Distributed Identification of Central Nodes with Less Communication

Jordan F. Masakuna (jordan@aims.ac.za)
University of Kinshasa
Pierre K. Kafunda
University of Kinshasa

Research Article

Keywords: distributed systems, network analysis, closeness centrality, leader election

Posted Date: May 6th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1622718/v1

RESEARCH

Distributed Identification of Central Nodes with Less Communication

Jordan F. Masakuna[1,2*] and Pierre K. Kafunda[1]

*Correspondence: jordan@aims.ac.za

[1]Division of Computer Science, University of Kinshasa, Kinshasa, Dem. Rep. of Congo

Full list of author information is available at the end of the article

Abstract

This paper is concerned with distributed detection of central nodes in complex networks using closeness centrality. Closeness centrality plays an essential role in network analysis. Evaluating closeness centrality exactly requires complete knowledge of the network; for large networks, this may be inefficient, so closeness centrality should be approximated. Distributed tasks such as leader election can make effective use of centrality information for highly central nodes, but complete network information is not locally available. This paper refines a distributed centrality computation algorithm by You et al. [24] by pruning nodes which are almost certainly not most central. For example, in a large network, leave nodes can not play a central role. This leads to a reduction in the number of messages exchanged to determine the centrality of the remaining nodes. Our results show that our approach reduces the number of messages for networks which contain many prunable nodes. Our results also show that reducing the number of messages may have a positive impact on running time and memory size.

Keywords: distributed systems; network analysis; closeness centrality; leader election

1 Introduction

Centrality metrics play an essential role in network analysis [11]. For some types of centrality metrics such as betweenness centrality, evaluating network centrality exactly requires complete knowledge of the network; for large networks, this may be too costly computationally, so approximate methods (i.e. methods for building a view of a network to compute centrality) have been proposed. Centrality information for highly central nodes can be used effectively for distributed tasks such as leader election, but distributed nodes do not have complete network information. The relative centrality of nodes is important in problems such as selection of informative nodes, for example, in active sensing

[17]—i.e. the propagation time required to synchronize the nodes of a network can be minimized if the most central node is known [10, 18]. Here we consider closeness centrality—We wish to detect most central nodes in the network.

Not all nodes ultimately need to build a view of the network formed in their interaction since some nodes may realize quickly that they are not suitably central, i.e. they have small closeness centralities. The question is how to identify such insignificant nodes in a decentralised algorithm? This paper tackles this problem by introducing a pruning strategy which reduces the number of messages exchanged between nodes compared to the algorithm from [24]. When a leader is chosen based on closeness centrality, we observe that in the algorithm of [24], even nodes which can not play a central role, for example leaves, overload communication by receiving messages.[1]

This work proposes modifications to You et al's decentralised method to construct a view of a communication graph for distributed computation of node closeness centrality. Since we consider an approximate and decentralised method to construct a view of a communication graph, inevitably nodes will construct different topologies describing their interaction network. We will use the term view as shorthand for view of a communication network.

Our proposed method can be applied to arbitrary distributed networks, and is most likely to be valuable when nodes form very large networks: reducing the number of messages will be a more pressing concern in large-scale networks. Such applications include instrumented cars, monitoring systems, mobile sensor networks or general mobile ad-hoc networks. There are many reasons for reducing the number of messages in distributed systems. Here, we consider the following reason. A careful treatment of network communication of agents under weak signal conditions is crucial [22].

Contributions. At each iteration of view construction, each node prunes some nodes in its neighbourhood once and thereafter interacts only with the unpruned nodes to construct its view. We refer to this approach as pruning. The more of these prunable nodes a network contains the better our algorithm performs relative to the algorithm in [24] in terms of number of messages. We empirically evaluate our approach on a number of benchmark networks from [12] and [13], as well as some randomly generated networks, and observe our method outperforms the benchmark method [24] in terms of number of messages exchanged during

[1]Except in special cases, a leaf node should not play a central role.

interaction. We also observe some positive impact that pruning has on running time and memory usage.

We make the following assumptions as in [24]:

- nodes are uniquely identifiable;
- a node knows identifiers of its neighbours;
- communication is bidirectional, FIFO and asynchronous;
- each agent is equipped with its own round counter.

The rest of the paper is organised as follows: Section 2 discusses the state of the art for decentralised computation of closeness centrality distribution of networks. The new algorithm will be discussed in Section 3. Section 4 discusses the results. In Section 5, we conclude and propose further work.

2 Background and related work

In a distributed system, the process of building a view of the network can be centralised or decentralised. It is centralised when the construction of the view of a network is performed by a single node, known as an initiator. Once the initiator has the view of a network, it may send it to the other nodes [16]. If the initiator is not known in advance, the first stage of constructing a view of a network will involve a selection of the initiator.

Decentralised methods to construct views can be exact or approximate. Recent literature on methods to construct views can be found in [16]. Exhaustive methods build the complete topology of the communication graph—they involve an all-pairs shortest path computation. As a consequence, exact approaches suffer from problems of scalability [16]. This can be a particular issue for large networks of nodes with constrained computational power and restricted memory resources.

To overcome the problem mentioned above that exhaustive methods suffer from, approximate methods have been proposed. Unlike exhaustive methods, approximate methods do not result complete knowledge of the communication graph.

Many distributed methods for view construction are centralised, i.e. they require a single initiator in the process of view construction. The disadvantage of approximate methods is that the structure of a view depends on the choice of the initiator. An interesting distributed approach for view construction can be found in [10], where an initiator constructs a tree as the view.

To the best of our knowledge and according to [16], decentralised approximate methods for view construction are very scarce because many methods for view construction assume some prior information about the network, so centralised methods are more appropriate. The method proposed by You et al. [24] seems to be the state of the art for decentralised approximate methods for view construction, i.e. views are constructed only from local interactions. This method simply runs breadth-first search [20] on each node.

We next show the decentralised construction of a view using the algorithm in [24].

