Greedily Improving Our Own Closeness Centrality in a Network

Pierluigi Crescenzi, Gianlorenzo d'Angelo, Lorenzo Severini, Yllka Velaj

▶ **To cite this version:**

Pierluigi Crescenzi, Gianlorenzo d'Angelo, Lorenzo Severini, Yllka Velaj. Greedily Improving Our Own Closeness Centrality in a Network. ACM Transactions on Knowledge Discovery from Data (TKDD), 2016, 11 (1), pp.1-32. 10.1145/2953882 . hal-01390134

HAL Id: hal-01390134

https://inria.hal.science/hal-01390134

Submitted on 29 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Copyright

Greedily Improving Our Own Closeness Centrality in a Network

PIERLUIGI CRESCENZI, University of Florence
GIANLORENZO D'ANGELO, LORENZO SEVERINI, and YLLKA VELAJ,
Gran Sasso Science Institute

The closeness centrality is a well-known measure of importance of a vertex within a given complex network. Having high closeness centrality can have positive impact on the vertex itself: hence, in this paper we consider the optimization problem of determining how much a vertex can increase its centrality by creating a limited amount of new edges incident to it. We will consider both the undirected and the directed graph cases. In both cases, we first prove that the optimization problem does not admit a polynomial-time approximation scheme (unless $P = NP$), and then propose a greedy approximation algorithm (with an almost tight approximation ratio), whose performance is then tested on synthetic graphs and real-world networks.

General Terms: Closeness Centrality, Collaboration Networks, Citation Networks

Additional Key Words and Phrases: Approximation algorithms, graph augmentation, greedy algorithm, large networks

1. INTRODUCTION

Looking for the most important vertices within a given complex network has always been one of the main goals in the field of real-world network analysis. Different measures of importance have been introduced in the literature, and several of them are related to the notion of "centrality" of a vertex. This latter notion, in turn, has been explicitly formalized in different ways: one of the most popular is the closeness centrality measure (see, for example, Boldi and Vigna 2014). This measure somehow evaluates the efficiency of a vertex while spreading information to all other vertices in its connected component: more formally, the closeness centrality of u is equal to the sum of the reciprocal of the distances to u from all other vertices. Computing closeness centrality, however, is too time expensive, since it requires to run a breadth first search (BFS)

Preliminary results about this work have been presented in the 14th International Symposium on Experimental Algorithms (SEA) [Crescenzi et al. 2015].
Authors' addresses: G. D'Angelo, L. Severini, and Y. Velaj, Gran Sasso Science Institute (GSSI), Viale F. Crispi, 7, I-67100 L'Aquila, Italy; emails: {gianlorenzo.dangelo, lorenzo.severini, yllka.velaj}@gssi.infn.it; P. Crescenzi, Department of Information Engineering, University of Florence, Viale Morgagni, 65, I-50134 Florence, Italy; email: pierluigi.crescenzi@unifi.it.

for each vertex, which is clearly infeasible for networks with millions of vertices and edges (which is the "normal" size of many interesting real-world networks). For this reason, several randomized and/or approximation algorithms have been proposed for the computation of this centrality measure [Cohen et al. 2014].

In this paper, instead, we consider a different problem related to the closeness centrality, that is, the problem of identifying which "strategy" a vertex should adopt in order to increase its own centrality value. Indeed, increasing its own ranking in terms of centrality can have positive consequences for the vertex. For example, in the field of author citation networks closeness centrality seems to be significantly correlated with citation counts (as it has been already observed in the case of collaboration networks) [Yan and Ding 2009]. We then consider the optimization problem of efficiently determining, for a given vertex u, the set of k edges incident to u that, when added to the original graph, allows u to increase as much as possible its closeness centrality and its ranking according to this measure. We will analyze both the undirected and the directed graph cases. We first prove that this problem is hard to be approximated within an approximation factor greater than $1 - \frac{1}{15e}$ in the undirected case (respectively, $1 - \frac{1}{3e}$ in the directed case), and then show that a greedy approach yields a $(1 - \frac{1}{e})$-approximation algorithm in both undirected and directed cases. Successively, we present several experiments that we have performed (i) in order to evaluate how good is the approximation factor in the case of relatively small randomly generated graphs, (ii) in order to apply the greedy approach to real-world collaboration, citation and transportation networks, and (iii) in order to evaluate the actual improvement in information spread. As a result of the first set of experiments, we have that the greedy algorithm seems to perform much better than the theoretical results, since it often computes an optimal solution and, in any case, it achieves an approximation factor significantly larger than the theoretical one. By applying the greedy algorithm to real-world networks, instead, we observe that by adding a very few edges a vertex can drastically increase its centrality measure and, hence, its ranking. We note that after adding a limited number of edges, the number of informed nodes in the network highly increases.

The problem of adding edges to a graph in order to modify some general properties has been widely studied. To the best of our knowledge, the problems that aim at optimizing some property by adding a limited number of edges are: minimizing the average shortest-path distance between all pair of nodes [Meyerson and Tagiku 2009; Papagelis et al. 2011; Parotsidis et al. 2015], minimizing the average number of hops in shortest paths of weighted graphs [Bauer et al. 2012], maximizing the leading eigenvalue of the adjacency matrix [Saha et al. 2015; Tong et al. 2012], minimizing the diameter [Bilò et al. 2012; Frati et al. 2015], maximizing or minimizing the number of triangles [Dehghani et al. 2015; Li and Yu 2015], minimizing the eccentricity [Perumal et al. 2013], and minimizing the characteristic path length [Papagelis 2015].

The problem analyzed in this paper differs from above mentioned ones as it focuses on improving the centrality of a predefined vertex. As far as we know, our problem has never been attacked before, even though similar problems have been studied for other centrality measures, i.e., page-rank [Avrachenkov and Litvak 2006; Olsen and Viglas 2014], eccentricity [Demaine and Zadimoghaddam 2010], average distance [Meyerson and Tagiku 2009], some measures related to the number of paths passing through a given node [Ishakian et al. 2012], and betweenness centrality [Crescenzi et al. 2015; D'Angelo et al. 2016]. Hence, we had no other algorithms to compare with. However, we also consider other potential alternative algorithms and show that the greedy algorithm significantly outperforms them, whenever $k > 1$.

1.1. Motivating Applications

In this section we motivate our study by showing two applications in which improving the centrality of a specific node by adding edges incident to it can give benefits to the node itself or to the whole network.

1.1.1. Increasing the Spreading of Information. Intuitively closeness centrality evaluates the efficiency of a vertex while spreading information to all other vertices in its connected component. We show that solving our problem for a set of given vertices has positive consequences for the spreading of information through the network. To this aim, we consider the *Linear Threshold Model* which is a widely studied model in network analysis to represent the spread of information [Kempe et al. 2015]. In this model, we can distinguish between *active nodes* (which spread the information) and inactive ones. The idea is that a node becomes active if a large part of its neighbors are active. In detail, each node u has a threshold a chosen uniformly at random in the interval $[0, 1]$. The threshold represents the fraction of neighbors of u that must become active in order for u to become active. At the beginning of the process a small percentage of nodes of the graph is set to active to let the information diffusion process start, these nodes are called *seeds*. In subsequent steps of the process, a node becomes active if the fraction of its active neighbors is greater than its threshold.

In our experimental study, we show that adding a small number of edges incident to some randomly-chosen seeds highly increases the spreading of information in terms of number of nodes that become active. Note that this represents an improvement of the whole network in terms of the efficiency of propagating information. We performed such experiments on both undirected and directed networks.

1.1.2. Link Recommendation. The link recommendation task consists in suggesting potential connections to social network users with the aim of increasing their social circle. Link recommendations improve the user experience and at the same time help to increase the connectivity inside the network and speed-up the network growth. Most of the existing link recommendation methods focus on estimating the likelihood that a link is adopted by users and recommend links that are likely to be established [Backstrom and Leskovec 2011; Liben-Nowell and Kleinberg 2003; Popescul and Ungar 2003; Yin et al. 2010].

Recently, a new approach has been proposed whose aim is to recommend a set of links that, when added to the network, increase the centrality of a user in a network. In particular, suggesting links that minimize the expected average distance of a node accurately predicts the links that will actually appear in the graph [Parotsidis et al. 2016]. An important step in this approach is to determine the set of links that, when added to the network, maximize the specific centrality measure considered.[1]

Of particular interest in this context are the collaboration networks in which nodes represent users and links represent collaboration between users (e.g., authors collaborating in the same papers or actors that acted in the same movie). The link recommendation problem in such a case consists in suggesting possible persons to whom request for future collaboration. In our experiments we show that we are able to compute a set of nodes that highly increase the closeness centrality in very large collaboration networks such as those induced by the DBLP and IMDB databases [Ley; IMDB].

1.2. Structure of the Paper

The rest of the paper will be divided into two parts: the first part is devoted to the analysis of the undirected graph case, while the second one is devoted to the directed

[1]Note that the centrality measure used in Parotsidis et al. [2016] is the inverse of the arithmetic mean of the distances to a node, while in this paper we consider the harmonic mean of the distances to a node.

graph case. Each part is in turn split into a theoretical part and experimental part. In particular, in both parts, we first define the optimization problem (see Sections 2.1 and 3.1), then prove the non-approximability result (see Sections 2.2 and 3.2), and finally introduce and analyze the greedy approximation algorithm (see Sections 2.3 and 3.3). In the case of undirected graphs, we also show how the running time of this algorithm can be improved by making use of dynamic algorithm techniques (see Section 2.4). In the experimental sections of the two parts, instead, we first compare the greedy solution with the optimal one, then compare the performances of the greedy algorithm with the ones of several alternative baseline strategies, and lastly analyze the impact of adding edges on the performances of the information spreading process (see Sections 2.5 and 3.4). Finally, we measure the improvement in the value of the closeness of a node and in its closeness ranking within two large real-world undirected networks (see Section 2.6) and one large real-world directed network (see Section 3.4).

2. THE UNDIRECTED GRAPH CASE

In this section we will focus on undirected graphs. After giving all necessary definitions and preliminary results, we will introduce the optimization problem that will be considered, prove a non-approximability result, and then describe an approximation algorithm. Finally, we will present the experiments that we have performed in order to validate this algorithm and to apply it to two quite big collaboration networks.

2.1. The Maximum Closeness Improvement Problem

Let $G = (V, E)$ be an undirected graph, where V denotes the set of nodes, and E denotes the set of edges $\{u, v\}$ with $u, v \in V$. For each node u, N_u denotes the set of neighbors of u, i.e., $N_u = \{v \mid \{u, v\} \in E\}$. Given two vertices u and v, we denote by d_{uv} the distance from u to v in G, that is, the number of edges in a shortest path from u to v (if there is no path from u to v, we then set $d_{uv} = \infty$). For each node u, the *closeness centrality* (also called *harmonic centrality* [Boldi and Vigna 2014]) of u is defined as follows:

$$c_u = \sum_{\substack{v \in V \\ d_{uv} < \infty}} \frac{1}{d_{uv}}.$$

Given a set S of edges not in E, we denote by $G(S)$ the graph augmented by adding the edges in S to G, i.e., $G(S) = (V, E \cup S)$. For a parameter x of G, we denote by $x(S)$ the same parameter computed in the augmented graph $G(S)$ (for example, the distance from u to v in $G(S)$ is denoted as $d_{uv}(S)$).

The closeness centrality of a vertex clearly depends on the graph structure: if we augment a graph by adding a set of edges S, then the centrality of a vertex might change. Generally speaking, adding edges incident to some vertex u can only increase the centrality of u. Given a graph $G = (V, E)$, a vertex $u \in V$, and an integer k, the *Maximum Closeness Improvement* (MCI) problem consists in finding a set S of edges incident to u not in E (that is, $S \subseteq \{\{u, v\} : v \in V \setminus N_u\}$) such that $|S| \leq k$ and $c_u(S)$ is maximum.

2.2. The Non-Approximability Result

In this section, in order to derive our approximation hardness result for the MCI problem, we will make use of the *Minimum Dominating Set* (MDS) problem, which is defined as follows: given an undirected graph $G = (V, E)$, find a *dominating set* of minimum cardinality, that is, a subset D of V such that $V = D \cup \bigcup_{u \in D} N_u$. It is known that for any r with $0 < r < 1$, it cannot exist a $(r \ln |V|)$-approximation algorithm for the MDS problem, unless $P = NP$ [Dinur and Steurer 2014]. We will now use this result in order to show that the MCI problem does not admit a polynomial-time

Algorithm: A'
Input : an undirected graph $G = (V, E)$ and an integer k
Output: a dominating set D

```
1 D := ∅;
2 while V ≠ ∅ do
3     Compute graph G' starting from G (see Fig. 2);
4     S := A(G', z, k);
5     D' := {u : {z, u} ∈ S}
6     D := D ∪ D';
7     V := V − D' − ⋃_{u∈D'} N_u;
8     G := subgraph of G induced by V;
9 return D;
```

Fig. 1. The approximation algorithm for the MDS problem, given a γ-approximation algorithm A for the MCI problem and a "guess" k for the optimal value of MDS.

approximation scheme. To this aim, we will design an algorithm A' that, given an undirected graph $G = (V, E)$ and given the size k of the optimal dominating set of G, by using an approximation algorithm A for the MCI problem will return a dominating set of G whose approximation ratio is at most $(r \ln |V|)$. Clearly, we do not know the value of k, but we know that this value must be at least 1 and at most $|V|$: hence, we run algorithm A' for each possible value of k, and return the smallest dominating set found. Algorithm A' will run the approximation algorithm A for the MCI problem multiple times. Each time A will find k nodes $u \in V$ which are the "new" neighbors of the node whose centrality has to be increased: we then add these nodes to the dominating set and create a smaller instance of the MCI problem (which will contain, among the others, all the nodes in V not yet dominated). We continue until all nodes in V are dominated.

