Karlsruhe Institute of Technology

Computing Top-k Closeness Centrality in Fully-dynamic Graphs

Master's Thesis

Patrick Bisenius

1640015

At the Department of Informatics

Institute of Theoretical Informatics

Reviewer: Jun. Prof. Dr. Henning Meyerhenke

Advisor: Elisabetta Bergamini

31st May 2016

Abstract

Zusammenfassung

Contents

List of Figures

List of Tables

1. Introduction

The concept of a *network* is used as a tool to model interactions and connections in several fields of science. "The scientific study of networks [...] is an interdisciplinary field that combines ideas from mathematics, physics, biology, computer science, the social sciences and many other areas", Mark Newman writes in his book *Networks. An introduction* [14].

One fundamental concept in network analysis is *Centrality*. Centrality measures are used to ascertain the importance of nodes within a graph. Practical applications include finding the most influential people in a social network [12], identifying key infrastructure nodes in computer networks, analyzing the effects of human land use to organism movement [10], or finding so-called super-spreaders in disease transmission networks [7].

Perhaps one of the most widely-known centrality measures is *PageRank*, proposed by the founders of Google, Sergey Brin and Larry Page [15]. The idea behind PageRank is that a webpage is important if it is linked to by other important webpages. Instead of simply relying on the absolute number of links to a webpage (a metric that was often used in citation networks), the quality of the links is taken into account. PageRank is similar to Eigenvector centrality. At the time, PageRank was a considerable improvement over other search engines.

Another centrality measure is called Betweenness Centrality. It is based on counting the number of shortest paths between each pair of nodes in the graph. For instance, this can be useful to find the routers in a computer network that are most integral to its stability.

Closeness centrality

Closeness centrality is based on the intuition, first presented by Alexander Bavelas in 1950 [3], that a node is important if its distance to other nodes in the graph is small. There are different definitions of closeness centrality that are applicable in different contexts. For strongly connected graphs, one can simply compute the sum of the distances from one node to all the other nodes. However, this approach leads to problems on disconnected graphs. It is not initially clear how to treat nodes which are not connected by a path in the graph. Simply assuming an infinite (or arbitrarily large) distance would completely distort the resulting closeness values of affected nodes. One possible solution for this problem is to only take into account the distances of reachable nodes, then scaling the result with the number of reachable nodes. Another approach is called *harmonic centrality*. Instead of summing all distances and then computing the inverse, the harmonic centrality is obtained by computing the sum of the inverse distances between nodes. Disconnected node pairs do not contribute to the total sum of inverse distances.

Computing the closeness centrality of a node in an unweighted graph requires a complete breadth-first search (BFS), and a complete run of Dijkstra's algorithm [8] with weighted graphs. It requires solving the *all-pairs-shortest-path* problem to compute the closeness centrality of each node in the graph. The computational effort for this is often impractical,

especially on large real-world networks. Moreover, this effort cannot be avoided if the application requires an exact ranking of all nodes by their closeness centrality.

Cite survey of different centrality measures

For some applications, however, it is enough to compute a list of the k most central nodes. This problem is called *Top-k closeness*. Limiting the problem to the k most central nodes can decrease the required computational effort significantly. It is only necessary to compute the exact closeness centrality of the k most central nodes to rank them. For all other nodes, it is sufficient to obtain an upper bound for their closeness that is smaller than the exact closeness of the k-th most central node.

Dynamic Top-k closeness centrality

In some cases, it is enough to compute closeness centralities only once because the underlying graph is static. Now consider a social network which constantly adds new users, which is effectively a node insertion in the underlying graph, and existing users befriend other users, which is an edge insertion. Terminating a friendship in the social network corresponds to an edge removal.

Each modification of a graph affects at least the closeness centralities of the directly affected nodes, that is, the nodes incident to a newly inserted or removed edge. It is also possible that there are new shortest paths between pairs of nodes that use the newly inserted edge. Analogously, removing an edge might increase the distance between node pairs because there was only one shortest path between them and it contained the removed edge. A simple strategy to get the new closeness centralities of each node is to re-run the static algorithm on the modified graph, ignoring any information collected by previous runs of the algorithm.