2.1 Decentralised view construction

We consider the method proposed in You et al. [24]—the "YTQ method"—as the state of the art for decentralised construction of a view of a network. The YTQ method was proposed for decentralised approximate computation of centrality measures (closeness, degree and betweenness centralities) of nodes in arbitrary networks. As treated in [24], these computations require a limited view. At the end of the interaction between nodes, each node can estimate its centrality based on its own view. In the following, we show how nodes construct views using the YTQ method. Here we consider connected and unweighted graphs, and we are interested in computation of closeness centrality.

Let δ_{ij} denote the path distance between the nodes v_i and v_j in an (unweighted) graph G with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. The path distance between two nodes is the length of the shortest path between these nodes.

Definition 2.1 *The closeness centrality [2] of a node is the reciprocal of the average path distance from the node to all other nodes. Mathematically, the closeness centrality c_i is given by*

$$c_i = \frac{|\mathcal{V}| - 1}{\sum_j \delta_{ij}}. \tag{1}$$

Nodes with high closeness centrality score have short average path distances to all other nodes.

Each node's view is gradually constructed based on message passing. Each node sends its neighbour information to all of its immediate neighbours which relay it onward through the network. Communication between nodes is asynchronous, i.e. there is no common clock signal between the sender and receiver.

The YTQ method [24] that each node v_i uses to construct its view is given in Algorithm 1.

Let $\mathcal{N}_i$ be the set of neighbours of v_i and $\mathcal{N}_i^{(t)}$ the set of nodes at distance $t+1$ from a node v_i, so $\mathcal{N}_i^{(0)} = \mathcal{N}_i$. The initial set of neighbours, $\mathcal{N}_i$, is assumed to be known.

During each iteration, each node sends its neighbourhood information to all its immediate neighbours. We are restricted to peer-to-peer communication because nodes have limited communication capacity. Each node waits for communication from all of its direct neighbours after which it updates an internal round counter. A node v_i stores messages received in a queue, represented by $\mathcal{M}_i$. After round $t \geq 1$, the topology of v_i's view is updated as follows

$$\mathcal{N}_i^{(t)} \leftarrow \bigcup_{v_j \in \mathcal{N}_i} \mathcal{N}_j^{(t-1)} \setminus \mathcal{N}_{i,t}\,, \tag{2}$$

where

$$\mathcal{N}_{i,t} \leftarrow \bigcup_{k=0}^{t-1} \mathcal{N}_i^{(k)}\,. \tag{3}$$

The algorithm terminates after at most D iterations, where D is an input of the algorithm.[2] In this paper, we consider a pre-set value of D. However, some nodes can also reach their equilibrium stage before the iteration D (e.g. nodes which are more central than others). Such nodes need to terminate when equilibrium is reached (see Line 57 in Algorithm 1). A node v_i reaches equilibrium at iteration t when

$$\mathcal{N}_i^{(t)} = \emptyset\,.$$

At the end, every node has a view of network, and so the required centralities can be calculated locally. This view construction method is approximate when the total number of iterations D is less than the diameter of the graph, otherwise the method is exhaustive, i.e. all views correspond to the exact correct information, assuming a failure-free scenario. With a decentralised approximate method, nodes may have different views at the end. Each node will evaluate its closeness centrality based on its own view of the network.

3 Decentralised view construction

The idea behind pruning technique is that, during view construction, some nodes can be pruned (i.e. some nodes will stop relaying neighbour information). Pruned

[2]It can also be set or determined in a distributed manner (i.e. the value of D can be determined by nodes during interaction) as in [6].

nodes are not involved in subsequent steps of the algorithm and their closeness centralities are treated as zero.

Our approach thus applies pruning after each iteration t of communication of the YTQ method. During the pruning stage, each node checks whether it or any nodes in its one-hop neighbourhood should be pruned. Nodes can identify the other nodes in their neighbourhood being pruned so that they do not need to wait for or send messages to them in subsequent iterations. This reduces the number of messages exchanged between nodes.

Given two direct neighbours, their sets of pruned nodes are not necessarily the same, and nodes do not need to exchange such information between themselves.

3.1 Pruning

Before describing our proposed pruning method, we argue that pruning preserves information of most central nodes of a graph using closeness centrality.

3.1.1 Theoretical justification

While we are aiming to estimate closeness centrality distribution on a graph using pruning, the concept of pruning can directly be related to eccentricity centrality [8]. The eccentricity of a node is the maximum distance between the node and another node. Eccentricity and eccentricity centrality are reciprocal to each other. We consider the following points to achieve our goal.

- Pruned nodes have relatively high eccentricities (as will be discussed later in Lemma 3.2).
- Previous studies show that eccentricity and closeness centralities are strongly positively correlated for various types of graphs [1, 15]. This is partly due to the fact that they both operate on the concepts of paths.

From what precedes, our proposed pruning method is then recommended for approximations of closeness centralities for categories of graphs where eccentricity and closeness centralities are highly correlated.

3.1.2 Description

We will first show how a node identifies prunable nodes after each iteration. There are two types of objects (leaves, and nodes causing triangles) that a node can prune after the first iteration. When a node is found to be of one of these types, it is pruned. Since nodes have learnt about their 2-hop neighbours after the end of the first iteration, a node v_i knows the neighbours $\mathcal{N}_j$ of each of its direct neighbours v_j. Our pruning method is decentralised, so each node is responsible to identify

prunable nodes in its neighbourhood, including itself. Let d_i denote the degree of node v_i.

Definition 3.1 (A node causing a triangle) *A non-leaf node v_j causes a triangle if $d_j = 2$ and its two immediate neighbours are immediate neighbours to each other.*

Let $\mathcal{F}_i^{(t)}$ denote the set of pruned nodes known by a node v_i at the end of iteration t (with $t \geq 1$).