THEOREM 2.1. *For each $\gamma > 1 - \frac{1}{15e}$, there is no γ-approximation algorithm for the MCI problem, unless $P = NP$.*

PROOF. We will show that a γ-approximation algorithm A for the MCI problem, with $\gamma > 1 - \frac{1}{15e}$, would imply a $(r \ln n)$-approximation algorithm A' for the MDS problem, thus, proving the theorem. In particular, the algorithm A' is specified in Figure 1, where k denotes a "guess" of the size of an optimal solution for MDS with input the graph G. In the following, ω will denote the number of times the *while* loop is executed. Since, at each iteration of the loop, we include in the dominating set at most k nodes, at the end of the execution of algorithm A' the set D includes at most $k \cdot \omega$ nodes. Hence, if k is the correct guess of the value of the optimal solution for the MDS instance, then D is a ω-approximate solution for the MDS problem (as we have already noticed, we don't know the correct value of k, but algorithm A' can be executed for any possible value of k, that is, for each $k \in [|V|]$).

The first instruction of the *while* loop of algorithm A' computes a transformed graph G' (to be used as part of the new instance for MCI) starting from the current graph $G = (V, E_V)$, which is the subgraph of the original graph induced by the set $\{u_1, \ldots, u_n\}$, where $n = |V|$, of still not dominated nodes. This computation is done as follows (see Figure 2). We add a new node z and two new nodes x_i and y_i, for each i with $1 \leq i \leq n$. Moreover, we add to E_V the edges $\{z, y_i\}$, $\{x_i, y_i\}$, and $\{x_i, u_i\}$, for each i with $1 \leq i \leq n$. As it is shown in the second line of the *while* loop, z is the node whose centrality c_z has to be increased by adding at most k edges: that is, the MCI instance is formed by G', z, and k. Observe that any solution for this instance that contains an edge $\{x_i, z\}$ can be modified, without decreasing its measure, by substituting this edge with $\{u_i, z\}$:

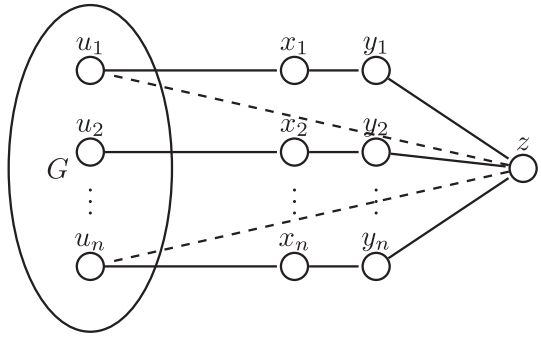


Fig. 2. The reduction used in Theorem 2.1. The dashed edges between node z and nodes u_i denote those added in a solution to MCI.

hence, we can assume that the solution S computed at the second line of the *while* loop of algorithm A' contains only edges connecting z to nodes in V (which are shown by dashed edges between node z and nodes u_i in Figure 2).

First of all, note that, since k is (a guess of) the measure of an optimal solution D^* for MDS with input G, we have that the measure $c^*(G', z, k)$ of an optimal solution S^* for MCI with input G' satisfies the following inequality:

$$c^*(G', z, k) \geq k + \frac{1}{2}(n-k) + \frac{3}{2}n = \frac{1}{2}k + 2n.$$

This is due to the fact that by connecting z to all the k nodes in D^*, in the worst case we have that k nodes in G are at distance 1, $n-k$ nodes in G are at distance 2 (since D^* is a dominating set), the n nodes y_i are at distance 1, and the n nodes x_i are at distance 2 from z.

Given the solution S computed by the approximation algorithm A for MCI, let a and b denote the number of nodes in G at distance 2 and 3, respectively, from z in $G'(S)$. Since all nodes in G' are at distance at most 3 from z, we have that $n = k + a + b$ (we can assume, without loss of generality, that $n \geq k$): hence, $a = n - b - k$. Since A is a γ-approximation algorithm for MCI, we have that $c_z(S) \geq \gamma c^*(G', z, k)$. That is,

$$k + \frac{1}{2}a + \frac{1}{3}b + \frac{3}{2}n \geq \gamma\left(\frac{1}{2}k + 2n\right).$$

From this inequality, it follows that:

$$a \geq \gamma(k + 4n) - 3n - 2k - \frac{2}{3}b.$$

By using the fact that $a = n - b - k$, we have that

$$n - b - k \geq \gamma(k + 4n) - 3n - 2k - \frac{2}{3}b.$$

That is,

$$b \leq 12(1-\gamma)n + 3(1-\gamma)k.$$

Since $k \leq n$, we then have that

$$b \leq 15n(1-\gamma).$$

Assuming $\gamma > 1 - \frac{1}{15e} > \frac{14}{15}$ (which implies $15(1-\gamma) < 1$), then after one iteration of the *while* loop of algorithm A', the number of nodes in G decreases by a factor $15(1-\gamma)$.

Algorithm: GREEDYIMPROVEMENT
Input : an undirected graph $G = (V, E)$; a vertex $u \in V$; and an integer $k \in \mathbb{N}$
Output: set of edges $S \subseteq \{\{u, v\} \mid v \in V \setminus N_u\}$ such that $|S| \leq k$

```
S := ∅;
for i = 1, 2, ..., k do
    foreach v ∈ V \ N_u(S) do
        Compute c_u(S ∪ {u, v});
    v_max := arg max{c_u(S ∪ {u, v}) | v ∈ V \ N_u(S)};
    S := S ∪ {{u, v_max}};
return S;
```

Fig. 3. The greedy algorithm for undirected graphs.

Hence, after $\omega - 1$ iterations, the number n of nodes in the graph G is at most a fraction $[15(1-\gamma)]^{\omega-1}$ of the number N of nodes in the original graph. Since we can stop as soon as $n < k$, we need to find the maximum value of ω such that $k \leq N[15(1-\gamma)]^{\omega-1}$. By solving this inequality and by recalling that $15(1-\gamma) < 1$, we obtain

$$\omega - 1 \leq \log_{15(1-\gamma)} \frac{k}{N} \leq \log_{15(1-\gamma)} \frac{1}{N} = \frac{\ln(N)}{\ln \frac{1}{15(1-\gamma)}}.$$

One more iteration might be necessary to trivially deal with the remaining nodes, which are less than k. Hence, the total number ω of iterations is at most $\frac{\ln(N)}{\ln \frac{1}{15(1-\gamma)}} + 1$. If $\gamma > 1 - \frac{1}{15e}$, we have that $r' = \frac{1}{\ln \frac{1}{15(1-\gamma)}} < 1$: as a consequence of the observation at the beginning of the proof, the solution reported by algorithm A' is an $(r' \ln N + 1)$-approximate solution. Clearly, for any r with $0 < r' < r < 1$, there exists N_r sufficiently large, such that for any $N > N_r$, $r' \ln N + 1 \leq r \ln N$: hence, algorithm A' would be an $r \ln N$-approximation algorithm for MDS, and, because of the result of Dinur and Steurer [2014], P would be equal to NP. Thus, we have that, if $P \neq NP$, then γ has to be not greater than $1 - \frac{1}{15e}$ and the theorem is proved. □

2.3. The Greedy Approximation Algorithm

Let us consider the following optimization problem. Given a set X and an integer k, find a subset Y of X of cardinality at most k that maximizes the value $f(Y)$, where $f : 2^X \to \mathbb{N}$ is a specific *objective function*. If f is *monotone submodular*, that is, if, for any pair of sets $S \subseteq T \subseteq X$ and for any element $e \in X \setminus T$, $f(S \cup \{e\}) - f(S) \geq f(T \cup \{e\}) - f(T)$, then the following greedy algorithm approximates the above problem within a factor $1 - \frac{1}{e}$ [Nemhauser et al. 1978]: start with the empty set, and repeatedly add an element that gives the maximal marginal gain. In this section, we exploit this result by showing that c_u is monotone and submodular with respect to the possible set of edges incident to u. Hence, the greedy algorithm GREEDYIMPROVEMENT (reported in Figure 3) provides a $(1 - \frac{1}{e})$-approximation. Note that the computational complexity of such algorithm is $O(k \cdot n \cdot g(n, m+k))$, where $g(n, m+k)$ is the complexity of computing c_u in a graph with n nodes and $m + k$ edges.

THEOREM 2.2. *For each vertex u, function c_u is monotone and submodular with respect to any feasible solution for MCI.*

PROOF. To simplify the notation, in the following we will assume that $\frac{1}{\infty} = 0$. To show that c_u is monotone increasing, it is enough to observe that for each solution S to

MCI, for each edge $\{u, v\} \notin E \cup S$, and for each node $x \in V \setminus \{u\}$, $d_{ux}(S \cup \{\{u, v\}\}) \leq d_{ux}(S)$ (since adding an edge cannot increase the distance between two nodes) and, therefore, $\frac{1}{d_{ux}(S \cup \{\{u,v\}\})} \geq \frac{1}{d_{ux}(S)}$. We now show that for each pair S and T of solutions for MCI such that $S \subseteq T$, and for each edge $\{u, v\} \notin T \cup E$,

$$c_u(S \cup \{\{u, v\}\}) - c_u(S) \geq c_u(T \cup \{\{u, v\}\}) - c_u(T).$$

To this aim, we prove that each term of c_u is submodular, i.e., that for each vertex $x \in V \setminus \{u\}$,

$$\frac{1}{d_{ux}(S \cup \{\{u, v\}\})} - \frac{1}{d_{ux}(S)} \geq \frac{1}{d_{ux}(T \cup \{\{u, v\}\})} - \frac{1}{d_{ux}(T)}. \tag{1}$$

Let us consider the shortest paths from u to x in $G(T \cup \{\{u, v\}\})$, and let us distinguish the following two cases:

(1) The first edge of a shortest path from u to x in $G(T \cup \{\{u, v\}\})$ is $\{u, v\}$ or belongs to $S \cup E$. In this case, such a path is a shortest path also in $G(S \cup \{\{u, v\}\})$, as it cannot contain edges in $T \setminus S$ (since these edges are all incident to u). Then, $d_{ux}(S \cup \{\{u, v\}\}) = d_{ux}(T \cup \{\{u, v\}\})$ and $\frac{1}{d_{ux}(S \cup \{\{u,v\}\})} = \frac{1}{d_{ux}(T \cup \{\{u,v\}\})}$. Moreover, $d_{ux}(S) \geq d_{ux}(T)$ (since $S \subseteq T$) and, therefore, $-\frac{1}{d_{ux}(S)} \geq -\frac{1}{d_{ux}(T)}$.

(2) The first edge of all shortest paths from u to x in $G(T \cup \{\{u, v\}\})$ belongs to $T \setminus S$. In this case, $d_{ux}(T) = d_{ux}(T \cup \{\{u, v\}\})$ and, therefore, $\frac{1}{d_{ux}(T \cup \{\{u,v\}\})} - \frac{1}{d_{ux}(T)} = 0$. As $\frac{1}{d_{ux}(S)}$ is monotone increasing, then $\frac{1}{d_{ux}(S \cup \{\{u,v\}\})} - \frac{1}{d_{ux}(S)} \geq 0$.

In both cases, we have that inequality (1) is satisfied and, hence, the theorem follows. □

COROLLARY 2.3. *The MCI problem is approximable within a factor* $(1 - \frac{1}{e})$.

As it can be seen, there is quite a significant gap between the non-approximability result proved in Theorem 2.1 (that is, the upper bound equal to $1 - \frac{1}{15e} \approx 0.98$), and the approximability result of the above corollary (that is, the lower bound $(1 - \frac{1}{e}) \approx 0.63$). One of the main goals of the next experimental session is to analyze the "real" performance, in terms of solution quality, of the greedy algorithm on relatively small real-world and synthetic graphs.

2.4. Improving the Greedy Algorithm Running Time

In this section we show how to improve the running time of GREEDYIMPROVEMENT. This algorithm requires $O(k \cdot n \cdot g(n, m + k))$ computational time, where $g(n, m + k)$ is the complexity of computing c_u in a graph with n nodes and $m + k$ edges. The classical algorithm to compute c_u consists in determining all the distances to u by running a BFS starting from u. Therefore, with such an approach, GREEDYIMPROVEMENT requires $O(k \cdot n \cdot (n + m + k))$ in the worst case. In this section we provide a dynamic algorithm to reduce the time required to compute c_u.[2] Furthermore, we show how to exploit the submodularity of c_u in order to reduce the running time of iterations $i \geq 2$ of the *for* loop at line 2 of GREEDYIMPROVEMENT.

[2]Note that the idea of incrementally updating the closeness centrality has been already explored in the literature [Kas et al. 2013; Sariyüce et al. 2013]. However, in this paper we consider the harmonic mean to compute the closeness centrality instead of the arithmetic mean that is used in other papers. The motivation is that the harmonic mean has been shown to be more robust in the case of undirected disconnected networks or directed not-strongly connected networks [Boldi and Vigna 2014]. Therefore, we cannot directly use the algorithms in the literature and we devise a new dynamic algorithm.