Group closeness

The concept of closeness centralities for single nodes can be extended to groups of nodes. The distance between a node v and a group S is defined as the smallest distance between v and any node of the group. The problem to find a group of size k such that the total distance of all nodes in the graph to the group is minimal is called the *maximum closeness centrality group identification* (MCGI) problem by Chen et al. in [6]. Since the problem is shown to be NP-hard, no efficient exact algorithm exists at this point. However, there are approximative greedy algorithms [6, 17] for the problem. Bergamini et al. improve on the work in [6] by reducing the memory requirements and total number of operations .

Cite unpublished work

Contributions

This thesis contributes a dynamic algorithm for Top-k closeness that handles both edge insertions and edge removals. It is based on the static algorithm first proposed by Borassi et al for complex networks in [5], and the additional optimizations for street networks proposed by Bergamini et al. in [4]. Our algorithm re-uses information obtained by an initial run of the static algorithm and tries to skip the re-computation of closeness centralities for nodes that are unaffected by modifications of the graph. In some cases,

even the upper bounds for the closeness centralities of affected nodes can be updated with little computational effort.

We also contribute an algorithm to update the group of nodes with the highest group closeness after an edge insertion. It is based on Bergamini and colleagues' improved version of the algorithm by Chen et al. The basic idea is to verify whether the choices of the greedy algorithm are still valid on the modified graph with as little computational expense as possible. Once the greedy algorithm would choose a different node than on the previous graph, the dynamic algorithm discards all information from the previous run and falls back to the static algorithm.

2. Preliminaries

In this thesis, $G = (V, E)$ usually denotes a simple, unweighted graph with a set of nodes V and a set of edges E between the nodes. An undirected edge is a subset of V with exactly two distinct elements. A directed edge is an element of $V \times V$. The distance between two nodes u and v is denoted by $d(u, v)$. The set of neighbor nodes for v is denoted by $N(v) := \{u : \exists(v, u) \in E\}$.

We now want to define the concept of closeness centrality. The most simple definition, while only useful for connected graphs, is the following:

Definition 2.1. *Let $G = (V, E)$ be a connected, unweighted graph. The closeness centrality of a node $v \in V$ is defined as*

$$c(v) = \frac{|V| - 1}{\sum_{u \in V} d(v, u)}.$$

In the disconnected case, there are node pairs without a path between them. Using Definition 2.1 and assuming $d(u, v) = \infty$ for such node pairs, the sum over all the distances in the denominator would blow up and make the resulting closeness centralities useless. To solve this problem, we first define $R(v) \coloneqq \{u \in V : u \text{ is reachable from } v \text{ in } G\}$ and $r(v) \coloneqq |R(v)|$. This leads to a generalized version of Definition 2.1:

$$c(v) = \frac{r(v) - 1}{\sum_{u \in R(v)} d(v, u)}. \tag{2.1}$$

However, this definition does not differentiate between central nodes in small components and central nodes in large components of a graph. Intuitively, a node v with $r(v) = 2000$ and a total distance of 4000 to all reachable nodes is more central than a node w with $r(w) = 20$ and a total distance of 40. In order to give preference to nodes in large components, we scale Equation 2.1 by $\frac{r(v)}{|V|-1}$.

Definition 2.2. *Let $G = (V, E)$ be an unweighted graph. The closeness centrality of a node $v \in V$ is defined as*

$$c(v) = \frac{r(v) - 1}{\sum_{u \in R(v)} d(v, u)} \cdot \frac{r(v)}{n - 1}.$$

Another approach to handle disconnected graphs is called *harmonic centrality*.

Definition 2.3. *Let $G = (V, E)$ be an unweighted graph. The harmonic centrality of a node $v \in V$ is defined as*

$$h(v) = \sum_{u \in V} \frac{1}{d(v, u)}.$$

If there is no path between u and v, the inverse distance is set to 0.