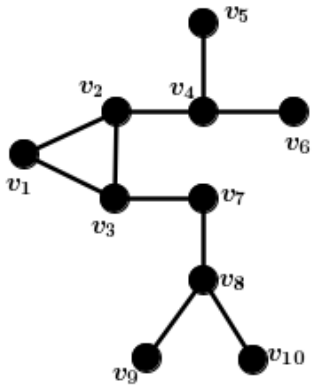


Figure 1: *Illustration of graphs with prunable nodes.*

Let $\mathcal{N}_{i,t}^{\text{up}} = \mathcal{N}_i \setminus \mathcal{F}_i^{(t-1)}$ denote the set of neighbours of v_i which have not yet been pruned at the beginning of iteration t. Functions that a node v_i applies to detect prunable elements in its neighbourhood after the first iteration are given in Algorithm 2 (see functions LEAVESDETECTION and TRIANGLEDETECTION).

Note that for complete graphs, pruning is not involved because, after the first iteration each node will realise that it its current view of the graph is complete.

So far, we have described how elements of $\mathcal{F}_i^{(1)}$ are identified by node v_i. We now consider the case of further pruning which is straightforward: new nodes should be pruned when they have no new information to share with their other active neighbours. Thus a node stops relaying neighbouring information to neighbours from which it receives no new information.

At the end of iteration t, a node v_i considers itself as element of $\mathcal{F}_i^{(t)}$ if it gets all its new information from only one of its neighbours at that iteration. Also, node v_i prunes v_j if the neighbouring information $\mathcal{N}_j^{(t)}$ sent by v_j to v_i does not contain

new information, i.e.

$$\mathcal{N}_j^{(t)} \subseteq \bigcup_{l<t} \mathcal{N}_i^{(l)} . \tag{4}$$

Our hope is that the most central node is among the nodes which are not pruned on termination of the algorithm. Equation 4 indicates for a node v_j to be pruned, there must be another node v_i which is unpruned because a comparison needs to be done. This is true because there are always unpruned nodes which remain after the first iteration, except a complete graph of at most three nodes in which case pruning is not invoked. This means that at the end of our pruning method, there will always remain some unpruned nodes.

Nodes in $\mathcal{F}_i^{(t)}$ for $t \geq 2$ could be viewed as leaves or nodes causing triangles in the subgraph obtained after the removal of all previously pruned nodes. Recall that an unpruned node only interacts with its unpruned neighbours and the number of the unpruned neighbours of a node may get reduced over iterations. So at some iteration an unpruned node can be viewed as leaf if it remains only with one unpruned neighbour.

Let

$$\mathcal{F}_{i,t} = \bigcup_{l=1}^{t} \mathcal{F}_i^{(l)} .$$

The procedure for how a node v_i detects elements of $\mathcal{F}_i^{(t)}$ at the end of each iteration $t \geq 2$ is given in Algorithm 2 (see function FURTHERPRUNINGDETECTION).

After describing pruning, we now connect it to eccentricity (see Lemma 3.2) as mentioned above. Let ecc_i denote the eccentricity of a node v_i, i.e.

$$\mathrm{ecc}_i = \max_{v_j \in \mathcal{V}} \delta_{ij} . \tag{5}$$

Lemma 3.2 *If $v_j \in \mathcal{N}_i$ such that*

$$v_j \in \bigcup_{t \geq 1} \mathcal{F}_i^{(t)} ,$$

then

$$\mathrm{ecc}_j \geq \mathrm{ecc}_i .$$

Proof A node v_j is pruned if it has a direct neighbour v_i from which it can receive new information while at the same time it can not provide new information to that

neighbour. Given two direct neighbours v_j and v_i where v_j has been pruned at iteration t by v_i, we have Equation 4. From the definition of $\mathcal{N}_i^{(t)}$ and Equation 4, it is straightforward that $\text{ecc}_j \geq \text{ecc}_i$. □

From Lemma 3.2 it can be seen that prunable nodes are nodes with relatively high eccentricities. So pruning can not introduce errors when searching for a node of maximum eccentricity centrality.

Example. Consider execution of Algorithm 2 on the communication graph in Figure 1, with $D = 4$. The results of our pruning method on this graph are presented in Table 1 and discussed below.

Node sets	ecc_i	$t=1$	$t=2$	$t=3$	$t=4$
$\mathcal{F}_1^{(t)}$	4	v_1	$\perp$	$\perp$	$\perp$
$\mathcal{F}_2^{(t)}$	4	v_1	v_4	v_2	$\perp$
$\mathcal{F}_3^{(t)}$	3	v_1	$\emptyset$	v_2, v_7	$\top$
$\mathcal{F}_4^{(t)}$	5	v_5, v_6	v_4	$\perp$	$\perp$
$\mathcal{F}_5^{(t)}$	6	v_5	$\perp$	$\perp$	$\perp$
$\mathcal{F}_6^{(t)}$	6	v_6	$\perp$	$\perp$	$\perp$
$\mathcal{F}_7^{(t)}$	4	$\emptyset$	v_8	v_7	$\perp$
$\mathcal{F}_8^{(t)}$	5	v_9, v_{10}	v_8	$\perp$	$\perp$
$\mathcal{F}_9^{(t)}$	6	v_9	$\perp$	$\perp$	$\perp$
$\mathcal{F}_{10}^{(t)}$	6	v_{10}	$\perp$	$\perp$	$\perp$

Table 1: *Table indicating pruned nodes, identified at each node after each iteration using the graph in Figure 1. $\perp$ indicates that the corresponding node has pruned itself and $\top$ indicates that the node has reached an equilibrium.*

After the first iteration, each node needs to identify prunable nodes in its neighbourhood, including itself. Using Algorithm 2, $\mathcal{F}_1^{(1)} = \{v_1\}$, v_1 will prune itself. Also, $\mathcal{F}_2^{(1)} = \mathcal{F}_3^{(1)} = \{v_1\}$, v_2 and v_3 will also prune v_1. At the first iteration, all the leaves (i.e. v_5, v_6, v_9 and v_{10}) are pruned; they all have $\text{ecc}_i = 6$. But, a non-leaf node, v_1 (with $\text{ecc}_i = 4$), is also pruned on the first iteration.