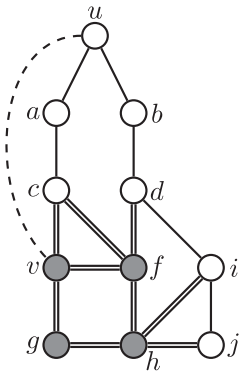
Algorithm: *DynamicBFS*
Input : An undirected graph $G(S)$; edge $\{u, v\}$; distances $d_{ux}(S)$, for each $x \in V$
Output: ΔClo, the increment to c_u obtained when adding edge $\{u, v\}$ to $G(S)$

```
1  Q := ∅;
2  visited := ∅;
3  d_uv(S ∪ {u, v}) := 1;
4  foreach x ∈ N_v(S) do
5      d_ux(S ∪ {u, v}) := min{2, d_ux(S)};
6  visited := {u, v} ∪ N_u(S);
7  ΔClo := 1 − 1/d_uv(S);
8  foreach x ∈ N_v(S) do
9      Q.push(x);
10 while ¬Q.empty() do
11     x := Q.pop();
12     ΔClo := ΔClo + 1/d_ux(S∪{u,v})(x) − 1/d_ux(S);
13     foreach y ∈ N_x(S) do
14         if (y ∉ visited) ∧ (d_uy(S) > d_ux(S ∪ {u, v}) + 1) then
15             d_uy(S ∪ {u, v}) := d_ux(S ∪ {u, v}) + 1;
16             Q.push(y);
17             visited := visited ∪ {y};
18 return ΔClo
```

Fig. 4. Algorithm *DynamicBFS*.

Let us assume that we add an edge $\{u, v\} \notin E \cup S$ to graph $G(S)$. The dynamic algorithm aims at computing only the distances between u and any other node that change as a consequence of the addition of edge $\{u, v\}$ (i.e., nodes w such that $d_{uw}(S) \neq d_{uw}(S \cup \{u, v\})$) and keep the old distances to any other node in the graph. The algorithm is based on the following observation: if we add an edge $\{u, v\}$ to $G(S)$, then $d_{uw}(S) \neq d_{uw}(S \cup \{u, v\})$, for some $w \in V$, only if the shortest path between u and w in $G(S \cup \{u, v\})$ contains edge $\{u, v\}$. Therefore, we can determine the nodes that change their distance to u by finding all the shortest paths passing through edge $\{u, v\}$ in $G(S \cup \{u, v\})$. To this aim, the dynamic algorithm executes a BFS starting from node v and prunes the search as soon as a node that does not change its distance to u is extracted from the queue. We report the dynamic algorithm *DynamicBFS* in Figure 4. In detail, Procedure *DynamicBFS* returns the value ΔClo which corresponds to the increment to $c_u(S)$ which is obtained by adding edge $\{u, v\}$. To compute ΔClo, the algorithm computes the distances between u and any node y such that $d_{uy}(S) \neq d_{uy}(S \cup \{u, v\})$. First, it computes the distances of u and its neighbors (lines 3–5) and the initial increment ΔClo that is equal to the difference between the reciprocal of the new distance and that of the old distance (line 7). Then, it pushes in queue Q the neighbors of u (lines 8 and 9) and performs the BFS starting from v (lines 10–17). For each extracted node, it updates ΔClo by subtracting the reciprocal of the old distance and adding the new one (line 12). After that, it enqueues a neighbor y of the extracted node x only if the old distance $d_{uy}(S)$ is greater than the length of the path made of the shortest path from u to x in $G(S \cup \{u, v\})$ and the edge $\{x, y\}$ (that is $d_{ux}(S \cup \{u, v\}) + 1$, see the test at line 14). Note that this condition is satisfied only if the shortest path between u and y passes through edge $\{u, v\}$. The procedure repeats this process until the queue is empty.

We give an example of execution of Algorithm *DynamicBFS* in Figure 5.



(a) Graph G, the dashed edge is the newly added edge $\{u, v\}$, gray nodes and double edges are visited by Algorithm $DynamicBFS$.

Iter.	**Node extracted from Q**	ΔClo	Q
0	v	2/3	(c, f, g)
1	c	2/3	(f, g)
2	f	5/6	(g, h, d)
3	g	13/12	(h, d)
4	h	7/6	(d, i, j)
5	d	7/6	(i, j)
6	i	7/6	(j)
7	j	7/6	$\emptyset$

(b) Iterations of the algorithm: the second column is the node extracted from Q, the last two columns represent the status of ΔClo and Q at the end of the iteration. Iteration 0 corresponds to lines 1–9 of Algorithm $DynamicBFS$.

x	d_{xu}	$d_{xu}(\{u, v\})$
u	0	0
a	1	1
b	1	1
c	2	2
d	2	2
v	3	1
f	3	2
g	4	2
h	4	3
i	3	3
j	4	4

(c) Distances before and after the edge addition.

Fig. 5. Example of execution of Algorithm *DynamicBFS*.

In order to analyze the computational complexity of Algorithm *DynamicBFS*, let us define as $\gamma_{uv}(S)$ as the set of nodes that change their distance to u as a consequence of the addition of edge $\{u, v\}$ to $G(S)$, that is

$$\gamma_{uv}(S) = \{w \in V \mid d_{xu}(S) \neq d_{xu}(S \cup \{\{u, v\}\})\}.$$

Moreover, let $\Gamma_{uv}(S)$ be the number of edges incident to nodes in $\gamma_{uv}(S)$, that is $\Gamma_{uv}(S) = \sum_{w \in \gamma_{uv}(S)} |N(w)|$. Parameters $|\gamma_{uv}(S)|$ and $\Gamma_{uv}(S)$ measure the minimal number of nodes and edges, respectively, that must be visited in order to update all the distance to u after the addition of edge $\{u, v\}$. Note that $\Gamma_{uv}(S) = O(m+n)$ in the worst case; however, it is much smaller than m in many practical cases as shown in the next section. In Figure 5, the nodes in $\gamma_{uv}(S)$ are represented in gray, while the number of double edges is $\Gamma_{uv}(S)$. The next theorem gives the computational complexity of Algorithm *DynamicBFS* as a function of $O(\Gamma_{uv}(S))$.

THEOREM 2.4. *Algorithm DynamicBFS requires $O(\Gamma_{uv}(S))$ time.*

PROOF. Lines 1–9 require $O(N_v(S)) = O(\Gamma_{uv}(S))$ time. In the loop at lines 10–17, variable *visited* ensures that each node is inserted into Q at most once. Therefore, the overall time requirement of such loop is equal to the sum of $N_x(S)$, for all the nodes x that are inserted into Q. Hence, to prove the statement, we show that all the nodes inserted into Q belong to $\gamma_{uv}(S)$. We first show that for each $x \in \gamma_{uv}(S)$ all the distances $d_{xu}(S \cup \{\{u, v\}\})$ between u and x in $G(S \cup \{\{u, v\}\})$ are correctly computed by Algorithm *DynamicBFS*. By contradiction, suppose that the distance between some node in $\gamma_{uv}(S)$ and u is not correctly computed and consider a node $y \in \gamma_{uv}(S)$ having minimal distance to u among such nodes. At the last iteration when y is inserted into Q, there exists a node $x \in N(y)$ such that $d_{uy}(S) > d_{ux}(S \cup \{u, v\}) + 1$. It follows that $d_{uy}(S \cup \{u, v\}) = d_{ux}(S \cup \{u, v\}) + 1$ (see the test at line 14). Since the distance between y and u is minimal among those that are not correctly computed by the algorithm, then $d_{ux}(S \cup \{u, v\})$ is correct. It follows that the distance between y and u is correctly computed at line 15, a contradiction. By contradiction, suppose that some node not in $\gamma_{uv}(S)$ is inserted into Q and consider a node $y \notin \gamma_{uv}(S)$ having minimal distance to u among such nodes. Since y has minimal distance to u among the nodes not in $\gamma_{uv}(S)$ inserted into Q, then the node x for which the condition at line 14 is satisfied when y is inserted into Q must belong to $\gamma_{uv}(S)$. By the previous arguments, $d_{ux}(S \cup \{u, v\})$ is correctly computed by the algorithm and then $d_{uy}(S \cup \{u, v\}) = d_{ux}(S \cup \{u, v\}) + 1 < d_{uy}(S)$, a contradiction to the fact that y does not belong to $\gamma_{uv}(S)$. □

The new dynamic algorithm can now be obtained by the GREEDYIMPROVEMENT algorithm shown in Figure 3, by doing the following modifications:

—Before line 2, we compute c_u in G.
—At line 4, we incrementally compute $c_u(S \cup \{u, v\})$ by making use of algorithm *DynamicBFS* instead of a full BFS.

Note that, for each $v \in V$, $\Gamma_{uv}(S)$ is maximized when $S = \emptyset$, then the algorithm requires an overall $O(k \cdot n\Gamma)$ computational time, where $\Gamma = \max_{v \in V}\{\Gamma_{uv}(\emptyset)\}$.

We now show how to exploit the definition of submodularity to reduce the running time of iterations $i \geq 2$ of the *for* loop at line 2 of GREEDYIMPROVEMENT. Let $\Delta c_u(S \cup \{\{u, v\}\})$ be the increment to the centrality of node u after adding the edge $\{u, v\}$ to graph $G(S)$. Since c_u is submodular, then $\Delta c_u(S \cup \{\{u, v\}\})$ is monotonic non-increasing. It follows that $\Delta c_u(S \cup \{\{u, v\}\})$ is upper bounded by $\Delta c_u(S' \cup \{\{u, v\}\})$, where $S' \subseteq S$. We exploit this observation in algorithm DYNAMICGREEDYIMPROVEMENT given in Figure 6.

First, we compute c_u and initialize Δc_u (lines 1–3). For each iteration i of the *for* loop at line 6, we use variable LB (line 7) to maintain the maximum improvement to closeness found so far, that is LB is a lower bound to the improvement that will be found at the end of iteration i. If at iteration $i \geq 2$, for some node $v \in V \setminus N_u(S)$, we have that $LB \geq \Delta c_u(\{S' \cup \{\{u, v\}\}\})$ (line 9), where S' is the value of S at iteration $i - 1$, then edge $\{u, v\}$ cannot increase the value of c_u more than the maximum found so far. Therefore, in this case we prune the search. Otherwise, we compute $\Delta c_u(S \cup \{\{u, v\}\})$ and check whether it improves LB or not (line 11). In the affirmative case, we update LB (line 12).

We can improve the performance of DYNAMICGREEDYIMPROVEMENT by means of two further heuristics. First, we sort the nodes of $N_v(S)$, for each $v \in V$, in non-increasing order of distance from u and we stop the *for* loop of line 13 of algorithm *DynamicBFS* when a node y such that $d_{uy}(S) \leq d_{ux}(S \cup \{u, v\}) + 1$ is extracted. In fact, for any other node adjacent to x with a distance to u greater than $d_{uy}(S)$ the condition at line 14 is not satisfied. Then, we can easily parallelize algorithm DYNAMICGREEDYIMPROVEMENT over p processors since $V \setminus (N_v(S))$ can be divided into sets of $\lfloor \frac{|V \setminus (N_v(S))|}{p} \rfloor$ nodes and the *for* loop

Algorithm: DynamicGreedyImprovement
Input : An Undirected graph $G = (V, E)$; a vertex $v \in V$; and an integer $k \in \mathbb{N}$
Output: Set of edges $S \subseteq \{\{u, v\} \mid u \in V \setminus N_v\}$ such that $|S| \leq k$

```
1  Compute c_u by using full BFS;
2  foreach v ∈ V \ N_u do
3      Δc_u({{u, v}}) := 0;
4  S := ∅;
5  S' := ∅;
6  for i = 1, 2, ..., k do
7      LB := 0;
8      foreach v ∈ V \ N_u(S) do
9          if (i = 1) ∨ (LB < Δc_u(S' ∪ {{u, v}})) then
10             Δc_u(S ∪ {{u, v}}) := DynamicBFS(G(S), {u, v}, {d_ux(S)}_{x∈V});
11             if Δc_u(S ∪ {{u, v}}) > LB then
12                 LB := Δc_u(S ∪ {{u, v}});
13                 max := v;
14     S' := S;
15     S := S ∪ {{u, max}};
16     Compute distances d_ux(S), for each x ∈ V;
17 return S
```

Fig. 6. Algorithm DynamicGreedyImprovement.

at line 8 of algorithm DynamicGreedyImprovement can be executed in parallel for each set. In this case, LB is given by the maximum over each subset.

2.5. The Experimental Study: Part I

In this section we analyze the greedy algorithm from an experimental point of view. First, we compare the solution of the greedy algorithm with the optimal solution computed by using an integer program formulation of the MCI problem, in order to assess its real performance in terms of solution quality. Then, we compare the greedy algorithm with several alternative baselines. Finally, we study how the spreading of information increases as a consequence of the augmentation of the graph due to our algorithm.

All our experiments have been performed on a computer equipped with two Intel Xeon E5-2643 CPUs, each with 6 cores clocked at 3.4GHz and 128GB of main memory, and our programs have been implemented in C++ (gcc compiler v4.8.2 with optimization level O3).

2.5.1. Evaluating the Solution Quality. In this section we evaluate the quality of the solution produced by the greedy algorithm by measuring its approximation ratio on several, relatively small, randomly generated networks and on four real-world networks. In particular, we considered four random graph generating models, that is, undirected Preferential Attachment (PA) [Barabasi and Albert 1999], Erdős–Rényi (ER) [Erdős and Rényi 1959], Configuration Model (CM) [Molloy and Reed 1995; Bender and Canfield 1978], and Watts–Strogatz model (WS) [Watts and Strogatz 1998]. The size of the generated graphs is reported in Table I. For each combination (n, m), we generated five random undirected graphs. Moreover, we considered the four real-world graphs, whose size is reported in Table II. The first graph is the collaboration network between Jazz musicians that have played together in a band, and it has been obtained from the Konect database [Kunegis 2013], while the last three graphs have been downloaded

Table I. Comparison between the GREEDYIMPROVEMENT Algorithm and the Optimum in Random Graphs. The First Three Columns Report the Type and Size of the Graphs, and the Fourth Column Reports the Minimum Measured Approximation Ratio

Network	$n = \|V\|$	$m = \|E\|$	Min Approx.
PA	100	130	0.9939
PA	500	650	0.9921
ER	100	200	0.9828
ER	100	500	0.9938
ER	100	1,000	0.9970
ER	500	5000	0.9971
ER	500	12,500	0.9991
ER	500	25,000	1
CM	100	200	0.9946
CM	500	1,000	0.9995
WS	100	500	0.9798
WS	100	600	0.9798
WS	100	800	0.9856
WS	100	1,200	0.9946

Table II. Comparison between the GREEDYIMPROVEMENT Algorithm and the Optimum in Real World Graphs. The First Three Columns Report the Name and Size of the Graphs, and the Fourth Column Reports the Minimum Measured Approximation Ratio

Network	$n = \|V\|$	$m = \|E\|$	Min Approx.
s838_ st	512	819	0.9862
jazz	198	2,742	0.9968
coli1	328	456	0.9947
celegans_metabolic	346	1,493	0.9981

from the Uri AlonLab database: in particular, s838_st is an electronic network, while the other two graphs are biological networks.

