieve the most benefit from increasing their centrality as opposed to nodes with already high centrality.

8 Conclusions

In this paper, we proposed a unifying definition of centrality of nodes. We showed that this definition subsumes most of the existing notions of centrality, and also allows us to define a generic algorithm for computing nodes' centrality in directed acyclic graphs. Then we defined the k-Group Centrality Maximization and the k-Edge Centrality Boosting problems and showed how our general framework can be used to solve them for any centrality measure. Our experimental results illustrate the usefulness of our framework, but also the efficiency and the efficacy of our algorithms for solving the k-Group Centrality Maximization and the k-Edge Centrality Boosting problems. In the future we plan to explore other types of centrality-management problems. For example, one problem we plan to consider is how to balance the centralities of the nodes within a group, by causing minimal changes to the structure of the input graph.

References

[1] J. M. Anthonisse. *The rush in a directed graph.* SMC, 1971.

[2] S. P. Borgatti. Centrality and network flow. *Social Networks*, 27:55–71, Jan. 2005.

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

[4] U. Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30(2):136 – 145, 2008.

[5] U. Brandes and D. Fleischer. Centrality measures based on current flow. In *STACS'05*, pages 533–544, 2005.

[6] S. Dolev, Y. Elovici, R. Puzis, and P. Zilberman. Incremental deployment of network monitors based on group betweenness centrality. *Inf. Process. Lett.*, 109, September 2009.

[7] M. Everett and S. Borgatti. Extending centrality. *Models and methods in social network analysis*, pages 57–76, 2005.

[8] M. G. Everett and S. P. Borgatti. The centrality of groups and classes. *The Journal of Mathematical Sociology*, 23:181–201, 1999.

[9] M. Fink and J. Spoerhase. Maximum betweenness centrality: approximability and tractable cases. *WALCOM: Algorithms and Computation*, pages 9–20, 2011.

[10] L. C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, Mar. 1977.

[11] K.-I. Goh, B. Kahng, and D. Kim. Universal behavior of load distribution in scale-free networks. *Phys. Rev. Lett.*, Dec 2001.

[12] W. Hwang, Y. rae Cho, A. Zhang, and M. Ramanathan. Bridging centrality: Identifying bridging nodes in scale-free networks. Technical report, 2006.

[13] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a social network or a news media? In *ACM WWW '10*, pages 591–600, New York, NY, USA, 2010.

[14] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *15th ACM SIGKDD*, 2009.

[15] G. Nemhauser, L. Wolsey, and M. Fisher. An analysis of approximations for maximizing submodular set functions. *Math. Prog.*, 14:265–294, 1978.

[16] M. E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27, 2005.

[17] J. J. Pfeiffer III and J. Neville. *Probabilistic Paths and Centrality in Time.* 2010.

[18] R. Puzis. Group betweenness centrality: Efficient computations and applications. 2008.

[19] A. Shimbel. Structural parameters of communication networks. *Bulletin of Mathematical Biology*, 15:501–507, 1953.

Theorem .1. *The* SP *problem is NPcomplete.*

Proof. We proceed by reducing the NP-complete SetCover problem to SP. An instance of SetCover consists of a universe $U = \{u_1, u_2, \cdots, u_m\}$ and a set $S = \{S_1, S_2, \cdots S_n\}$, where $\forall i, S_i \subseteq U$ is a subset of U and k is an integer. The goal is to find a subset $S' \subseteq S$ such that $|S'| \leq k$ and $\{u_j \in U : u_j \in \cup_{S_i \in S'} S_i\}$. Define an instance of SP by constructing a directed graph $G' = (V', E')$ as follows. Designate the nodes $u_i \in U$ as nodes $u_i \in V'$ and the sets $S_i \in S$ as nodes $s_i \in V'$. Thus, $V' = \{u_i\} \cup \{s_i\}$. Let E' consists of a set of edges e defined as follows. Create a directed edge between each node $n_j \in V$ and an $s_i \in V$ iff $n_j \in S_i$. Furthermore, create a additional directed edge from $s_i \in V$ to each node $n_j \in V$ iff $n_j \in S_i$. Observe that a directed shortest path between a node to itself, i.e a node pair (n_j,n_j) is of length two and will necessarily pass through the node s_i.

Let us assume that A' is a solution of size k to the SP problem. We will show that $S' \subseteq S$ is a solution for SetCover problem. Observe that A' will cover at least one shortest path between a node pair (n_j,n_j) thus conclude that an S' corresponding to the nodes of A' is a solution to the SetCover problem. Removing a node from A' will result that there will be no shotest path coverage for at least one node pair (n_j,n_j), which in turn will result in an incomplete solution to the SetCover problem. Since the decision problem of the SetCover is NP-Complete, this reduction shows that SP is also NP-Complete.

Theorem .2. *The #SP problem is NPcomplete.*

Proof. We proceed by reducing the NP-complete VertexCover problem to #SP. We say that for an undirected graph $G(V, E)$, a set $A \subseteq V$ is a vertex cover of G, if every edge in E is incident to at least one node in A. For an instance of the VertexCover problem, let $G(V, E)$ be an undirected graph and k an integer. The decision version of the problem asks for a set $A \subseteq V$ of size k that is a VertexCover.

Define the directed graph $G'(V', E')$ of the corresponding #SP problem as follows. Let $V' = V \cup \{s_i, d_i\}$ contain the nodes in G, and additional source s_i and destination d_i nodes where $i = 1..|E|$. Let E' contain all edges in E. In addition to that, add an edge from a source s_i to a node v such that v is one of the node incident to edge e_i. Also add an edge from the other node incident to edge e_i to d_i. Observe that a shortest path between node pairs (s_i,d_i) $i = 1..|E|$ will traverse the edge e_i and would be of length at least three. The total number of Nodes and edges in G' is $n + 2E$ and $3E$ respectively.