In this thesis, we will sometimes use *harmonic centrality* and *closeness centrality* interchangeably.

3. Static Top-k closeness

Computing the exact closeness centrality of each node in a graph requires solving the *all-pair-shortest-path* (APSP) problem . A simple approach is to compute a breadth-first search (or Dijkstra's algorithm) from each node. This results in a time complexity of $\mathcal{O}(|V| \cdot (|V| + |E|))$ for unweighted graphs and $\mathcal{O}(|V| \cdot (|V| \log |V| + |E|))$ for weighted graphs. The Floyd-Warshall algorithm has a time complexity of $\mathcal{O}(n^3)$ [11]; Johnsons's algorithm for weighted graphs without negative cycles has a time complexity of $\mathcal{O}(|V| \cdot (|V| \log |V| + |E|))$ [13]. Recently, Akiba et al. proposed an algorithm for fast distance queries on large networks which utilizes pruned searches to precompute the distances of each node to a limited set of individual landmarks from which the exact distances can be computed [1]. This reduces the amount of memory required to store all the distances while still providing fast distance queries. However, these algorithms might still be unfeasible for large networks.

3.1 Efficient algorithms for closeness centrality

There have been some ideas to reduce the effort to compute closeness centralities in large networks.

3.1.1 Approximation of closeness centrality

Eppstein and Wang propose a fast approximation algorithm in [9] for large networks exhibiting the *small world phenomenon*. It provides an $(1 + \epsilon)$-approximation in near-linear time. The algorithm works as follows:

1. Let k be the number of iterations to obtain the desired error bound
2. In iteration i, pick vertex i uniformly at random from G and solve the SSSP problem with v_i as the source
3. Let
$$\hat{c}_u = \frac{1}{\sum_{i=1}^{k} \frac{n \cdot d \cdot (v_i, u)}{k \cdot (n-1)}}$$
be the centrality estimator for vertex u.

Eppstein and Wang show that the expected value of $\frac{1}{\hat{c}_u}$ is equal to $\frac{1}{c_u}$. Using Hoeffding's bound, they also show that for $k = \mathcal{O}(\frac{\log n}{\epsilon^2})$ the additive error ist at most $\Delta\epsilon$ with high probability. The total runtime of the algorithm is $\mathcal{O}\left(\frac{\log n}{\epsilon^2}(n \log n + m)\right)$ for weighted graphs. For unweighted graphs it is $\mathcal{O}\left(\frac{\log n}{\epsilon^2} \cdot (n + m)\right)$.

3.1.2 Top-k closeness centrality

For some real-world applications, it is unnecessary to know the exact closeness centrality and the exact ranking of unimportant nodes. In these cases, it might be enough to compute a list of the k nodes with the highest exact closeness centrality. For all the other nodes it is

enough to provide an upper bound that is lower than the known exact closeness centrality of the k-th most central node.

Borassi et al. propose an efficient algorithm for the problem in [5] which works especially well on complex networks. Angriman presents an adaptation of the algorithms to work with harmonic centrality [2]. Bergamini et al. present some modifications for street networks (i.e. graphs with large diameter) in [4]. Since the dynamic algorithm presented in this thesis is based on these algorithms, we will provide detailed descriptions and analysis. The following explanation is based on the version of the algorithm for harmonic centrality.

3.1.2.1 Upper bounds for the closeness centrality of a node

The algorithm by Borassi et al. is briefly outlined in Algorithm 3. The main idea is to run a BFS from each node in the graph, but abort the search once it is clear that the node does not belong to the k nodes with highest closeness. During the BFS from a specific node v, the algorithm keeps track of the upper bound $\widetilde{h}(v)$ for the harmonic closeness of that node.

For that purpose, the algorithm keeps track of the current level d of the search, that is the current distance to the source node v of the search. When the BFS reaches a new level, the upper bound $\widetilde{h}(v)$ is recomputed. If the new upper bound is smaller than the known exact harmonic centrality of the k-th node in the list, the search is aborted.