For both random and real-world graphs we focused our attention on 20 vertices u, which have been chosen on the basis of their original closeness ranking. In particular, we have divided the list of vertices, sorted by their original ranking, in four intervals, and chosen five random vertices uniformly at random in each interval: we denote by $u_{X\%}$ the average value of the vertices in the interval of the top Xth percentile. The value of k ranged from 1 to 10. In the experiments, we measured the ratio between the value of the solution found by the greedy algorithm and the optimal value computed by using the integer program formulation of the MCI problem, defined as follows:

$$\text{Maximize} \quad \sum_{\substack{s \in V \setminus \{u\} \\ v \in V \setminus N_u}} \left(\frac{1}{d_{su}(\{u, v\})} - \frac{1}{d_{su}} \right) y_{sv} + \sum_{s \in V \setminus \{u\}} \frac{1}{d_{su}}$$

$$\begin{aligned}
\text{subject to} \quad & \sum_{v \in V \setminus N_u} y_{sv} \leq 1, && \text{for each } s \in V \setminus \{u\} \\
& y_{sv} \leq x_v, && \text{for each } s \in V \setminus \{u\},\ v \in V \setminus N_u, \\
& \sum_{v \in V \setminus N_u} x_v \leq k, && \\
& x_v, y_{sv} \in \{0, 1\}, && \text{for each } s \in V \setminus \{u\},\ v \in V \setminus N_u.
\end{aligned}$$

The decision variables x_v and y_{sv} specify a solution S of the MCI problem as follows. For any $v \in V \backslash N_u$,

$$x_v = \begin{cases} 1 & \text{if } \{u, v\} \in S, \\ 0 & \text{otherwise,} \end{cases}$$

and, for each $s \in V \backslash \{u\}$ and $v \in V \backslash N_u$,

$$y_{sv} = \begin{cases} 1 & \text{if a shortest path from } s \text{ to } u \text{ in } G(\{u, v\}) \text{ passes through edge } \{u, v\}, \\ 0 & \text{otherwise.} \end{cases}$$

The first constraint of the integer program ensures that each node s can be covered by at most one edge $\{u, v\}$ and, hence, that the distance from s to u is counted only once in the objective function, while the second constraint ensures that if $y_{sv} = 1$, then $x_v = 1$ and, hence, that the shortest path from s to u passing through $\{u, v\}$ is considered only if $\{u, v\} \in S$. Finally, note that in the objective function, the value of $\frac{1}{d_{su}(\{u,v\})}$ and $\frac{1}{d_{su}}$ can be preprocessed, and that the term $\sum_{s \in V \backslash \{u\}} \frac{1}{d_{su}}$ is a constant.

We solved the above integer program by using the GLPK solver [GNU]. The results are reported in Tables I and II where we show the minimum (i.e., worst-case) approximation ratio obtained by the greedy algorithm. The experiments clearly show that the experimental approximation ratio is by far better than the theoretical one proven in the previous section. In fact, in the worst case the ratio is 0.9798.

In Figure 7, instead, we plot the average closeness centrality and ranking of vertices u as a function of k in a small real-world network, namely the `s838_st` electrical network. We observe that the charts on the top, where the values are computed using the GreedyImprovement algorithm, and the charts on the bottom, in which we used the optimal algorithm, are almost identical. Indeed, the approximation ratio in the worst case is 0.9862: that is, the GreedyImprovement algorithm performs very well in practice.

Finally, we tested our algorithm on several artificial instances generated by the Erdős–Rényi and the Watts–Strogatz models. In the former model we can choose appropriate values of the graph density, while in the latter one we can choose the clustering coefficient. It turned out that the performance of our algorithm is not influenced by these two factors. Indeed, the approximation ratio ranges in $[0.9798, 1]$ and improves a little when the density is very high (i.e., $m > 0.5n^2$).

2.5.2. The Comparison with Alternative Baselines. In this section we compare our algorithm with the following algorithms:

(1) The algorithm that connects u to a set of k nodes extracted uniformly at random (Random).
(2) The algorithm that connects u to a set of k nodes having the highest degree (Degree).
(3) The algorithm that connects u to a set of k nodes having the highest harmonic centrality (Top-k).
(4) The algorithm that connects u to a set of k nodes that have the highest fractional value when solving the linear relaxation of the integer program (Rounding).
(5) The algorithm that connects u to a set of k nodes computed with an approximation algorithm for the k-median with penalties problem given in Meyerson and Tagiku [2009] (k-Median).

The first two algorithms are easy to describe and implement efficiently. In what follows we give more details on the implementation of the last three algorithms.

The Top-k *Algorithm*. The classical algorithm to find the k nodes having the highest value of centrality, consists, for each node v, in determining all the distances to v by running a BFS and computing c_v. With such an approach computing the k nodes having

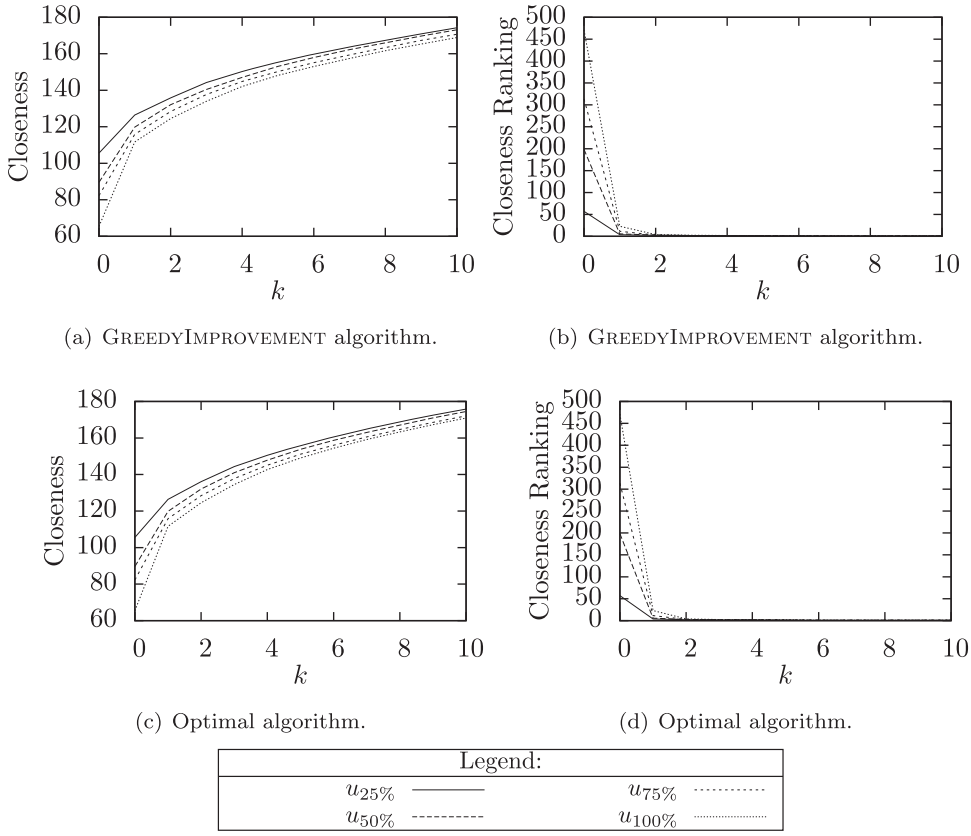


Fig. 7. Closeness centrality and ranking of vertices in the four intervals u as a function of k in the network `s838_st`. Comparison between the GreedyImprovement algorithm and the optimal one.

the highest value of centrality requires $O(n \cdot (n+m))$. In Figures 8 and 9, we give an algorithm that reduces the computation time by using a branch-and-bound technique that prunes the unnecessary BFS by comparing the intermediate results of centrality with a properly defined upper bound.

Let C_k be the set of k nodes having the highest closeness centrality. We represent C_k with a min-heap in order to find the minimum in constant time. First of all, Top-k algorithm inserts the k nodes with highest degree in C_k and computes their centrality. Then, it computes the closeness centrality of other nodes v by performing a BFS starting at each v. The algorithm uses the minimum value of centrality in C_k as a lower bound and prunes the BFS from v when such lower bound is greater than an upper bound (to be defined later) on c_v. Such upper bound is computed every time a node is extracted from the BFS queue. If the BFS is completed without any pruning, it removes the minimum from C_k and it inserts the node v in it.

The upper bound estimates the value of the closeness centrality of a node v. The main idea is that, at each BFS step, when we extract a node x at distance d_{vx} from v, we can maintain the exact number of nodes that are at distance d_{vx} and that are not visited yet. Moreover, we can upper bound the distance to any other node. When x is extracted from the queue, let V_x be the set of nodes at distance d_{vx} from the source v that are not visited, *visited* be the set of nodes currently visited during the BFS, and *Currc* be the value of the closeness centrality at the current step, that is

Algorithm: *PrunedBFS*
Input : An undirected graph $G(S)$; a node v, a double min_c
Output: c_v

$Q := \emptyset$;
$visited := \emptyset$;
$d_{uv}(S \cup \{u, v\}) := 0$;
foreach $x \in N_v(S)$ **do**
    $d_{ux}(S \cup \{u, v\}) := 1$;
    $V_x := V_x + 1$;
$visited := \{u, v\} \cup N_u(S)$;
$c_v := 0$;
foreach $x \in N_v(S)$ **do**
    $Q.push(x)$;
while $\neg Q.empty()$ **do**
    $x := Q.pop()$;
    $V_x := V_x - 1$;
    $c_v := \frac{1}{d_{ux}(S)}$;
    foreach $y \in N_x(S)$ **do**
        if $(y \notin visited) \wedge (d_{uy}(S) > d_{ux}(S \cup \{u, v\} + 1))$ **then**
            $d_{uy}(S \cup \{u, v\}) := d_{ux}(S \cup \{u, v\}) + 1$;
            $Q.push(y)$;
            $visited := visited \cup \{y\}$;
            $V_x := V_x + 1$
    $UB_v := c_v + |V_x| \cdot \frac{1}{d_{vx}} + (|V| - |visited| - |V_x|) \cdot \frac{1}{d_{vx}+1}$;
    if $UB_v \leq min_c$ **then**
        return
return c_v

Fig. 8. Algorithm PrunedBFS.

Algorithm: Top-k
Input : an undirected graph $G = (V, E)$; and an integer $k \in \mathbb{N}$
Output: set of nodes $S \subseteq V$ such that $|S| = k$

S: Min priority queue;
Let $u_1, u_2, \ldots, u_{|V|}$ be the nodes of G sorted according to their degree in non-ascending order;
for $j = 1, 2, \ldots, k$ **do**
    Compute the closeness centrality c_{u_j} of node u_j;
    $S.push(u_j, c_{u_j})$;
for $j = k + 1, k + 2, \ldots, |V|$ **do**
    $min_c := S.getMin()$;
    $c_{u_j} := PrunedBFS(G, u_j, min_c)$;
    if $c_{u_j} > min_c$ **then**
        $S.pop()$;
        $S.push(u_j, c_{u_j})$;
return S

Fig. 9. Algorithm Top-k.

Table III. Real-World Graphs Used in the Comparison between the GREEDYIMPROVEMENT Algorithm and the Other Baselines. The Columns Report the Type and Size of the Graphs

Network	$n = \|V\|$	$m = \|E\|$
advogato	5,272	45,903
ca-AstroPh	17,903	196,972
ca-CondMat	21,363	91,286
ca-HepPh	11,204	117,619
ca-HepTh	8,638	24,806
dip20090126	19,928	41,202
Newman-Cond_mat_95-99	22,015	58,578
PGPgiantcompo	10,680	24,316

$Currc = \sum_{y \in visited} \frac{1}{d_{vy}}$. Then, we have that $|V_x|$ nodes are at distance d_{vx} from v, while the remaining $|V| - |visited| - |V_x|$ nodes are at distance at most $d_{vx} + 1$ from v. Hence the upper bound is defined as

$$UB_v = Currc + |V_x| \cdot \frac{1}{d_{vx}} + (|V| - |visited| - |V_x|) \cdot \frac{1}{d_{vx} + 1}.$$

The ROUNDING *Algorithm*. This approach consists in adding the edges which are obtained by rounding to one k variables of the optimal solution to the linear relaxation of the integer program for the MCI problem given in Section 2.5.1. In particular, we connect u to the nodes corresponding to the k highest values in an optimal fractional solution.

The K-MEDIAN *Algorithm*. This approach consists in connecting u with the k nodes which are a solution of the *k-median with penalties* problem [Meyerson and Tagiku 2009]. In this problem, we are given a set of cities and a set of potential facility locations. Each city has a demand that needs to be served by a facility. Each city also has a penalty cost, which we pay if we refuse to serve the city. If we choose to serve a city, we must pay the distance between the city and its assigned facility for each unit demand. Our job is to find a set of k facilities to open, a set of cities to be served, and an assignment of cities to open facilities such that our total cost is minimized. We implemented the approximation algorithm given in Meyerson and Tagiku [2009] that is based on local search. We give an informal description of such algorithm and refer to Meyerson and Tagiku [2009] for more details. The algorithm starts from a solution S to the k-median with penalties problem and compute its objective function value. Then, at each iteration it changes S by swapping a node in S with a node in $V \backslash S$ and checking whether the objective function value is improved. In the affirmative case, the solution S is updated. The procedure is repeated by selecting another pair of nodes in $S \times V \backslash S$. The algorithm stops when no swap can improve the value of the current solution. In our implementation, at each iteration we select the pair in $S \times V \backslash S$ that maximizes the improvement of the objective function value.