At the beginning of iteration $t = 2$, v_1, v_5, v_6, v_9 and v_{10} are no longer involved since they have been identified as prunable nodes at the end of the previous iteration; $\mathcal{F}_2^{(2)} = \mathcal{F}_4^{(2)} = \{v_4\}$ and $\mathcal{F}_7^{(2)} = \mathcal{F}_8^{(2)} = \{v_8\}$; and the node v_3 does not identify any prunable node after this iteration. At this iteration, the remaining nodes with high ecc_i (i.e. nodes v_4 and v_8) are pruned. The results for the remaining iterations are shown in Table 1.

In our proposed distributed system, at the end of each iteration each node is aware of whether each of its direct neighbour is pruned or not). A pruned node neither sends a message nor waits for a message. So when a node is still unpruned, it knows which immediate neighbours to send messages to and which to wait

for messages from. This prevents the nodes from suffering from starvation or deadlock in failure-free scenarios [4].

3.2 Communication analysis

In this section, we evaluate the impact of pruning on the communication requirements for view construction. Let $u_i^{(t)}$ denote the number of neighbours of v_i which have been pruned at the end of iteration t. Let $Y_i^{(D)}$ and $P_i^{(D)}$ be the number of messages that the node v_i receives according to Algorithm 1 and the number of messages v_i receives through the use of pruning in Algorithm 2 for D rounds respectively. We expect the number of messages any node v_i saves due to pruning to satisfy

$$\Delta_i^{(D)} \equiv Y_i^{(D)} - P_i^{(D)} \geq 0\,. \tag{6}$$

A node v_i receives d_i messages at the end of each iteration using the YTQ method. Recall that our proposed pruning and the YTQ methods can also terminate when an equilibrium is reached. Let H_i denote the iteration after which a node v_i applying the YTQ and our pruning methods reaches an equilibrium. This value is the same for both algorithms because at iteration t, a node v_i (which should be an unpruned node using our proposed method) has the same view using both algorithms. Note that for our pruning method, a pruned node does not reach an equilibrium and it is not possible to prune all nodes before equilibrium. Let $h_i^{(t)}$ be the number of neighbours of v_i which have reached equilibrium at the end of iteration t. If at least one neighbour of an unpruned node v_i has reached equilibrium by iteration t, then v_i will reach equilibrium by iteration $t+1$. For the YTQ method,

$$Y_i^{(D)} = \sum_{t=1}^{\min(D,H_i)} \left(d_i - \sum_{l=0}^{t-1} h_i^{(l)} \right). \tag{7}$$

Let L_i denote the round at which a node v_i is pruned ($L_i = +\infty$ for unpruned node v_i).

Lemma 3.3 *The number of messages received by a node $v_i \in \mathcal{V}$ in Algorithm 2 is*

$$P_i^{(D)} = \sum_{t=1}^{\min(D,H_i,L_i)} \left(d_i - \sum_{l=0}^{t-1} \left(h_i^{(l)} + u_i^{(l)} \right) \right).$$

Proof At the end of each iteration t, the node v_i receives $\left(d_i - \sum_{l=0}^{t-1} \left(h_i^{(l)} + u_i^{(l)} \right) \right)$ messages. Note that $u_i^{(0)} = h_i^{(0)} = 0$. Also a node can stop interacting with

other nodes after it is pruned. If the node v_i is pruned at the end of iteration $\min(D, H_i, L_i)$, then it stops receiving messages. □

Theorem 3.4 *The number of messages saved by a node $v_i \in \mathcal{V}$ in a failure-free scenario is*

$$\Delta_i^{(D)} = \sum_{t=1}^{\min(D,H_i,L_i)} \sum_{l=0}^{t-1} u_i^{(l)} + \sum_{t=\min(D,H_i,L_i)+1}^{\min(D,H_i)} \left(d_i - \sum_{l=0}^{t-1} h_i^{(l)} \right).$$

Proof This is straightforward by Equations 6 and 7, and Lemma 3.3. □

It is clear that pruned nodes build very limited views as they stop interacting with others once they are pruned. These nodes would have built broader views using the YTQ method [24]. Thus if considering applications where all nodes are required to build broader views, our pruning method is not recommended.

3.3 Communication failure

We also extend our pruning method to take into account communication failures during view construction. For failure management, we simply incorporate the neighbour coordination approach proposed by Sheth et al. [19] into our pruning method. We found the coordination approach for failure management most suitable for our pruning method because each node can monitor the behaviour of its immediate neighbours and report failures and recoveries if detected, which suits our decentralised approach well. Details of the extended version of pruning with communication failure can be found in [14]. (It should be noted that we exclude details of failure management so that we can focus on the main contribution of this work).

4 Experimental investigation

For the comparison of our proposed method with the YTQ method [24] in terms of the number of messages, we consider the total and the maximum number of messages received per node. We wish to see the impact of our pruning method on message complexity in the entire network. We wish to reduce the maximum number of messages per node because if the communication time per message is the bottleneck, reducing only the total number of messages may not be helpful.

Our experiments considered the following cases, in an attempt to comprehensively test the proposed approach:

(1) Comparison of the number of messages received by nodes for each method on various networks. We also used a Wilcoxon signed-rank [23] test and the effect size [3] to verify whether the mean differences of the number of messages between pruning and the YTQ method are significantly different.
(2) Comparison of the approximated most central nodes obtained with our method to the YTQ method. A good approximation should choose a most central node with a small distance to the exact most central node. We also used a Wilcoxon signed-rank test and the effect size to verify whether the mean differences of shortest path distances between approximate central nodes obtained using the YTQ and our pruning methods with respect to the exact most central node on some random graphs are significantly different.

    In Section 3, we showed that pruning is related to node eccentricity which allows us to ensure approximation of closeness centrality using pruning because eccentricity and closeness centralities are positively and strongly correlated for various types of graphs [1, 15]. We run simulation experiments to determine the Spearman's ρ [21] and Kendall's τ [9] coefficients between eccentricity and closeness centralities. We consider the Spearman's ρ and Kendall's τ coefficients because they are appropriate correlation coefficients to measure the correspondence between two rankings.