Let Γ_d denote the set of nodes on level d from v, γ_d the number of nodes on level d. For each level, the algorithm computes an upper bound $\widetilde{\gamma}_{d+1} = \sum_{u \in \Gamma_d} deg(u)$ for the number of nodes on level $d+1$. Let $r(v)$ denote the number of nodes reachable from v. Let n_d denote the number of nodes up to level d. Let $h_d(v)$ denote the harmonic closeness based on all nodes up to level d from v.

During the BFS, the algorithms sums up the inverse distances of visited nodes. After visiting all nodes on level d, the resulting sum is $h_d(v)$. For the upper bound, we start with

$$h(v) \leq h'(v) = h_d(v) + \frac{\gamma_{d+1}}{d+1} + \frac{r(v) - n_{d+1}}{d+2}. \tag{3.1}$$

Basically, all nodes on level $d+1$ contribute $\frac{1}{d+1}$ to the harmonic centrality. All remaining unvisited nodes have at least distance $d+2$.

Since $n_{d+1} = n_d + \gamma_{d+1}$, we can write

$$h'(v) = h_d(v) + \frac{\gamma_{d+1}}{d+1} + \frac{r(v) - n_d - \gamma_{d+1}}{d+2} \tag{3.2}$$

We can now replace γ_{d+1} with its upper bound $\widetilde{\gamma}_{d+1}$, which is computed after visiting all

nodes on level d.

$$h'(v) \leq h_d(v) + \frac{\widetilde{\gamma}_{d+1}}{d+1} + \frac{r(v) - n_d - \widetilde{\gamma}_{d+1}}{d+2} \tag{3.3}$$

$$= h_d(v) + \frac{(d+2) \cdot \widetilde{\gamma}_{d+1} + (d+1) \cdot (r(v) - n_d - \widetilde{\gamma}_{d+1})}{(d+1) \cdot (d+2)} \tag{3.4}$$

$$= h_d(v) + \frac{\widetilde{\gamma}_{d+1} + (d+1) \cdot (r(v) - n_d)}{(d+1) \cdot (d+2)} \tag{3.5}$$

$$\widetilde{h}(v) = h_d(v) + \frac{\widetilde{\gamma}_{d+1}}{(d+1) \cdot (d+2)} + \frac{r(v) - n_d}{d+2} \tag{3.6}$$

3.1.2.2 Computing the number of reachable nodes

The number of reachable nodes $r(v)$ in Equation 3.6 can be computed in a preprocessing step. For undirected graphs, the number of reachable nodes from v is equal to the size of the connected component containing v. Algorithm 1 computes the number of reachable nodes for each node in $\mathcal{O}(n+m)$ with a single BFS.

Directed graphs

For directed graphs, $r(v)$ cannot be computes as easily if the graph is not strongly-connected. Instead, an upper bound for $r(v)$ is computed. The strongly-connected components of a graph G can be computed in linear time with Tarjan's algorithm [16]. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote the graph of the strongly-connected components of G, with

- $\mathcal{V}$ as the set of strongly-connected components in G
- $C \subset V$ and $D \subset V$ denoting strongly-connected components
- $(C, D) \in \mathcal{E} \iff \exists (u, v) \in E,$ such that $u \in C \wedge v \in D$
- the weight $w(C) = |C|$ for each strongly-connected component C.

Tarjan's algorithm already provides a topologically sorted list of the strongly-connected components of G. Components without any outgoing edges, so-called *sinks*, are placed at the end of the list. The list is then processed in reverse order and an upper bound $\omega(C)$ for the number of reachable nodes from that component can be computed with

$$\omega(C) = w(C) + \sum_{(C,D) \in \mathcal{E}} \omega(D).$$

Since the components are traversed in reverse topological order, the upper bound of each neighbor component is already known. Please note that there are no circles in this graph of strongly-connected components. Otherwise, all components comprising the circle would belong to one larger strongly-connected components.