Analysis of the Results. In order to compare the solution obtained by the GREEDYIMPROVEMENT algorithm with that obtained by using the other aforementioned approaches, we run all the algorithms on several real-world networks reported in Tables II and III. We first observe that the rounding and the k-median algorithms cannot be executed on networks having more than a few hundred of nodes in reasonable computational time. Therefore, in what follows we first compare GREEDYIMPROVEMENT with DEGREE, RANDOM, and TOP-K on network ca-HepPh, which is a well-known

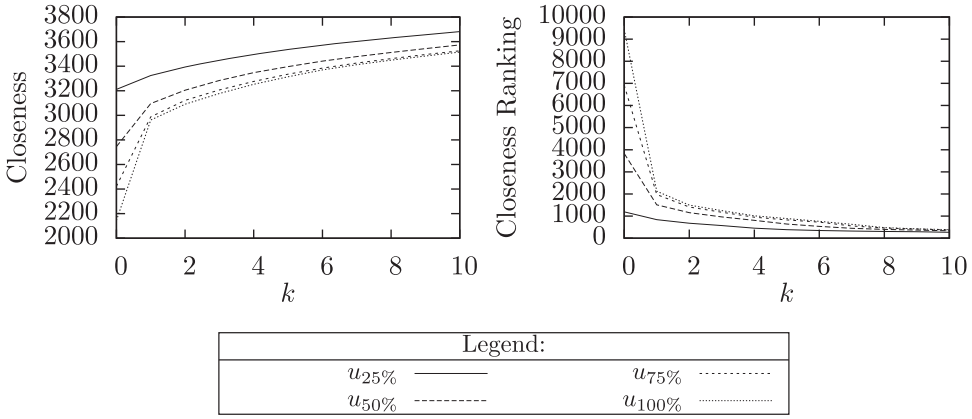


Fig. 10. Closeness centrality and ranking of vertex u as a function of k in network ca-HepPh.

collaboration network obtained from the SNAP database [Leskovec and Krevl 2014], and then we compare GREEDYIMPROVEMENT with ROUNDING and K-MEDIAN on the network jazz. The results for the other networks are similar (but for the networks in Table III and the ROUNDING and K-MEDIAN algorithms for which we have no results due to the high computational time).

Regarding the greedy algorithm, in Figure 10 we plot the closeness centrality and the ranking of vertex u as a function of k on network ca-HepPh. We observe that any vertex becomes central by adding just a few edges. In Figure 11, we compare the ranking obtained with the solution given by our algorithm with that obtained with the solution given by the other approaches on the same network. In particular, we show the *average relative ranking position (ARRP)*, that is,

$$ARRP = \frac{r_u(S_B) - r_u(S_{GR})}{r_u(S_{GR})},$$

where S_{GR} and S_B are the solutions given by our algorithm and one baseline algorithm, respectively, and $r_u(S)$ denotes the closeness ranking of node u in $G(S)$. The average relative ranking position represents the gain of our algorithm over any other baseline in terms of ranking position. Each curve represents the average relative ranking position in a given interval and the values are expressed in percentage. We observe that the greedy algorithm significantly outperforms RANDOM, DEGREE, and TOP-K, whenever $k > 1$.

In Figure 12, we compare, the ranking of node u in the solution given by our algorithm with that given by the ROUNDING and K-MEDIAN. We confirm that our algorithm is by far better than the other approaches.

We observe that similar results hold if we use the closeness value instead of the ranking position to compare the greedy algorithm against the other baselines. In some cases, ROUNDING gives better solutions in terms of objective function value, however, such cases correspond to the instances in which the fractional solution is integral and therefore is optimal for the problem. Note that computing such a solution requires a long computational time and that we cannot apply such an approach for instances having more than a few hundred nodes.

2.5.3. The Analysis of Information Spreading. In this section we analyze how adding a limited number of edges incident to some randomly chosen seeds highly increase the number of nodes that become active in the threshold model.

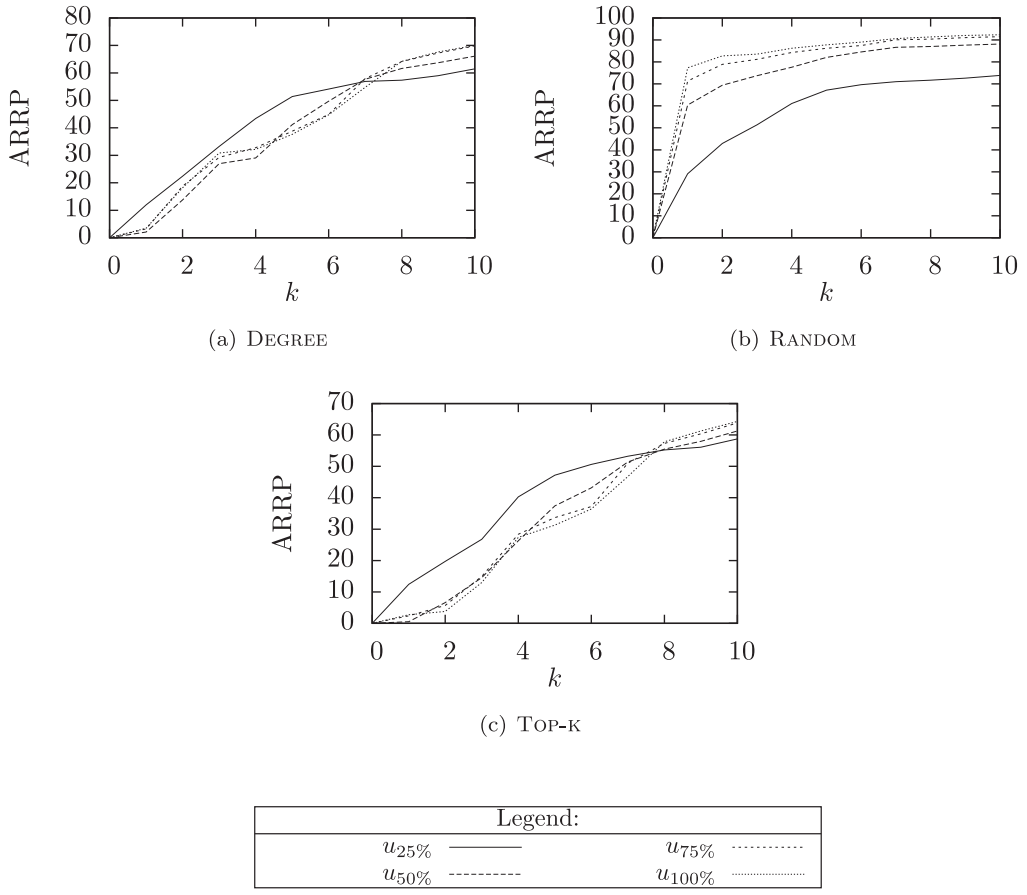


Fig. 11. Average relative ranking position between the Ranking of the solution obtained by the GreedyImprovement algorithm and the different baselines in network `ca-HepPh`.

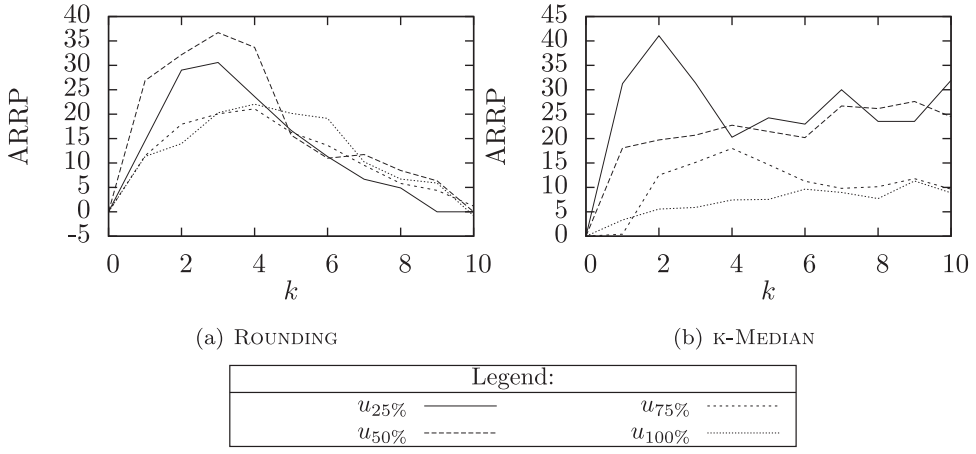


Fig. 12. Average relative ranking position between the Ranking of the solution obtained by the GreedyImprovement algorithm and the different baselines in network `jazz`.

In the experiments we have chosen a number of seeds, that is, 2%, 4%, 6%, 8%, and 10% of the number of nodes of the graph. The seeds are chosen uniformly at random. We run different experiments where the threshold α is uniform, i.e., all the nodes have the same threshold, or distributed uniformly at random. In detail we set a uniform threshold equal to 0.2, 0.3, and 0.4 and a threshold uniformly distributed in the intervals $(0, 0.4]$, $(0, 0.6]$, and $(0, 0.8]$. We measured the number of nodes that become active at the end of the process in the graphs of Tables II and III.

In Figure 13, we plot the percentage of active nodes as a function of k for the `coli1` undirected network for the case of uniform threshold. The value for $k = 0$ is the percentage of active nodes in the original graph. The plots clearly show that the number of active nodes highly increases even with the addition of a few edges and that the percentage of active nodes tends to 100%. The results for the other networks in Tables II and III and for non-uniform threshold are similar.

We run the same experiments by using the solutions obtained by RANDOM, DEGREE, and TOP-K and observe that such solutions are competitive with that obtained by the greedy algorithm if k is small (i.e., $k \leq 3$). If $k > 3$, the percentage of informed nodes is smaller than that obtained by using the greedy algorithm and the gap between the baselines and the greedy algorithm increases with k. As already observed, algorithms ROUNDING and K-MEDIAN can only be executed on small networks.

2.6. The Experimental Study: Part II

We also conducted a second type of experiment by measuring the improvement in the value of closeness of u and in the closeness ranking of u within the network. In particular, we studied two large real-world networks obtained from the DBLP [Ley] and IMDB database [IMDB]. In such networks, the nodes are authors or actors and there is an edge connecting vertex x and vertex y if the author, or actor, corresponding to vertex x collaborated with y for writing a paper or for acting in the same movie. For each graph, we used 20 vertices as u but, differently from the experiments on random graphs, these vertices have been chosen on the basis of their degree ranking: in particular, we divided the list of vertices sorted by their ranking in four parts and chosen randomly five vertices for each interval. The value of k ranges from 1 to 10.

The analysis of these two large networks has been possible only by using the DYNAMICGREEDYIMPROVEMENT algorithm, since this algorithm visits only a few edges of the graph (as explained in the previous section): in particular, for all the iterations $i = 1, 2, \ldots, k$ the algorithm visits only 0.09% of the edges. The results for the `DBLP` network ($n = 1{,}305{,}445$, $m = 6{,}108{,}727$) are plotted in Figure 14, while the results for the `IMDB` network ($n = 1{,}797{,}446$, $m = 72{,}880{,}156$) are similar and are shown in Figure 15. In the chart on the left we plot the closeness centrality of vertex u as a function of k. We observe that any vertex improves its closeness value by adding just a few edges. In the right chart we plot the execution time of the algorithm DYNAMICGREEDYIMPROVEMENT. We notice that the computational effort is high for $k = 1$ but then it is almost constant for $k > 1$: this is due to the submodularity property.

The great difference in execution time between different nodes is due to the dynamic algorithm: indeed, the most a node has a low degree, the most it is likely that adding an edge incident to a central node u can affect the majority of the distances between u and the other nodes (running the dynamic algorithms on nodes with small degree has complexity $O(\Gamma_{uv}(S)) \sim O(m + n)$). Due to the fact that the degree distribution in the big networks analyzed is not uniform, the difference between the execution times of $u_{25\%}$ and $u_{50\%}$, $u_{50\%}$ and $u_{75\%}$, and $u_{75\%}$ and $u_{100\%}$ is not proportional to the difference of the positions in the initial ranking.