For the hypothesis tests, the significance level we use is 0.01.

4.1 Experimental setup

We implemented our pruning method using Python and `NetworkX` [7]. Our simulation was run on two HPC (High Performance Computing) clusters hosted by Stellenbosch University. Our code can be found at `https://bitbucket.org/jmf-mas/codes/src/master/network`.

We ran several simulations with random graphs (generated as discussed below), as well as some real-world networks.

4.1.1 Randomly generated networks

We used a 200x200 grid with integer coordinates and generated 50 random connected undirected graphs as follows: We generated N uniformly distributed grid locations (sampling without replacement) as nodes. The number of nodes, N, was sampled uniformly from $[50, 500]$. Two nodes were connected by an edge if the Euclidean distance between them was less than a specified communication range $d = 8$. The number of edges and the diameter for these graphs were in the intervals $[50, 2000]$ and $[20, 60]$ respectively—see Figure 2.

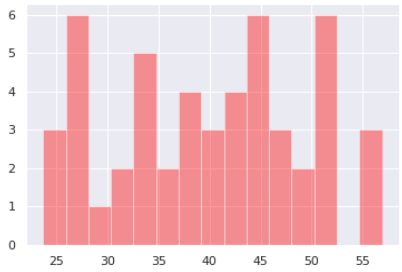


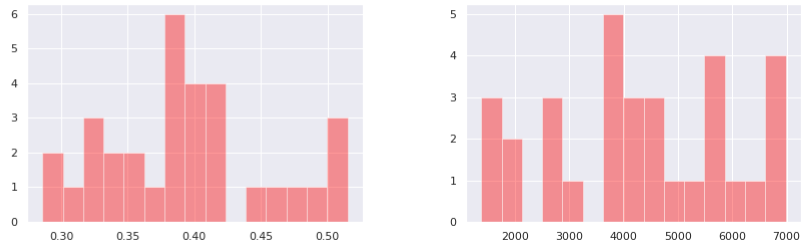
Figure 2: *Diameters of random graphs. We use a binwidth of* 2.

4.1.2 Real-world networks

The 34 real-world graphs we consider are a phenomenology collaboration network [13], a snapshot of the Gnutella peer-to-peer network [13], and 32 *autonomous graphs* [12]. The phenomenology collaboration network represents research collaborations between authors of scientific articles submitted to the Journal of High Energy Physics. In the Gnutella peer-to-peer network, nodes represent hosts in the Gnutella network and edges represent connections between the Gnutella hosts. Autonomous graphs are graphs composed of links between Internet routers. These graphs represent communication networks based on Border Gateway Protocol logs. Some characteristics of some of these networks are given in Table 2.

4.2 Results and discussion

4.2.1 Average and maximum number of messages



(a) (b)

Figure 3: *Differences in the number of messages between the YTQ method and our pruning method on the* 32 *autonomous graphs. Positive values indicate that our pruning method outperfoms the YTQ method.* ***(3a)****: Reduction in the average number of messages.* ***(3b)****: Reduction in the maximum number of messages.*

Our experiments illustrate the improved communication performance over the YTQ method in [24] resulting from pruning. Figure 3 shows differences in the

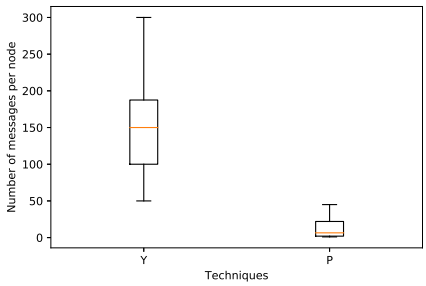


Figure 4: *Numbers of messages per node using each method on a random graph with* $D = 10$. *On the horizontal axis Y and P denote the YTQ and pruning methods respectively.*

averages and in the maximum number of messages per node between the YTQ and our pruning methods on the 32 autonomous graphs. We see that the approach reduced communication by $30 - 50\%$ on average for all network, with over 75% of networks reducing their maximum number of messages by 30% or more.

Nodes	Edges	Diameter	Y	P	Ymax	Pmax
Three autonomous networks						
1486	3422	9	*28.0*	8.8	*7005*	1413
2092	4653	9	*27.2*	7.9	*3312*	552
6232	13460	9	*25.4*	6.9	*7295*	1459
Phenomenology collaboration network						
10876	39994	9	*52.2*	14.3	*721*	120
Gnutella peer-to-peer network						
9877	25998	13	*63.0*	8.3	*3705*	787

Table 2: *Graph properties and the number of messages with each approach for five real-world networks (the phenomenology collaboration network, the Gnutella peer-to-peer network, and three autonomous networks). Y(Ymax) and P(Pmax) denote average(maximum) number of messages for the YTQ and pruning methods respectively.*

Table 2 shows the average and the maximum number of messages per node for five real-world networks respectively. The number of messages per node for each technique on one random network are contrasted in Figure 4. The results confirm that the pruning method is better than the YTQ method (Figure 3).

Hypothesis test

We observed a p-value of 7.96×10^{-90} and an effect size of 21.1140 between the number of messages obtained with our pruning and the YTQ methods on 50 random graphs containing 500 nodes each. The p-value is less than the threshold 0.01, so the means of the number of messages using the YTQ method against pruning are significantly different. In terms of effect size, according to the

classification in Gail and Richard [5], the effect size between our pruning and the YTQ methods is large ($e \geq 0.8$). So the means of the number of messages using the YTQ method against pruning differ markedly.

We also observed a reduction in total running time and memory usage from our approach. So no adverse effect on power usage from the approach.