3.1.2.3 Pruned breadth-first search

In order to compute the closeness centrality of a single node in an unweighted graph, a complete breadth-first search starting from that node is necessary. The algorithm only

```
   Data: G = (V, E)
   Result: r(v) for each node v
 1 P(v) ← none for each node v
 2 i ← 0
 3 forall v ∈ V do
 4 |   if P(v) = none then
 5 |   |   Run BFS from v and set P(u) = i for each visited node u, keep track of the
   |   |     component sizes
 6 |   |   i ← i + 1
 7 |   end
 8 end
 9 forall v ∈ V do
10 |   r(v) ← componentSizes[P(v)]
11 end
```

Algorithm 1: Computing connected components in an undirected graph

requires the exact closeness centrality of the k most central nodes. Therefore, the breadth-first searches for many nodes can be aborted early, once the upper bound obtained with Equation 3.6 is smaller than the exact closeness centrality of the k-th most central node among the already processed nodes. Algorithm 2 outlines the steps necessary to perform such a pruned BFS. The algorithm is structured like a standard BFS, but also keeps track of the harmonic closeness of the source node. The algorithm sums the inverse distances of all visited nodes in h (Line 22). Once the BFS reaches a new level (Line 7), $\widetilde{h}$ is updated (Line 10). If the new upper bound is smaller than the known exact closeness x_k of the k-th node, the algorithm returns with the last computed upper bound and a flag that indicates that the value is not exact. The algorithm also keeps track of an upper bound $\widetilde{\gamma}$ for the number of nodes on level $d+1$ by summing the out-degrees of all nodes on level d (Line 23). If the search encounters a node w that has already been marked as visited, that node must be on a level $l < d$. It is then possible to compute a new, lower upper bound $\widetilde{h}$ in some cases (Line 24).

3.1.2.4 Algorithm outline

Algorithm 3 is used to compute the list of the k nodes with the highest closeness in G. The preprocessing in Line 1 is used to compute the number of reachable nodes for each node in the graph in linear time (see Section 3.1.2.2). The algorithm then iterates over all nodes in decreasing order of degree (Line 5) and starts a pruned BFS from each node. If the pruned BFS returns an exact closeness value, and if the value is larger than the current k-th largest value, it can be added to the list with the most central nodes. x_k, representing the k-th largest value, can then also be updated.

3.1.2.5 Networks with large diameter

The previously described algorithm works well for complex (social) networks with small diameter. Bergamini et al. present a different approach to solve the problem faster on networks with large diameter, for instance street networks [4]. The basic idea is to always run a complete breadth-first search from a source node and then use the number of nodes

Data: $G = (V, E), v, x_k$
Result: A tuple $(h, isExact)$ with $isExact$ =`false` if h is only an upper bound for the exact harmonic closeness

Create queue Q
Q.enqueue(v)
Mark v as visisted
$d \leftarrow 0; h \leftarrow 0; \widetilde{\gamma} \leftarrow 0; nd \leftarrow 0$
while *!Q.isEmpty* **do**
  $u \leftarrow$ Q.dequeue()
  if $d(v,u) > d$ **then**
    $d \leftarrow d + 1$
    $r \leftarrow r(u)$
    $\widetilde{h} \leftarrow h + \frac{\widetilde{\gamma}}{(d+1)\cdot(d+2)} + \frac{r - n_d}{d+2}$
    if $\widetilde{h} \leq x_k$ **then**
      return $(\widetilde{h}$, `false`$)$
    end
  end
  forall $w \in N(u)$ **do**
    if *w is not marked as visited* **then**
      Mark w as visited
      Q.enqueue(w)
      $n_d \leftarrow n_d + 1$
      $pred[w] \leftarrow u$
      $d(v,w) \leftarrow d(v,u) + 1$
      $h \leftarrow h + \frac{1}{d(v,w)}$
      $\widetilde{\gamma} \leftarrow \widetilde{\gamma} + outdegree(w) - 1 \cdot G.isDirected()$
    else if $d(v,w) > 1 \wedge pred[u] \neq w$ **then**
      $\widetilde{h} \leftarrow \widetilde{h} - \frac{1}{d+1} + \frac{1}{d+2}$
      if $\widetilde{h} \leq x_k$ **then**
        return $(\widetilde{h}$, `false`$)$
      end
  end
  return $(h$, `true`$)$
end

Algorithm 2: BFScut()

on each level to compute an upper bound for the closeness centrality of all the other nodes in the graph. The nodes are kept in a list sorted in decreasing order by the corresponding upper bound for their closeness centrality.