In order to test the scalability of the parallelized version of DYNAMICGREEDYIMPROVEMENT algorithm, we run the same set of experiments with different numbers of cores

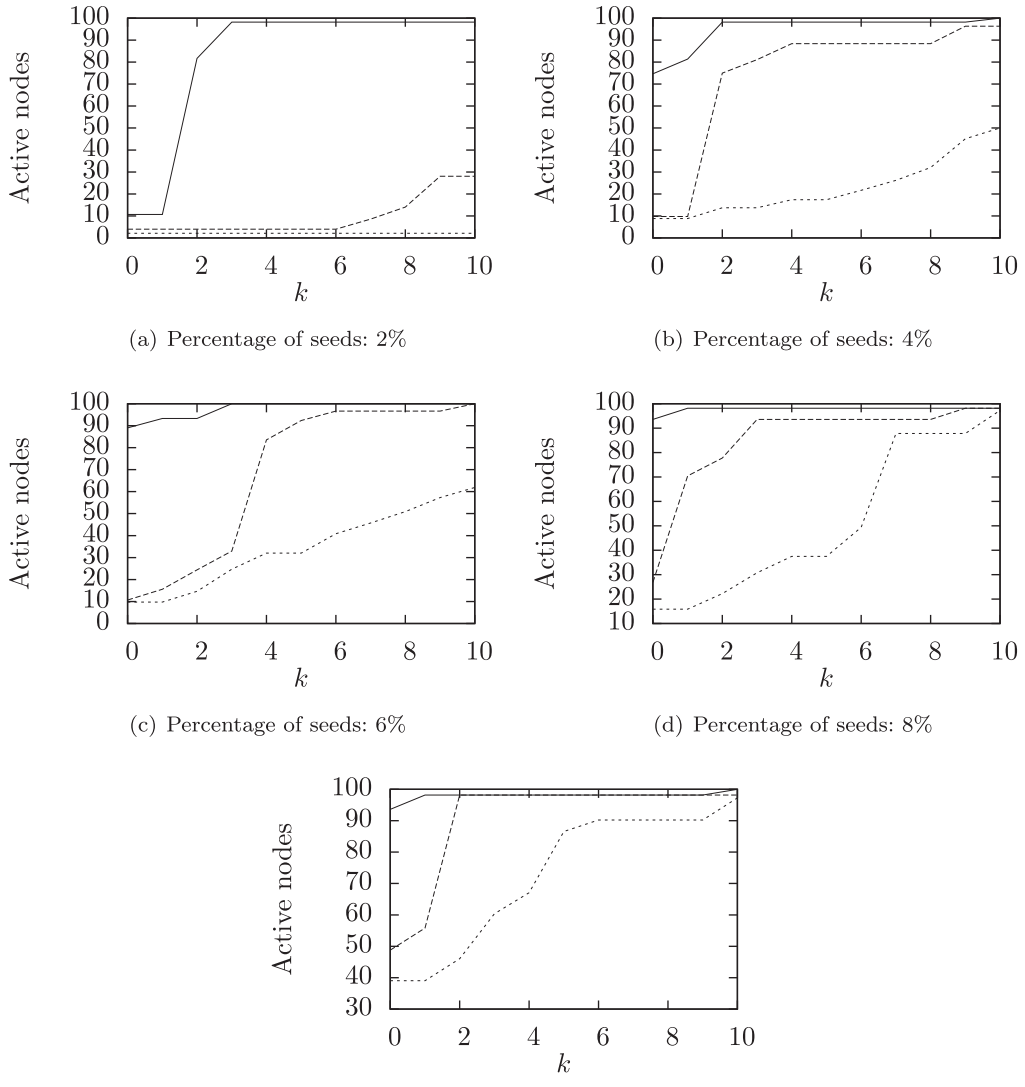


Fig. 13. Percentage of active nodes in `coli1` network as a function of k in the case of uniform threshold. Parameter a denotes the threshold and the percentage of seeds is equal to 2%, 4%, 6%, 8%, and 10%, respectively.

and we measured the execution time and the speedup, i.e., the ratio between the execution time with one core and the execution time with p cores for $p = 2, 4, 6, 8$. The results are reported in Table IV. We notice that the parallel algorithm shows a good scalability in terms of execution time. Note that the small increase in the case of eight

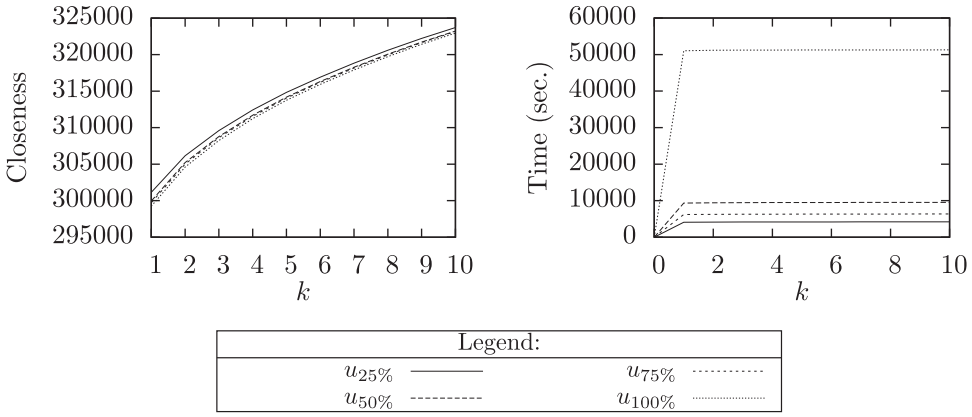


Fig. 14. Performance of DynamicGreedyImprovement algorithm on network DBLP.

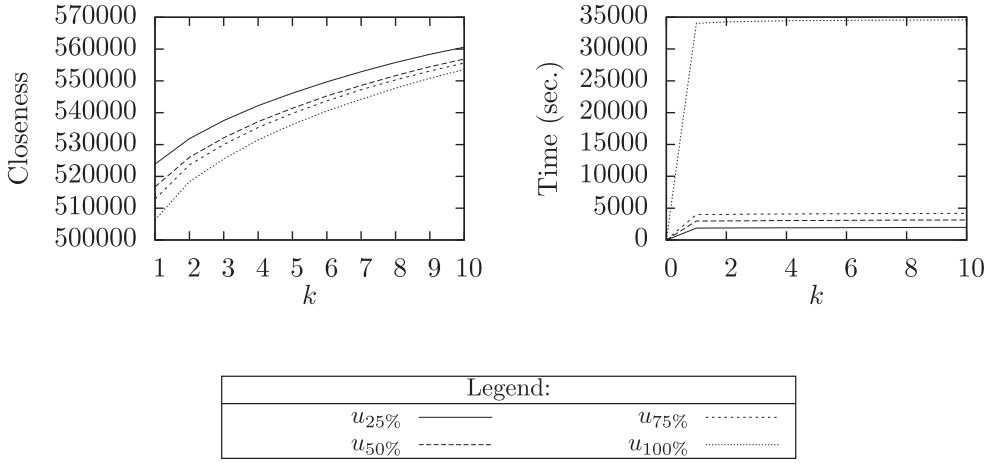


Fig. 15. Performance of DynamicGreedyImprovement algorithm on network IMDB.

Table IV. Execution Time and Speedup of DynamicGreedyImprovement Algorithm on DBLP and IMDB Networks with Different Number of Cores

Network	Avg. Execution Time (1 core)	Avg. Speedup (2 cores)	Avg. Speedup (4 cores)	Avg. Speedup (6 cores)	Avg. Speedup (8 cores)
DBLP	17,671s	1.91	3.01	4.01	4.63
IMDB	10,710s	1.57	2.10	3.12	3.27

cores is due to the fact that in our machine, each CPU has six physical cores and hence in the case of eight cores the computation is performed by two different processors.

3. THE DIRECTED GRAPH CASE

In this section we will focus on directed graphs. After giving all necessary definitions and preliminary results, we will first introduce the optimization problem that will be considered, then prove a non-approximability result, and lastly describe an almost optimal approximation algorithm. Finally, we will present the experiments that we

have performed in order to validate this algorithm and to apply it to a quite big citation network and a web graph.

3.1. The Maximum Directed Closeness Improvement Problem

Let $G = (V, A)$ be a directed graph, where V denotes the set of nodes, and A denotes the set of arcs (u, v) with u and v in V (note that $(u, v) \in A$ does not imply that $(v, u) \in A$). For each node u, N_u denotes the set of in-neighbors of u, i.e., $N_u = \{v \mid (v, u) \in A\}$. Given two vertices u and v, d_{vu} is defined as in the undirected graph case. Given a set S of arcs not in A, we denote by $G(S)$ the graph augmented by adding the arcs in S to G, i.e., $G(S) = (V, A \cup S)$. Once again, for a parameter x of G, we denote by $x(S)$ the same parameter in graph $G(S)$. For each node u, the closeness centrality of u is defined as follows:

$$c_u = \sum_{\substack{v \in V \setminus \{u\} \\ d_{vu} < \infty}} \frac{1}{d_{vu}}.$$

Given a directed graph $G = (V, A)$, a vertex $u \in V$, and an integer k, the *Maximum Directed Closeness Improvement* (MDCI) problem consists in finding a set S of arcs entering u not in A (that is, $S \subseteq \{(v, u) : v \in V \setminus N_u\}$) such that $|S| \leq k$ and $c_u(S)$ is maximum.

We observe that the following results hold also for the related problem in which the edges to be added to the graph are outgoing from u and the closeness centrality considers distances d_{uv} instead of d_{vu}.

3.2. The Non-Approximability Result

In this section, in order to derive our approximation hardness result for the MDCI problem, we will make use of the *Maximum Set Coverage* (MSC) problem, which is defined as follows: given a set X, a collection $\mathcal{F} = \{S_1, S_2, \ldots, S_{|\mathcal{F}|}\}$ of subsets of X, and an integer k, find a sub-collection $\mathcal{F}' \subseteq \mathcal{F}$ such that $|\mathcal{F}'| \leq k$ and $s(\mathcal{F}') = |\cup_{S_i \in \mathcal{F}'} S_i|$ is maximized. It is known that the MSC problem cannot be approximated within a factor greater than $1 - \frac{1}{e}$, unless $P = NP$ [Feige 1998]. We will now use this result in order to show that the MDCI problem does not admit a polynomial-time approximation scheme.

THEOREM 3.1. *The MDCI problem cannot be approximated within a factor greater than* $1 - \frac{1}{3e}$, *unless* $P = NP$.

PROOF. We give an L-reduction with parameters a and b [Papadimitriou and Yannakakis 1991]. In detail, we will give a polynomial-time algorithm that transforms any instance I_{MSC} of MSC into an instance I_{MDCI} of MDCI and a polynomial-time algorithm that transforms any solution S for I_{MDCI} into a solution $\mathcal{F}'$ for I_{MSC} such that the following two conditions are satisfied for some values a and b:

$$OPT(I_{\text{MDCI}}) \leq a OPT(I_{\text{MSC}}) \tag{2}$$

$$OPT(I_{\text{MSC}}) - s(\mathcal{F}') \leq b(OPT(I_{\text{MDCI}}) - c_u(S)), \tag{3}$$

where OPT denotes the optimal value of an instance of an optimization problem. If the above conditions are satisfied and there exists a α-approximation algorithm for MDCI, then there exists a $(1 - ab(1 - \alpha))$-approximation algorithm for MSC [Papadimitriou and Yannakakis 1991]. Since MSC is hard to approximate within a factor greater than $1 - \frac{1}{e}$, then $1 - ab(1 - \alpha) < 1 - \frac{1}{e}$, unless $P = NP$. This implies that $\alpha < 1 - \frac{1}{abe}$.

Given an instance $I_{\text{MSC}} = (X, \mathcal{F}, k)$ of MSC, we define an instance $I_{\text{MCI}} = (G, u, k)$ of MDCI as follows (see Figure 16): $G = (V, A)$, where $V = \{u\} \cup \{v_{x_i} \mid x_i \in X\} \cup \{v_{S_j} \mid S_j \in \mathcal{F}\}$ and $A = \{(v_{x_i}, v_{S_j}) \mid x_i \in S_j\}$.

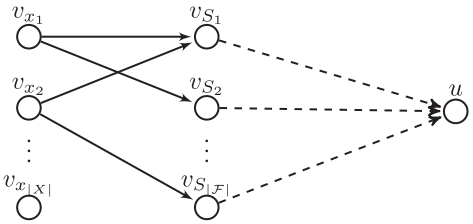


Fig. 16. The reduction used in Theorem 2.1 (in this example, $x_1 \in S_1$, $x_1 \in S_2$, $x_2 \in S_1$, and $x_2 \in S_{|\mathcal{F}|}$). The dashed arcs denote those added in a solution.

Without loss of generality, we can assume that any solution S of MDCI contains only arcs (v_{S_j}, u) for some $S_j \in \mathcal{F}$. In fact, if a solution does not satisfy this property, then we can improve it in polynomial time by repeatedly applying the following rule: if S contains an arc (v_{x_i}, u), for some $x_i \in X$, then exchange such arc with an arc (v_{S_j}, u) such that $(v_{S_j}, u) \notin S$ (note that such an arc must exist, since otherwise $|\mathcal{F}| \leq k$ and I_{MSC} could be easily solved). The above rule does not decrease the value of $c_u(S)$: indeed, if we exchange an arc (v_{x_i}, u) with an arc (v_{S_j}, u) such that $(v_{S_j}, u) \notin S$, then the closeness centrality of u decreases by either 1 or $\frac{1}{2}$ (because of the deletion of (v_{x_i}, u)) but certainly increases by 1 (because of the insertion of (v_{S_j}, u)).