4.2.2 Quality of selected most central node

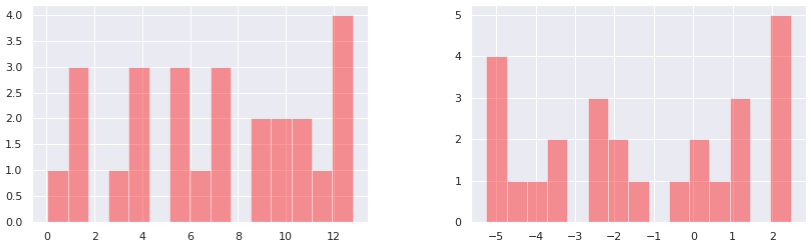
First, for various graphs considered here, the averages and standard deviations of the Spearman's ρ and Kendall's τ coefficients between eccentricity and closeness centralities are 0.9237 ± 0.0508 and 0.7839 ± 0.0728 respectively, and the correlation coefficients were all positive. This shows that there is predominantly a fairly strong level of correlation between eccentricity and closeness centralities for various graphs considered here. This confirms the results by Batool and Niazi [1], and Meghanathan [15]. Note that we chose real-world graphs and random graphs modelled on what might be realistic for a sensor network—so no attempt to choose graphs from models with high correlation.

Tables 3 shows the shortest path distances between the exact most central node and approximated most central nodes using our pruning method and the YTQ method [24] for two random graphs. Figure 5 shows differences of shortest path distances between the exact most central node and approximated central nodes obtained with the YTQ and our pruning methods on two random graphs. For each of the graphs, we randomly vary D in a range of values smaller than the diameter of the graph.

D	$\mathbf{Y}_1$	$\mathbf{P}_1$	$\mathbf{Y}_2$	$\mathbf{P}_2$
2	5	5	14	14
6	14	10	19	5
10	13	1	20	0
14	14	4	19	2
18	8	0	0	0
22	0	0	0	2
26	0	0	0	0

Table 3: *Shortest path distances between the exact most central node and approximated most central node using pruning and the YTQ method. We use two random graphs, one with* 70 *nodes and diameter of* 35*, and another with* 72 *nodes and diameter of* 32*. One method achieves better approximations than another if the distance of the selected node from the true most central node is smaller. Y_i and P_i indicate shortest path distances for the YTQ and pruning methods on the i-th random graph respectively.*

The evaluation of node centrality based on a limited view of the communication graph has an impact on the choice of the most central node. When using our



(a) (b)

Figure 5: *Differences of shortest path distances between approximated central nodes obtained using the YTQ and our pruning methods with respect to the exact most central node on two random graphs:* ***(5a)****: the first (diameter of* 26*,* 125 *nodes and* 180 *edges) and* ***(5b)****: second (diameter of* 36*,* 289 *nodes and* 597 *edges)random graphs. Positive values indicate that our method is better than the YTQ method.*

pruning and the YTQ methods to choose a leader based on closeness centrality, the methods can yield different results under the same conditions. We found that (Table 3 and Figure 5) our pruning method generally gave better approximations to closeness centrality than the YTQ method when D is considerably smaller than the diameter, with the results of the YTQ method improving as D increases. This supports our claim that our pruning method effectively identifies nodes which should not be chosen as leaders as they are highly unlikely to have the highest closeness centrality. Note that, even though the two methods sometimes give the exact most central nodes for some D (for example for $D = 26$ in Table 3), these exact most central nodes are not guaranteed.

The reason why the YTQ method yields poor results when D is smaller than the diameter of the graph is as follows. When some of the nodes have different views of the communication graph and each evaluates its closeness centrality based only on its own view, a node with small but unknown exact closeness centrality may have a high estimated closeness centrality. This can lead to poor conclusions. In the YTQ method, the central node is selected from all nodes. The advantage of the pruning method is that only unpruned nodes compute their approximate closeness centralities, i.e. the many nodes that are pruned are no longer candidates for central nodes. This reduces the chance of yielding poor performance as the central node is selected from a shorter list of candidates, i.e. the unpruned nodes.

Hypothesis test

We observed a p-value of 0.1197 between the results obtained with our pruning and the YTQ methods on 50 random graphs of 500 nodes each. The p-value is greater than the threshold 0.01, so there is no significant difference between the means for the two approaches. This means that the qualities of the selected most central nodes using both methods are almost the same. This is beneficial to pruning—though they both provide almost the same qualities of selected most central nodes, pruning reduces the number of messages significantly compared to the YTQ method [24].

5 Conclusion

We proposed an enhancement to a benchmark method [24] for view construction. The main motivation of this enhancement was to reduce the amount of communication: we aim to reduce the number of messages exchanged between nodes during interaction. Given a network, some nodes can be identified early as being unlikely to be central nodes. Our main contribution was noting that we can identify such nodes and reduce communication by pruning them.

Our proposed method improves the benchmark method in terms of number of messages. We found that reduction of the number of messages has a positive impact on running time and memory usage [14].

Future work. Our message counting model ignores the fact that in large networks, messages comprise multiple packets. Analysis of the savings of our approach in terms of the actual amount of data communicated could be investigated in future. Further, it may be possible to identify further types of prunable nodes and consider richer classes of graphs for assessment.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

In this manuscript authors have proposed a new distributed method for determination of central nodes of a network using centrality metrics. The proposed method outperforms other techniques in terms of number of messages exchanged during interaction. The method has also a positive impact on running time and memory usage.

Author details

[1]Division of Computer Science, University of Kinshasa, Kinshasa, Dem. Rep. of Congo. [2]Data Science Unit, Fimproso SARL, Kinshasa, Dem. Rep. of Congo.

References

1. K. Batool and M. A. Niazi. Towards a Methodology for Validation of Centrality Measures in Complex Networks. *PloS one*, 9(4):e90283, 2014.
2. A. Bavelas. Communication Patterns in Task-Oriented Groups. *The Journal of the Acoustical Society of America*, 22(6):725–730, 1950.