Let us assume that A' is a solution of size k to the #SP problem. We will show that $A \subseteq V$ is a solution for VertexCover problem. A' will cover all shortest paths between pairs (s_i,d_i). But covering a shortest path between (s_i,d_i) would necessarily imply covering one of the nodes incident to edge e_i. Since the decision problem of the VertexCover is NP-Complete, this reduction shows that #SP is also NP-Complete.

Theorem .3. *The #P problem is NP-complete.*

Proof. We reduce the NP-complete VertexCover problem to the #P problem on DAGs. We say that for an undirected graph $G(V, E)$, a set $A \subseteq V$ is a *vertex cover* of G, if every edge in E is incident to at least one node in A. For an instance of the VertexCover problem, let $G(V, E)$ be an undirected graph and k an integer. The decision version of the problem asks for a set $A \subseteq V$ of size k that is a vertex cover.

Define the DAG $G'(V', E')$ of the corresponding #P problem as follows. Let $V' = V \cup \{s, t\}$ contain the nodes in G, an additional source node s and an additional sink t. Let E' contain all edges in E. In addition to that, add an edge from the source to every node, and from every node to the sink. Fix an arbitrary order σ of the nodes in V', such that s is the first and t is the last in this ordering. Then direct every edge $(u \to v) \in E'$ from u to v if $\sigma(u) < \sigma(v)$, otherwise from v to u. This will naturally result in a DAG. Let m be an arbitrary integer such that $m > \Omega(|V'|^5)$. We will replace every directed edge in E' (including the edges incident to s and t) with the following *multiplier* tool (Figure 5). For every edge $(u \to v)$ we add m new nodes: $w_1, w_2, \ldots, w_m$, and $2m$ new directed edges: (u, w_i) and (w_i, v). Observe, that by this exchange, the size of the graph only changes by a polynomial factor of the original size. Let $P_{V'}$ be the total number of directed paths in V'. The instance of #P consists of the graph $G'(V', E')$ and of the pair (s, t). Which means ,the objective of #P is to cover a maximal number of paths from s to t. Now we proof that there exists a vertex cover A of size at most k for this instance of the VertexCover problem if and only if there exists an #P A' of size k where $U(A')$ (the number of uncovered paths) is $U(A') = P_{V'} - F(A') < \Omega(m^3)$. Thus, the number of uncovered paths is at most of order $O(m^2)$. In addition we claim that $A' \subseteq V$ and thus $A = A'$ is the desired solution for the VertexCover.

Let $A' \subseteq V'$ be a solution of size k for the #P problem. First we show, that if $k \leq n$, then $A' \subseteq V$. Let us assume that node w has indegree 1, and his parent is v. Observe now, that every path that goes through w is also covered by v, moreover v might have other children besides w. Hence at least as many paths are covered if $v \in A'$, then if $w \in A'$. For this reason we can assume that a node with indegree 1 is only in A' if all other nodes with larger indegree are already in the set. Since all new nodes w_i have indegree 1, this implies $A' \subseteq V$.

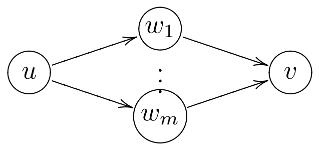


Figure 5: "Multiplier edge" construction for G'. When x items leave u, $x \cdot m$ items arrive at v.

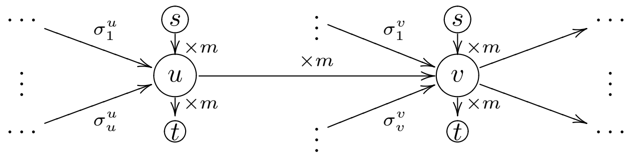


Figure 6: Isolated subgraph $G_{uv} = \{v, v, s, t\}$ of G'.

Now we show that $A' = A$ is a vertex cover. We show that (1.) if for every edge at least one of its ends is incident to A, then $U(A) = O(m^2)$, and (2.) if there is an edge $(u \rightarrow v) \in E'$, such that $u, v \notin A$, then $U(A) = \Omega(m^3)$. For this, let us consider the subgraph G_{uv} depicted in Figure 6, corresponding to the nodes $u, v \in V$ and the adjacent edges. σ_i depicts the number of incoming uncovered paths on that edge. Let $\Sigma_u = \sigma_1^u + \sigma_2^u + \ldots + \sigma_u^u + m$ and $\Sigma_v = \sigma_1^v + \sigma_2^v + \ldots + \sigma_v{}^v + m$ be the total number of incoming uncovered paths through u and v. Let us assume that every edge $(u', v') \in E$ different from $(u \rightarrow v)$ is incident to A. For the edge $(u \rightarrow v) \in E$ we have to check four cases (in order to compute $U(A)$ we have to consider, how many uncovered paths go to node t, through all the nodes in V):

case $u, v \in A$**:** $U(A) = 0 * \Sigma_u + 0 * \Sigma_v + O(nm) = O(nm) << \Omega(m^3)$
case $u \in A, v \notin A$**:** $U(A) = 0 * \Sigma_u + m * \Sigma_v + O(nm) = O(m + m^2) = O(m^2) < \Omega(m^3)$
case $u \notin A, v \in A$**:** $U(A) = m * \Sigma_u + 0 * \Sigma_v + O(nm) = O(m^2 + m) = O(m^2) < \Omega(m^3)$
case $u, v \notin A$**:** $U(A) = m * \Sigma_u + m * \Sigma_u * \Sigma_v + O(nm) = O(m^2 + m^3 + m) = O(m^3)$

The above cases show our claim, that A is a vertex cover if and only if $U(A) < \Omega(m^3)$.