Level-based upper bounds

It is possible to compute upper bounds for the closeness centrality of all nodes in a graph with a single BFS. Let G denote an undirected graph and s the source node of the BFS. A node v is on level i ($l(v) = i$) if $d(s,v) = i$ and we write $v \in \Gamma_i(s) \iff d(s,v) = i$ (see Section 3.1.2.1). The distance between two arbitrary nodes $v \in \Gamma_i(s)$ and $w \in \Gamma_j(s)$ for $i \leq j$ is at least $j - i$. If $d(v,w)$ was smaller than $j - i$, w would have been discovered earlier and its level would be $i + d(v,w) < j$. This is a contradiction to the assumption

```
Data: G = (V, E)
Result: A list with the k nodes with the highest closeness
Preprocessing(G)
Top ← empty priority queue h(v) ← 0 for each node v
isExact(v) ← false for each node v
x_k ← 0
forall v ∈ V in decreasing order of degree do
    (h, isExact) ← BFSCut(v, x_k)
    h(v) ← h
    isExact(v) ← isExact
    if isExact ∧ h > x_k then
        Top.insert(h, v)
        if Top.size() > k then
            Top.removeMin()
        end
        x_k ← Top.getMin()
    end
end
```

Algorithm 3: Static computation of the k nodes with the highest closeness

that w is on level j. With this discovery, it is possible to obtain an upper bound for the harmonic closeness of each node $v \in V$:

$$h(v) \leq \widetilde{h}(v) = \sum_{w \in R(s)} \left| \frac{1}{d(s,w)} - \frac{1}{d(s,v)} \right|.$$

This bound can be improved further. The degree of a node v is equal to the number of nodes w at distance 1. All the other nodes with $\left| \frac{1}{d(s,w)} - \frac{1}{d(s,v)} \right| \leq 1$ must have at least distance 2. This leads to

$$\begin{aligned} \widetilde{h}(v) = 1 \cdot deg(v) \\ + \frac{1}{2} \cdot (|\{w \in R(s) : |d(s,w) - d(s,v)| \leq 1\}| - deg(v) - 1) \\ + \sum_{\substack{w \in R(s) \\ |d(s,w)-d(s,v)|>1}} |\frac{1}{d(s,w)} - \frac{1}{d(s,v)}|. \end{aligned} \tag{3.7}$$

After the BFS, the number of nodes γ_j on each level j is known. With that information, Equation 3.7 can be rewritten to

$$\widetilde{h}(v) = \frac{1}{2} \cdot \left(\sum_{|j-d(s,v)| \leq 1} \gamma_j \right) + \left(\sum_{|j-d(s,v)|>1} \gamma_j \cdot \left| \frac{1}{j - d(s,v)} \right| \right) - \frac{1}{2} + \frac{1}{2} \cdot deg(v). \tag{3.8}$$

For all nodes v with the same distance from s, the first three terms of the equation are the same. Therefore, a preliminary *level bound* can be computed once for each level. For each individual node v, only $\frac{1}{2} \cdot deg(v)$ needs to be added to get $\widetilde{h}(v)$.