3. J. Cohen. The Statistical Power of Abnormal-Social Psychological Research: A Review. *The Journal of Abnormal and Social Psychology*, 65(3):145, 1962.
4. G. F. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Pearson Education, 2005.
5. M. S. Gail and F. Richard. Using Effect Size–or Why the P Value Is Not Enough. *Journal of Graduate Medical Education*, 4(3):279–282, 2012.
6. F. Garin, D. Varagnolo, and K. H. Johansson. Distributed Estimation of Diameter, Radius and Eccentricities in Anonymous Networks. *IFAC Proceedings Volumes*, 45(26):13–18, 2012.
7. A. Hagberg, D. Schult, P. Swart, D. Conway, L. Séguin-Charbonneau, C. Ellison, B. Edwards, and J. Torrents. Networkx. High Productivity Software for Complex Networks. *Webová strá nka https://networkx. lanl. gov/wiki*, 2013.
8. P. Hage and F. Harary. Eccentricity and Centrality in Networks. *Social Networks*, 17(1):57–63, 1995.
9. M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.
10. C. Kim and M. Wu. Leader Election on Tree-Based Centrality in Ad Hoc Networks. *Telecommunication Systems*, 52(2):661–670, 2013.
11. S. Lam and M. Reiser. Congestion Control of Store-and-Forward Networks by Input Buffer Limits–an Analysis. *IEEE Transactions on Communications*, 27(1):127–134, 1979.
12. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 177–187. ACM, 2005.
13. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph Evolution: Densification and Shrinking Diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.
14. J. F. Masakuna. *Active Strategies for Coordination of Solitary Robots*. PhD thesis, Stellenbosch University, 2020.
15. N. Meghanathan. Correlation Coefficient Analysis of Centrality Metrics for Complex Network Graphs. In *Computer Science On-line Conference*, pages 11–20. Springer, 2015.
16. A. Naz. *Distributed Algorithms for Large-Scale Robotic Ensembles: Centrality, Synchronization and Self-Reconfiguration*. PhD thesis, Université Bourgogne Franche-Comté, 2017.
17. M. E. Nelson and M. A. MacIver. Sensory Acquisition in Active Sensing Systems. *Journal of Comparative Physiology A*, 192(6):573–586, 2006.
18. R. J. Ramírez and N. Santoro. Distributed Control of Updates in Multiple-Copy Databases: A Time Optimal Algorithm. In *Proceedings of the 4th Berkeley Conference on Distributed Data Management and Computer Networks (Berkeley, Calif, Aug.)*, pages 191–207, 1979.
19. A. Sheth, C. Hartung, and R. Han. A Decentralized Fault Diagnosis System for Wireless Sensor Networks. In *IEEE International Conference on Mobile Adhoc and Sensor Systems*, pages 1–3. IEEE, 2005.
20. S. S. Skiena. *The Algorithm Design Manual*, volume 1. Springer Science & Business Media, 1998.
21. C. Spearman. "General Intelligence" Objectively Determined and Measured. *American Journal of Psychology*, 15:663–671, 1961.
22. T. Tomic, K. Schmid, P. Lutz, A. Domel, M. Kassecker, E. Mair, I. L. Grixa, F. Ruess, M. Suppa, and D. Burschka. Toward a Fully Autonomous UAV: Research Platform for Indoor and Outdoor Urban Search and Rescue. *IEEE Robotics & Automation Magazine*, 19(3):46–56, 2012.
23. F. Wilcoxon. Individual Comparisons by Ranking Methods. In *Breakthroughs in Statistics*, pages 196–202. Springer, 1992.
24. K. You, R. Tempo, and L. Qiu. Distributed Algorithms for Computation of Centrality Measures in Complex Networks. *IEEE Transactions on Automatic Control*, 62(5):2080–2094, 2017.

Algorithm 1 *Our presentation of the YTQ method in [24]. The algorithm gives the code run on a single node v_i. An example of pseudocode is given in the procedure* runDistributedBFS.

```
1: procedure runDistributedYTQ(N_i, D)
2:     YTQObject←distributedYTQ(N_i, D)
3:     while not YTQObject.isEnded() do
4:         YTQObject.oneHop()
5:         YTQObject.update()
6:     end while
7: end procedure
8:
9: class distributedYTQ
10:    Class variables
11:    D                 the pre-set maximum number of iterations
12:    M_i               a queue used for messages received by the node v_i
13:    N_i               a list of immediate neighbours of the node v_i
14:    N_i^(t)           the set of (t+1)-hop neighbours of v_i
15:    N_{i,t}           the set nodes known by v_i until the end of iteration t
16:    c_i               the closeness centrality of node v_i
17:    δ_i               a variable used for computation of c_i
18:    t                 the current iteration number
19:    T                 the actual maximum number of iterations
20: ---------------------------------------------------------------------
21:    constructor (N_i, D)
22:        (t, D, δ_i) ← (0, D, |N_i|)
23:        N_i^(0) ← {v_j : v_j ∈ N_i}
24:        N_{i,i} ← N_i^(0).clone()
25:    end constructor
26:
27:    procedure oneHop()
28:        if not isEnded() then
29:            for v_j ∈ N_i do
30:                v_i sends ⟨NeighbouringMessage(i, N_i^(t−1))⟩ to v_j
31:            end for
32:        else
33:            T ← min(t, D)
34:            c_i ← closenessCentrality(T)
35:        end if
36:    end procedure
37:
38:    procedure update()
39:        if not isEnded() then
40:            t ← t + 1
41:            N_i^(t) ← ∅
42:            while M_i.size() ≥ 1 do
43:                (j, N_j^(t−1)) ← M_i.dequeue()
44:                N_i^(t) ← N_i^(t) ∪ N_j^(t−1)        ▷ "v_i fuses the messages received"
45:            end while
46:            N_i^(t) ← N_i^(t) \ N_{t−1,i}
47:            N_{t,i} ← N_{t−1,i} ∪ N_i^(t)
48:            δ_i ← δ_i + t|N_i^(t)|
49:        end if
50:    end procedure
51:
52:    function closenessCentrality(T)
53:        return (|N_{i,T}| − 1) / δ_i
54:    end function
55:
56:    function isEnded()
57:        return N_i^(t) = ∅ or t = D
58:    end function
59: end class
```