Given a solution S of MDCI, let $\mathcal{F}'$ be the solution of MSC such that $S_j \in \mathcal{F}'$ if and only if $(v_{S_j}, u) \in S$. We now show that $c_u(S) = \frac{1}{2}s(\mathcal{F}') + k$. To this aim, let us note that the distance from a vertex v_{x_i} to u is equal to 2 if an arc (x_{S_j}, u) such that $x_i \in S_j$ belongs to S, and it is ∞ otherwise. Similarly, the distance from a vertex v_{S_j} to u is equal to 1 if $(x_{S_j}, u) \in S$, and it is ∞ otherwise. Moreover, the set of elements x_i of X such that $d_{v_{x_i}u}(S) < \infty$ is equal to $\{x_i \mid x_i \in S_j \wedge (v_{S_j}, u) \in S\} = \bigcup_{S_j \in \mathcal{F}'} S_j$. Therefore,

$$
\begin{aligned}
c_u(S) = \sum_{\substack{v \in V \setminus \{u\} \\ d_{vu}(S) < \infty}} \frac{1}{d_{vu}(S)} &= \sum_{\substack{x_i \in X \\ d_{v_{x_i}u}(S) < \infty}} \frac{1}{d_{v_{x_i}u}(S)} + \sum_{\substack{S_j \in \mathcal{F} \\ d_{v_{S_j}u}(S) < \infty}} \frac{1}{d_{v_{S_j}u}(S)} \\
&= \frac{1}{2}|\{x_i \in X \mid d_{v_{x_i}u}(S) < \infty\}| + |\{S_j \in \mathcal{F} \mid d_{v_{S_j}u}(S) < \infty\}| \\
&= \frac{1}{2}\left|\bigcup_{S_j \in \mathcal{F}'} S_j\right| + |\{S_j \mid (v_{S_j}, u) \in S\}| = \frac{1}{2}s(\mathcal{F}') + k.
\end{aligned}
$$

It follows that Conditions (2) and (3) are satisfied for $a = \frac{3}{2}$ and $b = 2$. Indeed, $OPT(I_{\mathrm{MDCI}}) = \frac{1}{2}OPT(I_{\mathrm{MSC}}) + k \leq \frac{3}{2}OPT(I_{\mathrm{MSC}})$, where the inequality is due to the fact that $OPT(I_{\mathrm{MSC}}) \geq k$, since otherwise the greedy algorithm would find an optimal solution for I_{MSC}. Moreover, $OPT(I_{\mathrm{MSC}}) - s(\mathcal{F}') = 2(OPT(I_{\mathrm{MDCI}}) - k) - 2(c_u(S) - k) = 2(OPT(I_{\mathrm{MDCI}}) - c_u(S))$. The theorem follows by plugging the values of a and b into $\alpha < 1 - \frac{1}{abe}$. □

3.3. The Greedy Approximation Algorithm

As in the case of undirected graphs, we can show that for each vertex u, the function c_u is monotone and submodular with respect to any feasible solution for MDCI. Indeed, the proof of Theorem 2.2 can be easily adapted to the directed graph case, and the following result holds.

Algorithm: DIRECTEDGREEDYIMPROVEMENT
Input : a directed graph $G = (V, A)$, a vertex $u \in V$, and an integer $k \in \mathbb{N}$
Output: set of arcs $S \subseteq \{(v, u) \mid v \in V \setminus N_u\}$ such that $|S| \leq k$

```
1 S := ∅;
2 for i = 1, 2, ..., k do
3     foreach v ∈ V \ N_u(S) do
4         Compute c_u(S ∪ {(v, u)})
5     v_max := arg max{c_u(S ∪ {(v, u)}) | v ∈ V \ N_u(S)};
6     S := S ∪ {(v_max, u)};
7 return S;
```

Fig. 17. The greedy algorithm for directed graphs.

COROLLARY 3.2. *The algorithm shown in Figure 17 is a* $(1 - \frac{1}{e})$*-approximation algorithm for the MDCI problem.*

Also in this case, there is a gap between the non-approximability result proved in Theorem 3.1 (that is, the upper bound equal to $1 - \frac{1}{3e} \approx 0.88$), and the approximability result of the above corollary (that is, the lower bound $(1 - \frac{1}{e}) \approx 0.63$). One of the main goals of the next experimental session is to analyze the "real" performance, in terms of solution quality, of the greedy algorithm on relatively small real-world and synthetic graphs.

3.4. Experimental Results

As in the undirected graph case in this section we analyze the greedy algorithm from an experimental point of view. First, we compare the solution of the greedy algorithm with the optimal solution in order to assess its real performance in terms of solution quality. Then, we compare the greedy algorithm with the other approaches used in the undirected cases, but K-MEDIAN because it cannot be applied in the directed case. We adapted the DYNAMICGREEDYIMPROVEMENT algorithm used for the undirected case. In particular, we run the (pruned) BFSs on the transpose graph of G to reduce the time required to compute c_u and to improve the computational complexity of GREEDYIMPROVEMENT. Moreover, we show the results of our experiments on a real-world graph measuring the improvement in the value of closeness of u within the network. Finally, we analyze how the information spread increases.

We measured the approximation ratio of the greedy algorithm on five types of randomly generated directed networks, namely directed PA [Bollobás et al. 2003], ER [Erdős and Rényi 1959], Copying (COPY) [Kumar et al. 2000], Compressible Web (COMP) [Chierichetti et al. 2009] and Forest Fire (FF) [Leskovec et al. 2007]. The size of the graphs is reported in Table V. For each combination (n, m), we generated five random directed graphs and used 20 vertices as u. These vertices have been chosen on the basis of their original closeness ranking as in the undirected case.

The results of the experiments are reported in Table V, where we show, similarly to the undirected graph case, the minimum ratio obtained. The experiments show that in the worst case the ratio is 0.9668.

For the comparison with the other approaches we used real-world citation networks obtained from the Arnetminer database [Arnetminer] (see Table VI for details). In Arnetminer's networks, there is a vertex for each author and an arc from vertex x to vertex y if the author corresponding to vertex x cited in his paper one paper written by the author corresponding to y. We parsed the Arnetminer database in order to select a sub-network induced by the authors that published at least a paper in one of the

Table V. Comparison between the DIRECTEDGREEDYIMPROVEMENT Algorithm and the Optimum. The First Three Columns Report the Type and Size of the Graphs, and the Fourth Column Reports the Approximation Ratio

Network	$n = \|V\|$	$m = \|E\|$	Min Approx. Ratio
PA	100	130	0.9816
PA	500	650	0.9956
PA	1,000	1,300	1
ER	100	200	0.9668
ER	100	500	0.9744
ER	100	1,000	0.9780
ER	500	5,000	0.9890
ER	500	12,500	0.9819
ER	500	25,000	0.9994
COMP	100	200	0.9968
COMP	100	500	0.9764
COMP	100	1,000	1
COMP	500	5000	0.9848
COMP	500	12,500	1
COMP	500	25,000	1
COPY	100	200	0.9911
COPY	100	500	0.9753
COPY	100	1,000	0.9820
COPY	500	5000	0.9825
COPY	500	12,500	0.9726
COPY	500	25,000	0.9690
FF	100	200	0.9911
FF	200	400	0.9714
FF	500	1,000	0.9892

Table VI. Collaboration Networks Obtained from Arnetminer Database

Network	$n = \|V\|$	$m = \|E\|$
Software Engineering	3,141	14,787
Information Security	1,067	4,253
Computer Graphics Multimedia	8,336	41,925
Theoretical Computer Science	4,172	14,272
Artificial Intelligence	27,617	268,460
High-Performance Computing	4,869	35,036
Computer Networks	9,420	53,003
Interdisciplinary Studies	577	1,504

main conferences or journals. As in the previous experiment, for each graph, we used 20 vertices as u. The value of k ranges from 1 to 10.

The results for the citation network Information Security are plotted in Figure 18. In the two charts we plot the closeness centrality and the ranking of vertex u as a function of k. We observe that any vertex becomes central by adding just a few arcs. For example, a vertex with the smallest closeness centrality which initially has closeness 0 and is ranked 509 improves its closeness and ranking to 213.32 and 1, respectively, by adding only seven arcs.

In the chart in Figure 19 we compare the greedy algorithm with the other approaches. We report the comparison of the average relative ranking position reached by the nodes. We do not show the results for ROUNDING as such algorithm is not able to terminate within a reasonable amount of time. The experiments clearly show that the greedy

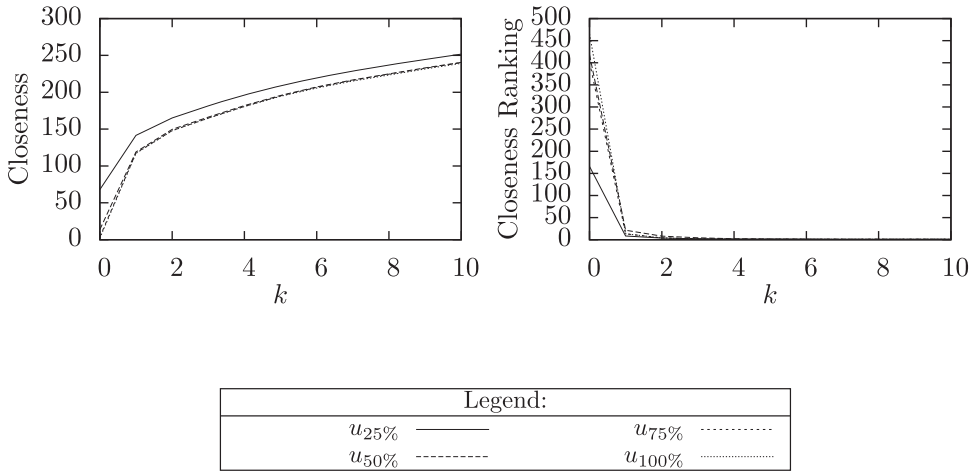


Fig. 18. Performance of the DIRECTEDGREEDYIMPROVEMENT algorithm on network `Information Security`.

Table VII. Execution Time and Speedup of DYNAMICGREEDYIMPROVEMENT Algorithm on `uk-2007` Network with Different Number of Cores

Network	Avg. Execution Time (1 core)	Avg. Speedup (2 cores)	Avg. Speedup (4 cores)	Avg. Speedup (6 cores)	Avg. Speedup (8 cores)
`uk-2007`	1,382s	1.83	3.70	5.58	7.44

algorithm outperforms the other approaches. Also in this case, similar results hold if we use the closeness value instead of the ranking position to compare the greedy algorithm against the other approaches.

We further used the DYNAMICGREEDYIMPROVEMENT algorithm to analyze a web network [LAW]. The results for the web network `uk-2007` ($n = 100{,}000$, $m = 3{,}050{,}615$) are plotted in Figure 20. In the right chart, we report the execution time of the algorithm. The DYNAMICGREEDYIMPROVEMENT algorithm is up to 10^3 times faster than the basic GREEDYIMPROVEMENT algorithm and for all the iteration it visits only the 0.18% of the arcs of the graph: using the DYNAMICGREEDYIMPROVEMENT algorithm, it is possible to solve the MCI problem on very large graphs where it is impossible to obtain a solution using the GREEDYIMPROVEMENT algorithm.

Like in the undirected case, we run the same set of experiments with different numbers of cores and we measured the execution time and the speedup, i.e., the ratio between the execution time with one core and the execution time with p cores for $p = 2, 4, 6, 8$. The results are reported in Table VII. We notice that the parallel algorithm shows a very good scalability in terms of execution time.

Finally, we measured the increase in information spreading when a few edges are added to a small set of randomly-chosen seeds. We performed experiments similar to those shown in the undirected case in the graphs of Table VI. In Figure 21, we plot the results for the `Information_security` directed network in the case of uniform threshold. We observe that also in this case the informed nodes percentage increases. The results for the other networks in Table VI and for non-uniform threshold are similar. As for the case of undirected graphs, we observe that when k is big enough, the percentage of nodes that become informed by using algorithms RANDOM, DEGREE, and TOP-K is smaller than that obtained by using the greedy algorithm and that the gap between the other baselines and the greedy algorithm increases with k.

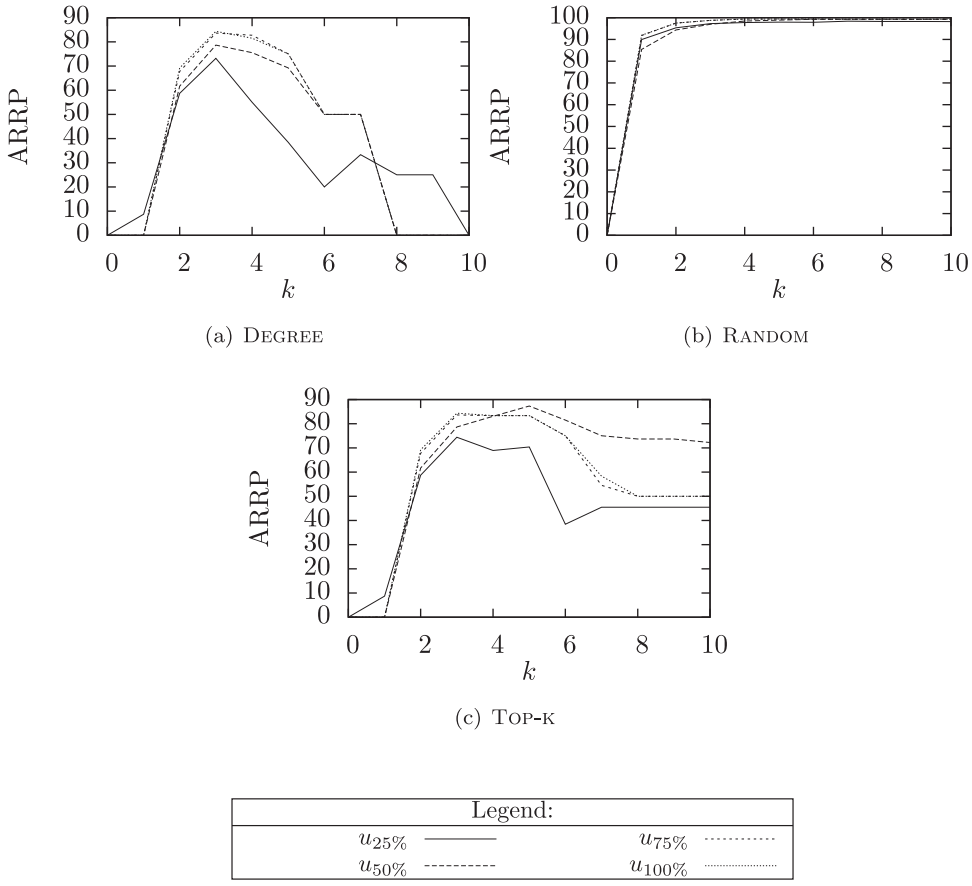


Fig. 19. Average relative ranking position between the Ranking obtained by the DirectedGreedyImprovement algorithm and the different baselines in the network `Information_security`.