For directed graphs, Equation 3.7 does not hold. Consider two nodes v and w with $l(w) < l(v)$. In the undirected case, it is possible to infer a lower bound for the distance

between the two nodes. This is not possible in the directed case because there could be a shortcut between v and w as shown in Figure 3.1. For all nodes w with $l(w) < l(v)$ and $w \notin N(v)$, we can only assume that the distance must be at least 2. This leads to slightly less tight upper bounds for directed graphs. Modifying Equation 3.7 accordingly leads to

$$\begin{aligned}\widetilde{h}_{directed}(v) = &\frac{1}{2} \cdot deg(v) \\ &+ \frac{1}{2} \cdot (|\{w \in R(s) : d(s,w) - d(s,v) \leq 1\}|) \\ &+ \sum_{\substack{w \in R(s) \\ d(s,w)-d(s,v)>1}} \frac{1}{d(s,w)} - \frac{1}{d(s,v)}. \end{aligned} \tag{3.9}$$

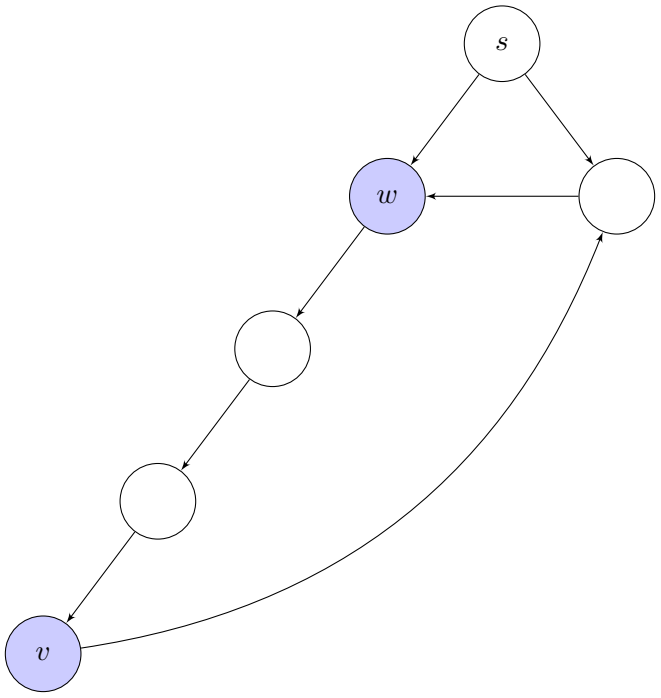

Figure 3.1: BFS tree
The distance between w and v is 3 in the BFS tree from s, while the actual distance is 2.

Computing level-based bounds

The level bound is

$$\begin{aligned} L(i) &= \frac{1}{2} \cdot \sum_{|j-i| \leq 1} \gamma_j + \sum_{|j-i|>1} \gamma_j \cdot \frac{1}{|j-i|} - \frac{1}{2} \\ &= \frac{1}{2} \cdot (\gamma_{i-1} + \gamma_i + \gamma_{i+1}) + \sum_{|j-i|>1} \gamma_j \cdot \frac{1}{|j-i|} - \frac{1}{2} \end{aligned} \tag{3.10}$$

in the undirected case. This difference between $L(i)$ and $L(i-1)$ is

$$\begin{aligned} L(i) - L(i-1) &= \frac{1}{2} \cdot (\gamma_{i-1} + \gamma_i + \gamma_{i+1}) + \sum_{|j-i|>1} \gamma_j \cdot \frac{1}{|j-i|} \\ &\quad - \frac{1}{2} \cdot (\gamma_{i-2} + \gamma_{i-1} + \gamma_i) + \sum_{|j-i+1|>1} \gamma_j \cdot \frac{1}{|j-i+1|} \\ &= \frac{1}{2} \cdot (\gamma_{i+1} - \gamma_{i-2}) + \frac{1}{2} \cdot \gamma_{i-2} - \frac{1}{2} \cdot \gamma_{i+1} \\ &\quad + \sum_{j<i-2 \cup j>i+1} \left(\frac{1}{|j-i|} - \frac{1}{|j-i+1|} \right) \cdot \gamma_j \\ &= \sum_{j<i-2 \cup j>i+1} \left(\frac{1}{|j-i|} - \frac{1}{|j-i+1|} \right) \cdot \gamma_j \end{aligned} \tag{3.11}$$

Note: Obviously, this does not lead to a simplification in the harmonic closeness case. We either have to keep this part and rewrite it as a proof that the optimization with the prefix sum does not work, or we don't keep it because it's not important in practice anyway.