Algorithm 2 *Our proposed pruning method in a failure-free scenario. The algorithm gives the code executed for a single node v_i. An example of pseudo code is given in the procedure* RUNPRUNING.

```
1: procedure runPruning(N_i, D)
2:     PruningObject←pruning(N_i, D)
3:     PruningObject.initialOneHop()
4:     PruningObject.initialUpdate()
5:     PruningObject.firstPruningDetection()
6:     while not PruningObject.isEnded() do
7:         PruningObject.nextOneHop()
8:         PruningObject.nextUpdate()
9:     end while
10: end procedure
11:
12: class pruning
13:     Class variables
14:     D          the pre-set maximum number of iterations
15:     F_i^(t)    set of pruned nodes known by v_i at the end of iteration t
16:     F_{i,t}    set of pruned nodes known by v_i at the end of iteration t
17:     M_i        a message queue for messages received by the node
18:     N_i        set of immediate neighbours of v_i
19:     N_{i,t}^up set of neighbours of v_i which are still active up to iteration t
20:     Q_ij       the one-hop neighbours of node v_j sent to v_i at the first iteration
21:     N_i^(t)    set of new nodes discovered by v_i at the end of iteration t
22:     N_{i,t}    view of communication graph of v_i up to iteration t
23:     c_i        the closeness centrality of node v_i
24:     δ_i        a variable used for computation of c_i
25:     t          current iteration number
26:     T          the actual maximum number of iterations
27: ---------------------------------------------------------------------
28:     constructor (N_i, D)
29:         (t, D, δ_i) ← (0, D, |N_i|)
30:         N_i^(0) ← {v_j : ∀v_j ∈ N_i}
31:         N_{i,0} ← N_i^(0).clone()                ▷ "v_i detects its immediate neighbours"
32:     end constructor
33:
34:     function initialOneHop()
35:         for v_j ∈ N_i do
36:             v_i sends ⟨NeighbouringMessage(i, N_i^(0))⟩ to v_j
37:         end for
38:     end function
39:
40:     procedure initialUpdate()
41:         N_i^(1) ← ∅
42:         t ← t + 1
43:         while M_i.size() ≥ 1 do
44:             (j, N_j^(0)) ← M_i.dequeue()
45:             N_i^(1) ← N_i^(1) ∪ N_j^(0)          ▷ "v_i fuses the messages received"
46:             Q_ij ← N_j^(0)
47:         end while
48:         N_i^(1) ← N_i^(1) \ N_{i,0}
49:         N_{i,1} ← N_{i,0} ∪ N_i^(t)
50:         δ_i ← δ_i + t|N_i^(1)|
51:     end procedure
52:
53:     procedure firstPruningDetection()
54:         leavesDetection()
55:         triangleDetection()
56:         F_{i,t} ← F_i^(t).clone()
57:     end procedure
58:
59:     procedure leavesDetection()                  ▷ "v_i detects leaves in its neighbourhood
60:         F_i^(1) ← ∅
61:         for v_j ∈ N_i ∪ {v_i} do
62:             if |Q_ij| = 1 then
```

```
63:                 F_i^(1) ← F_i^(1) ∪ {v_j}
64:             end if
65:         end for
66:     end procedure
67:
68:     procedure TRIANGLEDETECTION                ▷ "v_i detects elements causing triangles in its neighbourhood
69:         for v_j ∈ N_{i,1}^up ∪ {v_i} do
70:             if |Q_ij| = 2 then
71:                 Let v_f, v_g be the two immediate neighbours of v_j
72:                 if v_f ∈ Q_ig then
73:                     F_i^(1) ← F_i^(1) ∪ {v_j}
74:                 end if
75:             end if
76:         end for
77:     end procedure
78:
79:     procedure NEXTONEHOP()
80:         if not ISENDED() then
81:             N_{i,t}^up ← N_{i,t}^up \ F_{i,t}
82:             for v_j ∈ N_{i,t}^up do
83:                 v_i sends ⟨NeighbouringMessage(i, N_i^(t−1))⟩ to v_j
84:             end for
85:         end if
86:     end procedure
87:
88:     procedure NEXTUPDATE()
89:         if not ISENDED() then
90:             t ← t + 1
91:             N_i^(t) ← ∅
92:             while M_i.size() ≥ 1 do
93:                 (j, N_j^(t−1)) ← M_i.dequeue()
94:                 N_i^(t) ← N_i^(t) ∪ N_j^(t−1)          ▷ "v_i fuses the messages received"
95:             end while
96:             N_i^(t) ← N_i^(t) \ N_{i,t−1}
97:             N_{i,t} ← N_{i,t−1} ∪ N_i^(t)
98:             FURTHERPRUNINGDETECTION()
99:             F_{i,t} ← F_{i,t} ∪ F_i^(t)
100:            δ_i ← δ_i + t|N_i^(t)|
101:        else
102:            T ← min(t, D)
103:            c_i ← CLOSENESSCENTRALITY(T)
104:        end if
105:    end procedure
106:
107:    procedure FURTHERPRUNINGDETECTION()      ▷ "v_i detects elements of F_i^(t) in its neighbourhood"
108:        F_i^(t) ← ∅
109:        for v_j ∈ N_{i,t}^up do
110:            if N_j^(t) ⊆ N_{i,t−1} then
111:                F_i^(t) ← F_i^(t) ∪ {v_j}
112:            end if
113:        end for
114:        if |N_{i,t}^up| = 1 and N_i^(t) ≠ ∅ then
115:            F_i^(t) ← F_i^(t) ∪ {v_i}
116:        end if
117:    end procedure
118:
119:    function CLOSENESSCENTRALITY(T)
120:        if v_i ∈ F_{i,T} then
121:            return 0
122:        end if
123:        return (|N_{i,T}| − 1) / δ_i
124:    end function
125:
126:    function ISENDED()
127:        return t = D or N_i^(t) = ∅ or v_i ∈ F_{i,t}
128:    end function
129: end class
```