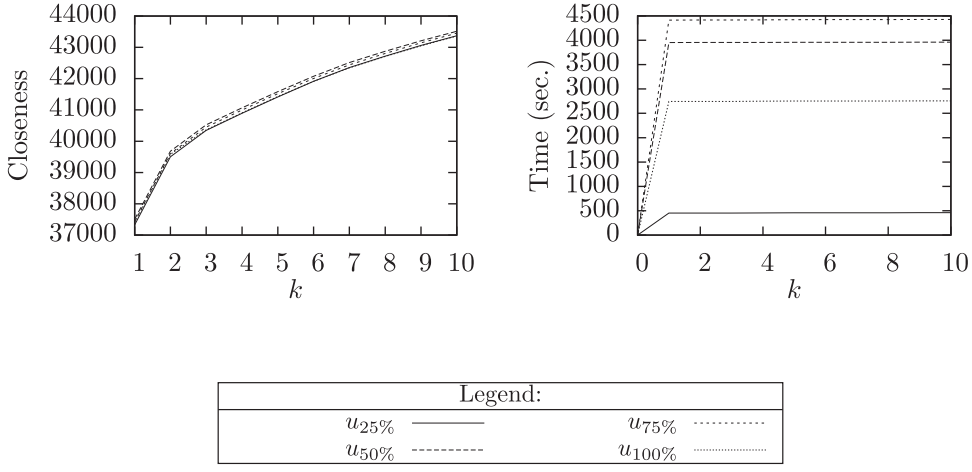
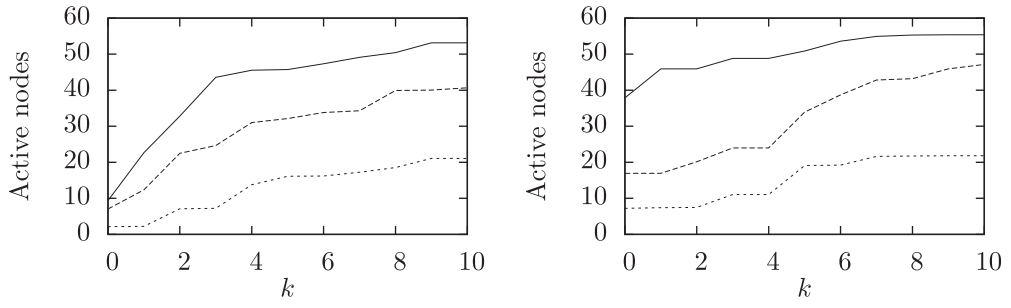
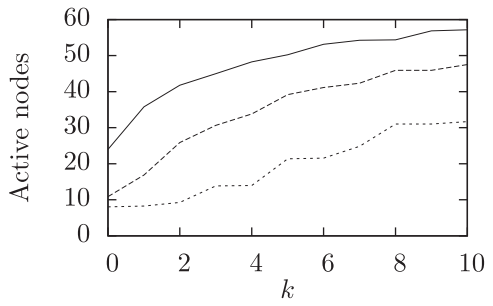


Fig. 20. Performance of the DynamicGreedyImprovement algorithm on network `uk-2007`.

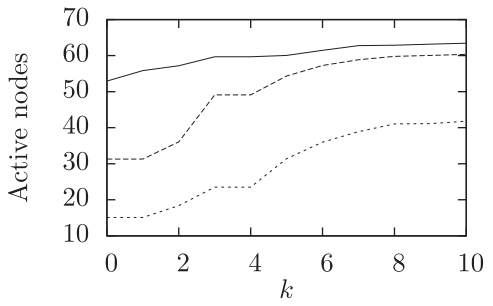


(a) Percentage of seeds: 2%

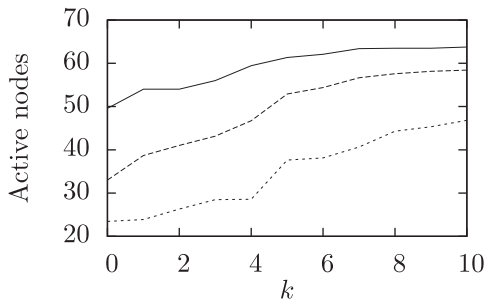
(b) Percentage of seeds: 4%



(c) Percentage of seeds: 6%



(d) Percentage of seeds: 8%



(e) Percentage of seeds: 10%

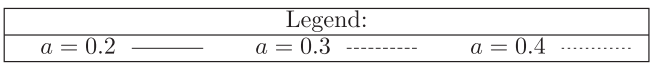


Fig. 21. Percentage of active nodes in `Information_security` network as a function of k in the case of uniform threshold. Parameter a denotes the threshold and the percentage of seeds is equal to 2%, 4%, 6%, 8%, and 10%, respectively.

4. CONCLUSION AND FUTURE RESEARCH

We considered the problem of adding k edges in a (directed or undirected) graph in order to maximize the closeness of a predefined vertex. For undirected graphs, we have shown that the problem cannot be approximated within a factor larger than $1 - \frac{1}{15e}$, and we proposed a greedy algorithm that guarantees an approximation factor of $1 - \frac{1}{e}$.

For directed graphs, the problem cannot be approximated within a factor larger than $1 - \frac{1}{3e}$, while the greedy algorithm still guarantees an approximation factor of $1 - \frac{1}{e}$. We experimentally evaluated such algorithms and showed that they often compute an optimal solution and, in any case, they achieve an approximation factor significantly better than the theoretical one. Moreover, by adding a very few edges a vertex can drastically increase its centrality measure and its ranking.

As future works, we plan to extend our work to further centrality measures such as betweenness, to generalize the problem by allowing the addition of edges incident to other nodes of the graph, and to maximize the ranking of a node instead of the centrality value.

REFERENCES

Arnetminer. Accessed January 15, 2015 from http://arnetminer.org.

Konstantin Avrachenkov and Nelly Litvak. 2006. The effect of new links on Google PageRank. *Stoch. Models* 22, 2 (2006), 319–331.

Lars Backstrom and Jure Leskovec. 2011. Supervised random walks: Predicting and recommending links in social networks. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*. ACM, 635–644.

Albert-Laszlo Barabasi and Reka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (1999), 509–512.

Reinhard Bauer, Gianlorenzo D'Angelo, Daniel Delling, Andrea Schumm, and Dorothea Wagner. 2012. The shortcut problem – Complexity and algorithms. *J. Graph Algorithms Appl.* 16, 2 (2012), 447–481.

Edward A. Bender and E. Rodney Canfield. 1978. The asymptotic number of labeled graphs with given degree sequences. *J. Comb. Theory Ser. A* 24, 3 (1978), 296–307.

Davide Bilò, Luciano Gualà, and Guido Proietti. 2012. Improved approximability and non-approximability results for graph diameter decreasing problems. *Theor. Comput. Sci.* 417 (2012), 12–22.

Paolo Boldi and Sebastiano Vigna. 2014. Axioms for centrality. *Internet Math.* 10, 3–4 (2014), 222–262.

Béla Bollobás, Christian Borgs, Jennifer Chayes, and Oliver Riordan. 2003. Directed scale-free graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*. SIAM, 132–139.

Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Alessandro Panconesi, and Prabhakar Raghavan. 2009. Models for the compressible web. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science (FOCS'09)*. IEEE, 331–340.

Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F. Werneck. 2014. *Computing Classic Closeness Centrality, at Scale*. Technical Report MSR-TR-2014-71. http://research.microsoft.com/apps/pubs/default.aspx?id=217367.

Pierluigi Crescenzi, Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. 2015. Greedily improving our own centrality in a network. In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*. Lecture Notes in Computer Science, Vol. 9125. Springer, 43–55.

Gianlorenzo D'Angelo, Lorenzo Severini, and Yllka Velaj. 2016. On the maximum betweenness improvement problem. *Electron. Notes Theor. Comput. Sci.* 322 (2016), 153–168. Proceedings of the 16th Italian Conference on Theoretical Computer Science (ICTCS'15).

Sina Dehghani, Mohammad Amin Fazli, Jafar Habibi, and Sadra Yazdanbod. 2015. Using shortcut edges to maximize the number of triangles in graphs. *Oper. Res. Lett.* 43, 6 (2015), 6.

Erik D. Demaine and Morteza Zadimoghaddam. 2010. Minimizing the diameter of a network using shortcut edges. In *Proceedings of the 12th Scandinavian Symposium and Workshop on Algorithm Theory (SWAT'10)*. Lecture Notes in Computer Science, Vol. 6139. Springer, 420–431.

I. Dinur and D. Steurer. 2014. Analytical approach to parallel repetition. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. ACM, 624–633.

P. Erdős and A. Rényi. 1959. On random graphs I. *Publ. Math.* 6 (1959), 290–297.

Uriel Feige. 1998. A threshold of ln n for approximating set cover. *J. ACM* 45, 4 (1998).

Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. 2015. Augmenting graphs to minimize the diameter. *Algorithmica* 72, 4 (2015), 995–1010.

GNU. GLPK – GNU Linear Programming Kit. http://www.gnu.org/software/glpk.

IMDB. Accessed January 15, 2015 from http://www.imdb.com.

Vatche Ishakian, Dóra Erdös, Evimaria Terzi, and Azer Bestavros. 2012. A framework for the evaluation and management of network centrality. In *Proceedings of the 12th SIAM Int. Conf. on Data Mining (SDM)*. SIAM, 427–438.

Miray Kas, Matthew Wachs, Kathleen M. Carley, and L. Richard Carley. 2013. Incremental algorithm for updating betweenness centrality in dynamically growing networks. In *Proceedings of Advances in Social Networks Analysis and Mining (ASONAM'13)*. ACM, 33–40.

David Kempe, Jon Kleinberg, and Éva Tardos. 2015. Maximizing the spread of influence through a social network. *Theory Comput.* 11, 4 (2015), 105–147.

Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tomkins, and Eli Upfal. 2000. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS'00)*. IEEE, 57–65.

Jérome Kunegis. 2013. KONECT – The Koblenz network collection. In *Proceedings of the 1st International Web Observatory Workshop (WOW'13)*. 1343–1350.

LAW. Laboratory for Web Algorithmics. Accessed January 15, 2015 from http://law.di.unimi.it/index.php.

Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data* 1, 1 (March 2007), Article no. 2. DOI:http://dx.doi.org/10.1145/1217299.1217301

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

Michael Ley. DBLP. Accessed January 15, 2015 from http://dblp.uni-trier.de/.

Rong-Hua Li and Jeffrey Xu Yu. 2015. Triangle minimization in large networks. *Knowl. Inf. Syst.* 45, 3 (2015), 617–643.

David Liben-Nowell and Jon Kleinberg. 2003. The link prediction problem for social networks. In *Proceedings of the 12th International Conference on Information and Knowledge Management (CIKM'03)*. ACM, 556–559.

Adam Meyerson and Brian Tagiku. 2009. Minimizing average shortest path distances via shortcut edge addition. In *Proceedings of the 13th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'09)*. Lecture Notes in Computer Science, Vol. 5687. Springer, 272–285.

Michael Molloy and Bruce Reed. 1995. A critical point for random graphs with a given degree sequence. *Random Struct. Algorithms* 6, 2–3 (1995), 161–180.

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

Martin Olsen and Anastasios Viglas. 2014. On the approximability of the link building problem. *Theor. Comput. Sci.* 518 (2014), 96–116.

Christos H. Papadimitriou and Mihalis Yannakakis. 1991. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.* 43, 3 (1991), 425–440.

Manos Papagelis. 2015. Refining social graph connectivity via shortcut edge addition. *ACM Trans. Knowl. Discov. Data* 10, 2 (2015), 12.

Manos Papagelis, Francesco Bonchi, and Aristides Gionis. 2011. Suggesting ghost edges for a smaller world. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM'11)*. ACM, 2305–2308.

Nikos Parotsidis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2015. Selecting shortcuts for a smaller world. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 28–36.

Nikos Parotsidis, Evaggelia Pitoura, and Panayiotis Tsaparas. 2016. Centrality-aware link recommendations. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM'16)*. ACM, 503–512.

S. Perumal, P. Basu, and Ziyu Guan. 2013. Minimizing eccentricity in composite networks via constrained edge additions. In *Proceedings of IEEE Military Communications Conference (MILCOM'13)*. 1894–1899.

Alexandrin Popescul and Lyle H. Ungar. 2003. Statistical relational learning for link prediction. In *Proceedings of IJCAI Workshop on Learning Statistical Models from Relational Data*.

Sudip Saha, Abhijin Adiga, B. Aditya Prakash, and Anil Kumar S. Vullikanti. 2015. Approximation algorithms for reducing the spectral radius to control epidemic spread. In *Proceedings of the 2015 SIAM International Conference on Data Mining*. SIAM, 568–576.

Ahmet Erdem Sariyüce, Kamer Kaya, Erik Saule, and Ümit V. Çatalyürek. 2013. Incremental algorithms for closeness centrality. In *Proceedings of the 2013 IEEE International Conference on Big Data*. IEEE, 487–492.

Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2012. Gelling, and melting, large graphs by edge manipulation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12)*. ACM, 245–254.

Uri AlonLab. Accessed January 15, 2015 from http://www.weizmann.ac.il/mcb/UriAlon/.

Duncan J. Watts and Steven H. Strogatz. 1998. Collective dynamics of 'small-world' networks. *Nature* 393, 6684 (1998), 440–442.

Erjia Yan and Ying Ding. 2009. Applying centrality measures to impact analysis: A coauthorship network analysis. *J. Assoc. Inf. Sci. Technol.* 60, 10 (2009), 2107–2118.

Zhijun Yin, Manish Gupta, Tim Weninger, and Jiawei Han. 2010. A unified framework for link recommendation using random walks. In *Proceedings of International Conference on Advances in Social Networks Analysis and Mining (ASONAM'10)*. IEEE Computer Society, 152–159.