Data: $G = (V, E)$
Result: A list with the k nodes with the highest closeness

```
1  Preprocessing(G)
2  Q ← V, sorted by decreasing degree or a precomputed upper bound
3  Top ← []
4  score ← array indexed by node ID storing the current upper bounds for all nodes
5  for v ∈ V do
6  |  score[v] ← ∞
7  end
8  while Q is not empty do
9  |  v ← Q.extractMax()
10 |  if |Top| ≥ k∧score[v]≤ Top[k] then
11 |  |  return Top
12 |  end
13 |  score[v] ← updateBounds(v)
14 |  add v to Top
15 |  sort Top by score and reduce it to at most k elements
16 |  update Q according to the new values in score
17 end
```

Algorithm 4: Static computation of the k nodes with the highest closeness in networks with large diameter

Data: $G = (V, E), s \in V$
Result: The exact closeness centrality of s, upper bounds for the closeness of all other nodes

```
1  d ← BFSfrom(s)
2  maxD ← max_{v∈V} d(s, v)
3  sumΓ_{≤0} ← 0; sumΓ_{≤−1} ← 0; sumΓ_{>maxD+1} ← 0;
4  for i = 0 to maxD do
5  |  Γ_i ← {w ∈ V : d(s, w) = i}
6  |  γ_i ← |Γ_i|
7  |  sumΓ_{≤i} ←sumΓ_{≤i−1} + γ_i
8  |  sumΓ_{>i} ← |V| − sumΓ_{≤i}
9  end
10 L(1) ← γ_i + 1/2 · γ_2
```

Algorithm 5: Incomplete, does not reflect the implementation for the harmonic closeness case. The `updateBounds` function computes the exact closeness centrality of the supplied source node s and provides upper bounds for the closeness centrality of all other nodes in the graph

4. Conclusion

5. Declaration

Ich versichere hiermit wahrheitsgemäß, die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des Karlsruher Instituts für Technologie (KIT) zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 31. Mai 2015 _____

Patrick Bisenius

Bibliography

[1] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 349–360. ACM, 2013.

[2] Eugenio Angriman. Efficient computation of harmonic centrality on large networks: theory and practice. 2016.

[3] Alex Bavelas. Communication patterns in task-oriented groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950.

[4] Elisabetta Bergamini, Michele Borassi, Pierluigi Crescenzi, Andrea Marino, and Henning Meyerhenke. Computing top-k closeness centrality faster in unweighted graphs. In *2016 Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 68–80. SIAM, 2016.

[5] Michele Borassi, Pierluigi Crescenzi, and Andrea Marino. Fast and simple computation of top-k closeness centralities. *arXiv preprint arXiv:1507.01490*, 2015.

[6] Chen Chen, Wei Wang, and Xiaoyang Wang. *Efficient Maximum Closeness Centrality Group Identification*, pages 43–55. Springer International Publishing, Cham, 2016.

[7] AH Dekker. Network centrality and super-spreaders in infectious disease epidemiology.

[8] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[9] David Eppstein and Joseph Wang. Fast approximation of centrality. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 228–229. Society for Industrial and Applied Mathematics, 2001.

[10] Ernesto Estrada and Örjan Bodin. Using network centrality measures to manage landscape connectivity. *Ecological Applications*, 18(7):1810–1825, 2008.

[11] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962.

[12] Linton C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215 – 239, 1978.

[13] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, January 1977.

[14] Mark Newman. Networks: an introduction. 2010. *United Slates: Oxford University Press Inc., New York*, pages 1–2.

[15] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[16] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[17] Junzhou Zhao, John Lui, Don Towsley, and Xiaohong Guan. Measuring and maximizing group closeness centrality over disk-resident graphs. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 689–694. ACM, 2014.

Todo list